



John Fulcher
Lakhmi C. Jain (Eds.)

Computational Intelligence: A Compendium

John Fulcher and Lakhmi C. Jain (Eds.)

Computational Intelligence: A Compendium

Studies in Computational Intelligence, Volume 115

Editor-in-chief

Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springer.com

Vol. 94. Arpad Kelemen, Ajith Abraham and Yuehui Chen (Eds.)
Computational Intelligence in Bioinformatics, 2008
ISBN 978-3-540-76802-9

Vol. 95. Radu Dogaru
Systematic Design for Emergence in Cellular Nonlinear Networks, 2008
ISBN 978-3-540-76800-5

Vol. 96. Aboul-Ella Hassanien, Ajith Abraham and Janusz Kacprzyk (Eds.)
Computational Intelligence in Multimedia Processing: Recent Advances, 2008
ISBN 978-3-540-76826-5

Vol. 97. Gloria Phillips-Wren, Nikhil Ichalkaranje and Lakhmi C. Jain (Eds.)
Intelligent Decision Making: An AI-Based Approach, 2008
ISBN 978-3-540-76829-9

Vol. 98. Ashish Ghosh, Satchidananda Dehuri and Susmita Ghosh (Eds.)
Multi-Objective Evolutionary Algorithms for Knowledge Discovery from Databases, 2008
ISBN 978-3-540-77466-2

Vol. 99. George Meghabghab and Abraham Kandel
Search Engines, Link Analysis, and User's Web Behavior, 2008
ISBN 978-3-540-77468-6

Vol. 100. Anthony Brabazon and Michael O'Neill (Eds.)
Natural Computing in Computational Finance, 2008
ISBN 978-3-540-77476-1

Vol. 101. Michael Granitzer, Mathias Lux and Marc Spaniol (Eds.)
Multimedia Semantics - The Role of Metadata, 2008
ISBN 978-3-540-77472-3

Vol. 102. Carlos Cotta, Simeon Reich, Robert Schaefer and Antoni Ligeza (Eds.)
Knowledge-Driven Computing, 2008
ISBN 978-3-540-77474-7

Vol. 103. Devendra K. Chaturvedi
Soft Computing Techniques and its Applications in Electrical Engineering, 2008
ISBN 978-3-540-77480-8

Vol. 104. Maria Virvou and Lakhmi C. Jain (Eds.)
Intelligent Interactive Systems in Knowledge-Based Environment, 2008
ISBN 978-3-540-77470-9

Vol. 105. Wolfgang Guenther
Enhancing Cognitive Assistance Systems with Inertial Measurement Units, 2008
ISBN 978-3-540-76996-5

Vol. 106. Jacqueline Jarvis, Dennis Jarvis, Ralph Rönnquist and Lakhmi C. Jain (Eds.)
Holonic Execution: A BDI Approach, 2008
ISBN 978-3-540-77478-5

Vol. 107. Margarita Sordo, Sachin Vaidya and Lakhmi C. Jain (Eds.)
Advanced Computational Intelligence Paradigms in Healthcare - 3, 2008
ISBN 978-3-540-77661-1

Vol. 108. Vito Trianni
Evolutionary Swarm Robotics, 2008
ISBN 978-3-540-77611-6

Vol. 109. Panagiotis Chountas, Ilias Petrounias and Janusz Kacprzyk (Eds.)
Intelligent Techniques and Tools for Novel System Architectures, 2008
ISBN 978-3-540-77621-5

Vol. 110. Makoto Yokoo, Takayuki Ito, Minjie Zhang, Juhnyoung Lee and Tokuro Matsuo (Eds.)
Electronic Commerce, 2008
ISBN 978-3-540-77808-0

Vol. 111. David Elmakias (Ed.)
New Computational Methods in Power System Reliability, 2008
ISBN 978-3-540-77810-3

Vol. 112. Edgar N. Sanchez, Alma Y. Alanís and Alexander G. Loukianov
Discrete-Time High Order Neural Control: Trained with Kalman Filtering, 2008
ISBN 978-3-540-78288-9

Vol. 113. Gemma Bel-Enguix, M. Dolores Jimenez-Lopez and Carlos Martín-Vide (Eds.)
New Developments in Formal Languages and Applications, 2008
ISBN 978-3-540-78290-2

Vol. 114. Christian Blum, María José Blesa Aguilera, Andrea Roli and Michael Sampels (Eds.)
Hybrid Metaheuristics, 2008
ISBN 978-3-540-78294-0

Vol. 115. John Fulcher and Lakhmi C. Jain (Eds.)
Computational Intelligence: A Compendium, 2008
ISBN 978-3-540-78292-6

John Fulcher
Lakhmi C. Jain
(Eds.)

Computational Intelligence: A Compendium

With 321 Figures and 67 Tables

 Springer

Prof. John Fulcher
School of Computer Science
and Software Engineering
Faculty of Informatics
University of Wollongong
Northfields Ave
Wollongong NSW 2522
Australia
john@uow.edu.au

Prof. L.C. Jain
Knowledge-Based Engineering
Founding Director of the KES Centre
SCT-Building
Mawson Lakes Campus
University of South Australia
Adelaide South Australia SA 5095
Australia
Lakhmi.jain@unisa.edu.au

ISBN 978-3-540-78292-6

e-ISBN 978-3-540-78293-3

Studies in Computational Intelligence ISSN 1860-949X

Library of Congress Control Number: 2008922060

© 2008 Springer-Verlag Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: Deblik, Berlin, Germany

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Dedicated to the creative spark within us all

Preface

At this point in time, Computational Intelligence (CI) has yet to mature as a discipline in its own right. Accordingly, there is little consensus as to a precise definition of this emerging field. Nevertheless, most practitioners would include Artificial Neural Network (ANN), Fuzzy and evolutionary techniques (and perhaps others), and more especially *hybrids* of these (this will be expanded upon in Chap.1.) Our emphasis in this Compendium is very much on *applied* methods – ones which have been tired-and-proven effective on real-world problems.

The 25 chapters have been grouped into the following ten themes (Parts):

- I. Overview, Background
- II. Data Preprocessing, Systems Integration & Visualization
- III. Artificial Intelligence
- IV. Logic and Reasoning
- V. Ontology
- VI. Intelligent Agents
- VII. Fuzzy Systems
- VIII. Artificial Neural Networks
- IX. Evolutionary Approaches
- X. DNA and Immunity-based Computing

This grouping is not the only one we could have used – indeed some chapters could have just as easily appeared in alternate Parts of the Handbook. For example, Lam & Lee’s iJADE tourist guidance system could just as readily been grouped into Part-VI (Agents) as Part-V (Ontology); likewise, Fyfe’s Topographic Maps would have fitted just as well into Part-VIII (ANNs), Ishibuchi et al. would have been just as equally well placed in either Part-VII or Part-IX, and Islam & Yao could have fitted equally well into Parts VIII or IX. Nevertheless, we have attempted to group together chapters with common foci.

Returning briefly to the question of real-world applications, Table 1 summarizes those covered in the chapters herein.

Table 1. Chapter methods and applications

Chapter	Method	Application(s)	Data set(s)
2	REDR; SOM; Mitra Multi-Scale	handwritten digits; yeast; synthetic	UCI KDD
3	SOM; GTM; HaToM; ToPE	data visualization	UCI-ML (algae; wine)
4	networked selfish agents; probabilistic cellular automata	self-maintenance/repair; 'internet being'	Denial-Of-Service & Traceroute websites
5	affective embodied agent	synthetic therapist; human behaviour change	—
6	paraconsistent annotated logic	pipeline safety process verification	—
7	Data-Oriented Parsing	Natural Language Processing; musical notation, physics problem solving	Penn Treebank; Essen Folksong Collection
8	Conceptual Graph Theory	ontologies: architectural design; air operations officer	—
9	mobile/GPS agents; ontologies	tourist guidance system	DAML; Protégé
10	Agent-Based Modelling software	'Stupid' model	Santa Fe Institute artificial stock market
11	agents; Peer-to-Peer & cluster computing	resource allocation; communication; scheduling	—
12	multi-agents; dynamic clustering	sensor networks	—
13	agents; ANN; SVM; Evolutionary Comp.	computational economics; foreign exchange rates	SFI and AI-ECON artificial stock markets
14	reconciliation via optimization	fuzzy rule-based systems	—
15	Evolutionary multi-objective design	fuzzy rule-based classifiers	UCI-ML (breast; glass; heart; iris; wine)
16	ANNs; MAS	network bandwidth prediction	—
17	SOM; ViSOM; SOMN; kernel methods	vector quantization; image compression/ segmentation; text mining	UCI-ML (iris; yeast)
18	neural systems engineering	Blue Brain; SPINN; SpiNNaker	—
19	GenNt; CGA; FPGA	'artificial brain'; UXO robot	—
20	Evolutionary ANN ensembles	Credit Card; Diabetes; Heart Disease; Glass; Letter; Soybean; Breast Cancer	UCI-ML
21	genetic simulated annealing	graph colouring; bin packing; timetabling	DIMACS; Scholl & Klein; Falkenauer
22	Genetic Programming (GP)	modeling; the 'Humies'; image/signal processing; time series prediction; et al.	—

Table 1. (continued)

Chapter	Method	Application(s)	Data set(s)
23	Particle Swarm Optimization	finding origin; Rastrigin/Schwefel functions; timetabling	—
24	DNA computing	multiple elevator scheduling	—
25	immunity-based computing	stable marriage problem; auto sensor diagnosis; noise neutralization	—

Chapters commence with an overview of the field in question, and conclude with a *Resources Appendix*. These resources cover key references (classic texts, survey articles, key papers), pertinent journals, professional societies/organizations and research groups, international conferences and workshops, and/or electronic/on-line material (with a particular emphasis on databases and Open Source software). Each chapter is thus complete unto itself for those readers wanting to explore just a single topic from the many on offer. In this mode, the Compendium could be used as a text for graduate programs in Artificial Intelligence, Intelligent Systems, Soft Computing, Computational Intelligence, and the like. The *Index* doubles as a Glossary of Terms (with acronyms shown in parentheses).

We have gathered together in a single volume chapters by leading experts in their respective fields – all of international repute, as evidenced (in part) by the following:

- I. Learned Society Fellows: *IEEE* (Furber, Pedrycz, Yao); *Intl. Fuzzy Systems Association* (Pedrycz); *Royal Society* (Furber); *Royal Academy of Engineering* (Furber); *British Computer Society* (Furber); *Intl. Society for Genetic and Evolutionary Computation* (Koza, Langdon, Poli); *Institution of Engineers, Australia* (Jain)
- II. Editors-in-Chief: *Computing and Information Systems* (Fyfe); *IEEE Trans. Evolutionary Computing* (Yao); *Information Sciences* (Pedrycz); *Intelligent Decision Technologies* (Jain); *Intl. J. Hybrid Intelligent Systems* (Jain); *Intl. J. Knowledge-based Intelligent Engineering Systems (Founding Editor)* (Jain); *Intl. J. Logic & Reasoning* (Nakamatsu); *Intl. J. Metaheuristics* (Mumford); *New Mathematics & Natural Computing* (Chen)
- III. Associate/Area Editors: *Advances in Natural Computation* (Yao); *Computer and Information Systems* (Jain); *Evolutionary Computation* (Poli); *IEEE Computational Intelligence Magazine* (Ishibuchi); *IEEE Trans. Evolutionary Computation* (Ishibuchi); *IEEE Trans. Fuzzy Systems* (Ishibuchi, Pedrycz); *IEEE Trans. Knowledge and Data Engineering* (Wang); *IEEE Trans. Neural Networks* (Pedrycz, Wang); *IEEE Trans. Neural Networks* (Chen, Wang); *IEEE Trans. Systems, Man and*

- Cybernetics* (Ishibuchi, Jain, Pedrycz); *Intl. J. Advances in Fuzzy Systems* (Ishibuchi); *Intl. J. Applied Intelligence* (Hendtlass); *Intl. J. Computational Intelligence Research* (Ishibuchi, Poli); *Intl. J. Information Technology* (Chow); *Intl. J. Knowledge-based Intelligent Engineering Systems* (Ishida); *Intl. J. Metaheuristics* (Ishibuchi); *Intl. J. Pattern Recognition and Artificial Intelligence* (Jain); *J. Artificial Evolution and Applications* (McPhee); *J. Economic Research* (Chen); *J. Genetic Programming and Evolvable Machines* (Poli); *J. Intelligent and Fuzzy Systems* (Jain); *Mathware and Soft Computing* (Ishibuchi); *Neural Computing and Applications* (Jain); *Soft Computing J.* (Ishibuchi)
- IV. Book Series Editors: *CRC Press (Intl. series on CI)* (Jain); *IGI (CI: Theory and Applications series)* (Fyfe, Jain); *Kluwer (Genetic Programming)* (Koza); *Springer (Advanced Information and Knowledge Processing)* (Jain)
- V. Invited Keynote/Plenary Conference Speakers: Chen, deGaris, Fulcher, Fyfe, Ishida, Jain, Koza, Nakamatsu, Prokopenko, Wang, Yao
- VI. Awards: *Royal Society Wolfson Research Merit Award* (Furber); *IET Faraday Medal* (Furber); *Queen's Award for Technology* (Furber); *Japanese Society for the Promotion of Science Prize* (Ishibuchi); *IEEE Donald G. Fink Prize Paper* (Yao); *AJB CEBIT Exhibition Award (Most Innovative Technology)* (Prokopenko); *Japanese Society for Artificial Intelligence Award* (Prokopenko); *Best Paper Awards* (Chow, Langdon, MCPhee, Poli)
- VII. Advisory Boards: *UK EPSRC Peer Review College* (Poli); *EU Expert Evaluator* (Poli)

Indeed, this Handbook is a truly international undertaking, with a total of 43 authors from 10 different countries contributing (9 in Australia; 19 in Asia; 11 in UK; and 4 in North America). All chapters were peer reviewed to ensure a uniformly high standard for the Handbook overall.

Next follows an overview of each Chapter.

In Part-I, Fulcher introduces fundamental Computational Intelligence concepts. This Introductory chapter is intended to serve both as background reading (tutorial) for students/newcomers to the field, as well as setting the scene/laying the groundwork for the more specialist chapters that follow.

A critical consideration with applying *any* CI approach to real-world problems is data pre-processing. Part-II begins with a chapter by Chow & Huang on Data Reduction for Pattern Recognition. They provide an overview of ‘filter’ and ‘wrapper’ methods, before focusing on their Representative Entropy Data Reduction (REDR) approach. In their discussion they cover not only *data reduction* but also *feature selection* – another key aspect of pre-processing.

Fyfe’s concern in Chap. 3 is data visualization. He illustrates how the Self-Organizing Map, Generative and Harmonic Topographic Maps, and the

Topographic Product-of-Experts can all be effectively used on the UCI-ML algae and wine data sets.

Ishida adopts a game-theoretic approach to self-maintenance and repair in large-scale, complex systems such as the Internet – more specifically, ‘selfish’ agents and Probabilistic Cellular Automata. Emphasis is placed on the emergence of intelligent systems at the Nash Equilibrium in such networks. He concludes with speculation as to the conditions necessary for the emergence of a so-called ‘Internet Being’, in the context of such ‘selfishware’.

Part-III covers conventional (traditional) AI, but from a slightly different perspective. In Chap.5 Creed and Beale provide a fascinating insight into Emotional Intelligence and affective computing. They commence with a discussion of emotion theory, before introducing ‘affective embodied’ agents. The authors then demonstrate how such agents can be applied in the real world to engender behavioural change in humans, through the medium of a so-called ‘simulated (artificial) therapist’.

Part-IV covers logic, reasoning and related approaches to CI. In Chap.6 Nakamatsu provides a comprehensive discussion of various paraconsistent annotated logics, including his own extended vector-annotated logic program with strong negation – EVALPSN; defeasible reasoning proofs are also included. Nakamatsu then proceeds to show how EVALPSN can be applied to pipeline process safety verification, and by extension to pipeline process *order* safety verification (by way of before-after EVALPSN).

Bod uses the supervised, corpus-based probabilistic Data-Oriented Parsing approach to reveal the underlying structures in areas as diverse as Natural Language Processing, the melodic analysis of musical scores, and physics problem solving. A mechanism for determining the optimum parse tree is described, and the chapter concludes with an explanation of how DOP could be modified to support unsupervised learning.

Part-V of the Handbook covers ontology, which is often closely related to intelligent agents (Part-VI). In Chap.8, Corbett takes the view that frameworks for intelligent systems are best represented by a combination of concept type hierarchy, canonical formation rules, conformity relations and subsumption. He illustrates the validity of this approach with regard to both architectural design and air operation officer ontologies, and concludes that his automated reasoning approach could be readily extended to the Semantic Web.

In Chap.10, Lam and co-authors provide an account of an Intelligent Ontology Agent-based Tourist Guidance System, which they developed using the Intelligent Java Agent-based Development Environment (iJADE).

Part-VI is devoted to (intelligent) software agents. Standish commences with a comparative review in Chap.11 of open source Agent-based Modelling

platforms, including a performance comparison on a simple pedagogical model (the so-called ‘Stupid Model’).

Zhang and co-authors follow in Chap. 11 with a discussion of their agent-based **SmartGRID** model – incorporating both Peer-to-Peer and clustering techniques – and which can yield improved resource allocation, as well as task communication and scheduling in agent grids operating in open environments. Piraveenan and co-authors are likewise interested in grids, but in their case scale-free sensor networks. They use a dynamic, decentralized MAS algorithm for predicting convergence times for cluster formation in such grids (networks).

Chen provides a comprehensive account of agent-based computational economics (ACE) in Chap. 13, with a particular focus on Genetic Algorithms. After introducing the cobweb and overlapping generations models, he describes several applications of ACE, including inflation, foreign exchange rate, and artificial stock markets.

Fuzzy Systems are the focus of Part-VII. Pedrycz commences with a discussion of the semantics and perception of fuzzy sets and fuzzy mapping. He proceeds to show that reconciliation of fuzzy set perception and granular mappings can be expressed in the form of an optimization pattern, which in turn can be subsequently applied to rule-based systems.

In Chap. 15, Ishibushi and co-authors discuss the principles underlying the evolutionary design of fuzzy classifiers, illustrating the effectiveness of their approach by way of data sets selected from the UCI-ML repository (breast cancer; glass; heart; iris; wine).

Part-VIII covers Artificial Neural Networks. Fu and co-authors commence by illustrating how supervised, feedforward networks (MLP/BP) can be used in the data mining of Quality-of-Service aware media grids.

Yin provides a comprehensive coverage of Kohonen’s Self-Organizing Map and its more recent variants in Chap. 17. The performance of SOM, ViSOM, SOM Mixture Network and SOM kernel methods is compared on vector quantization, image compression and segmentation, density modelling, data visualization and text mining.

In Chap. 18, Furber & Temple provide an overview of neural systems engineering, namely the realization of ANNs in hardware form, rather than the more usual approach of software simulation. DeGaris follows in a similar vein, describing how Field Programmable Gate Arrays (FPGAs) can be used to build ‘artificial brains’.

Evolutionary approaches to CI are the subject of Part-IX of the Handbook. Islam & Yao commence with an account of how ANN ensembles can be first evolved, then used as classifiers on representative data sets from the UCI-ML repository.

In Chap. 21 Mumford concerns herself with a so-called memetic EA – in the form of a Genetic Simulated Algorithm (GSA) – which she proceeds to demonstrate can be used to solve set partitioning problems (graph colouring, bin packing, timetabling).

Genetic Programming is the focus of the Chapter by Langdon and co-authors. An extensive coverage of basic principles is followed by descriptions of how to apply GPs in various application domains, including hints for the novice user (‘tricks-of-the-trade’, as it were). The Chapter rounds off with a section on theoretical aspects of GP. The authors include an extensive (420 item) reference list.

In Chap. 23, after providing an overview of the basic Particle Swarm Optimization algorithm, Hendtlass then proceeds to describe enhancements to handle multiple optima, niching and ‘Waves of Swarm Particles’, before outlining how one can minimize the computational cost of such algorithms. He illustrates the effectiveness of the PSO approach by way of finding the origin, solving Rostrigin’s and Schwefel’s functions, and timetabling.

In the last Part of the Handbook we cover CI approaches inspired by Nature, but which do not fall under the umbrella of ANNs, EAs or Fuzzy. Firstly, Watada provides an overview of DNA Computing, then proceeds to show how this can be successfully applied to the problem of scheduling the movements of a group of elevators in a high-rise building.

Chapter 25 is concerned with Immunity-based computing (IBC). After introducing the basic concepts, Ishida shows how IBC can be applied to automobile sensor diagnosis, noise neutralization, and the ‘Stable Marriage Problem’. He concludes by proposing a general (immunity-based) problem solver.

One of your Editors (JF) formatted the Handbook, using `MikTeX v2.4` and `WinEdit v5.4`. In this (considerable) endeavour we are indebted to the following for their assistance along the way: Professor Philip Ogunbona (for imparting ‘the joy of LaTeX’), Associate Professor Willy Susilo, Associate Professor Russell Standish, Professor Riccardo Poli, Dr. Bill Langdon, and Jia Tang for insight into the finer points of LaTeX, as well as Nik Milosevic (for creation of .eps figures). Thanks are also due to Dr. Thomas Ditzinger, Senior Editor, and Heather King, Engineering Editorial, respectively at Springer-Verlag GmbH, as well as Srilatha Achuthan, Project Manager at SPi Technologies, Chennai.

We sincerely trust that you find much of interest in the ensuing pages.

Wollongong NSW, Australia
Adelaide SA, Australia
September 2007

*John Fulcher
Lakhmi C. Jain*

Contents

Part I Overview, Background

Computational Intelligence: An Introduction

<i>John Fulcher</i>	3
1 Introduction, Overview, Definitions	3
2 Historical Background	7
2.1 Artificial Intelligence (AI)	7
2.2 Machine Learning (ML)	14
2.3 Decision Trees	16
3 Approaches to CI	17
3.1 The Intuitive Appeal of Nature	17
3.2 Brains <i>versus</i> Computers	20
4 CI Paradigms	20
4.1 Pre-Processing	21
5 Expert Systems	21
6 Fuzzy Systems	24
7 Artificial Neural Networks	26
7.1 ANN Types	26
7.2 Multi-Layer Perceptron/BackPropagation	27
7.3 Other ANN Models	29
8 Evolutionary Methods	31
8.1 Genetic Algorithms	32
8.2 Evolutionary Programming	36
8.3 Genetic Programming	36
8.4 Swarms	36
9 Immunity-Based and Membrane-Based Computing	39
9.1 Immunity-Based Computing	39
9.2 Membrane-Based Computing	40
10 DNA Computing	40
11 Intelligent Agents	41

12	Hybrid Methods	42
13	Conclusion	48
	References	50
	Resources	67
1	Key Books	67
1.1	Computational Intelligence	67
1.2	Artificial Neural Networks	68
1.3	Evolutionary Methods	69
1.4	Fuzzy Systems	71
1.5	Other	71
2	Key Survey/Review Articles	72
2.1	Artificial Neural Networks	72
2.2	Evolutionary Methods	73
2.3	Fuzzy Systems	73
2.4	Other	73
3	Organizations, Societies, Special Interest Groups, Journals	74
3.1	Computational Intelligence	74
3.2	Artificial Neural Networks	74
3.3	Evolutionary Methods	75
3.4	Fuzzy Systems	75
3.5	Other	75
4	Key International Conferences/Workshops	76
5	(Open Source) Software	77
6	Data Bases	78

Part II Preprocessing, Visualization, Systems Integration

Data Reduction for Pattern Recognition and Data

Analysis

	<i>Tommy W.S. Chow and Di Huang</i>	<i>81</i>
1	Introduction	81
2	Data Reduction	82
2.1	Wrapper Methods	83
2.2	Filter Methods	83
2.3	Examples of Filter Methods	84
3	Feature Selection	89
3.1	Feature Evaluation	91
3.2	Search Engine	97
3.3	Example Feature Selection Models	98
4	Trends and Challenges of Feature Selection and Data Reduction ..	101
	References	103

Resources 107

1 Key Books 107

2 Key Survey/Review Articles 107

3 Organizations, Societies, Special Interest Groups 108

4 Research Groups 108

5 Discussion Groups, Forums 108

6 Key International Conferences/Workshops 108

7 (Open Source) Software 109

8 Data Bases 109

Topographic Maps for Clustering and Data Visualization

Colin Fyfe 111

1 Introduction 111

2 Clustering and Visualization 112

3 The Self-Organizing Map 113

 3.1 Competitive Learning 113

 3.2 Illustrative Example 117

 3.3 Alternative Traditional Topology Preserving Mappings 118

 3.4 A Last Word 120

4 The Generative Topographic Mapping 121

 4.1 Illustrative Examples 122

 4.2 Adjusting the Latent Space 124

 4.3 Deleting Latent Points 126

5 Topographic Product of Experts (ToPoE) 126

 5.1 Comparison with the GTM 129

 5.2 Illustrative Example 130

 5.3 Projections 131

 5.4 Growing and Pruning ToPoEs 133

 5.5 Different Noise Models 135

 5.6 Twinned ToPoEs 135

 5.7 Visualizing and Clustering Real Data Sets 136

 5.8 Discussion 139

6 Harmonic Averages 140

 6.1 Harmonic k-means 141

 6.2 The Harmonic Topographic Map 142

 6.3 Simulations 142

 6.4 Generalized Harmony Learning 144

 6.5 Conclusion 146

7 Conclusion 146

References 147

Resources 151

1 Key Books 151

2 Key Survey/Review Articles 151

3 Key Journals 152

4	Key International Conferences/Workshops	152
5	Software	153
6	Data Bases	153

Complex Systems Paradigms for Integrating Intelligent Systems: A Game Theoretic Approach

	<i>Yoshiteru Ishida</i>	155
1	Introduction	155
2	Economic Theory for the Internet Being with Selfish Agents	157
3	A Microscopic Model: Negotiation Between Agents	159
	3.1 The Prisoner's Dilemma	159
	3.2 Repairing from Outside the System: A Conventional Model [12]	160
	3.3 Mutual Repair within Systems	160
	3.4 Mutual Repair with Selfish Agents	161
4	A Macroscopic Model: Boundary Formation among Agents	163
	4.1 A Model with Uniform Control	163
	4.2 The Spatial Prisoner's Dilemma	167
	4.3 A Model with Selfish Agents	168
	4.4 Strategic Repair with Systemic Payoff	169
	4.5 Comparison Between Uniform Repair and Strategic Repair	170
5	Selfishware and Internet Being	173
6	Conclusion	175
	References	175

	Resources	179
1	Key Books	179
2	Organisations, Societies, Special Interest Groups	179
3	Research Groups	180
4	Discussion Groups, Forums	180
5	Key International Conferences/Workshops	180
6	(Open Source) Software	181
7	Data Bases	181

Part III Artificial Intelligence

Emotional Intelligence: Giving Computers Effective Emotional Skills to Aid Interaction

	<i>Chris Creed and Russell Beale</i>	185
1	Introduction	185
2	Overview of Affective Computing	187
	2.1 What Are Emotions?	187
	2.2 Emotions and Moods	190
	2.3 Expression of Emotion	191

2.4	Influence of Emotion on Human behavior	193
2.5	Emotional Intelligence	196
2.6	Approaches Used in Developing Emotionally Intelligent Computers	197
2.7	Ethics	203
3	Evaluating Affective Embodied Agents	206
3.1	What are Affective Embodied Agents?	207
3.2	Psychological Responses to Simulated Emotion	207
3.3	Evaluating Agents over Extended Interactions	209
3.4	Our Affective Embodied Agent	210
4	Application of Affective Embodied Agents	211
4.1	Affective Embodied Agents for behavior Change	212
4.2	Behavior Change Models	213
5	Summary	216
	References	217
	Resources	225
1	Key Books	225
2	Key Survey/Review Articles	226
3	Organisations, Societies, Special Interest Groups	226
4	Research Groups	227
5	Discussion Groups, Forums	228
6	Key International Conferences/Workshops	228
7	(Open Source) Software	229
8	Data Bases	229
8.1	Multimodal Databases	229
8.2	Face Databases	230

Part IV Logic and Reasoning

The Paraconsistent Annotated Logic Program EVALPSN and its Application

	<i>Kazumi Nakamatsu</i>	233
1	Introduction	233
1.1	Background	233
1.2	Overview	234
2	Preliminary	235
2.1	Paraconsistent Annotated Logics PT	235
2.2	Generally Horn Program(GHP)	237
2.3	ALPSN (Annotated Logic Program with Strong Negation) and Stable Model Semantics	241
2.4	VALPSN (Vector Annotated Logic Program with Strong Negation)	243

2.5	EVALPSN (Extended Vector Annotated Logic Program with Strong Negation) and Defeasible Deontic Reasoning	244
2.6	Defeasible Reasoning and VALPSN	247
2.7	Defeasible Deontic Reasoning and EVALPSN	256
3	EVALPSN Safety Verification for Control	265
3.1	Outline of EVALPSN Safety Verification	265
3.2	EVALPSN Safety Verification for Pipeline Control	266
4	Before-after EVALPSN	284
4.1	Before-after Relation in EVALPSN	285
4.2	Implementation of bf-EVALPSN	291
4.3	Safety Verification in bf-EVALPSN	295
5	Conclusion and Future Work	299
	References	300
	Resources	305
1	Logic Programming	305
2	Paraconsistent Annotated Logic	305
3	Defeasible Logic	305
4	Defeasible Deontic Logic	306
5	ALPSN, VALPSN, EVALPSN	306
6	EVALPSN Safety Verification	306
	The Data-Oriented Parsing Approach: Theory and Application	
	<i>Reus Bod</i>	307
1	Introduction	307
2	A DOP Model for Language: Combining Likelihood and Simplicity	308
3	A DOP Model for Music	315
4	A DOP Model for Problem Solving in Physics	318
5	Towards a Unifying Approach	323
6	Test Corpora for DOP+	325
7	Computing T_{best}	327
8	Experiments with DOP+	330
9	Current Developments: Unsupervised DOP	333
10	Conclusion	336
	References	336
	Resources	343
1	Key Books	343
2	Key Survey/Review Articles	343
3	Organisations, Societies, Special Interest Groups	344
4	Research Groups	345
5	Discussion Groups, Forums	345

6	Key International Conferences/Workshops	346
7	(Open Source) Software.....	347
8	Data Bases	347
8.1	Multimodal Databases	347
8.2	Face Databases	348

Part V Ontology

Graph-Based Representation and Reasoning for Ontologies

<i>Dan R. Corbett</i>	351
1 Introduction.....	351
2 Overview of Conceptual Graphs	352
2.1 The Basics	352
2.2 Fundamental Concepts.....	354
2.3 Canonical Formation Rules	355
2.4 Types and Inheritance	355
2.5 Specialization, Projection and Subsumption.....	357
3 Projection as an Ontology Operator	358
4 Projection of Ontology Types.....	360
5 Knowledge Conjunction.....	362
5.1 Ontology Comparison and Conjunction.....	362
5.2 Unification, Constraints and Conceptual Graphs.....	364
5.3 Knowledge Structures, Partialness and Unification	365
6 An Architectural Design Tool.....	367
7 An Architectural Design Tool: Results and Discussion	368
8 The Air Operations Officer	370
9 The Air Operations Officer: Results and Discussion.....	372
10 Conclusions: Semantics for a Knowledge Web	372
References	374

Resources

1 Key Books	377
2 Key Survey/Review Articles.....	377
3 Research Groups	378
4 Discussion Groups, Forums	378
5 Key International Conferences/Workshops	378
6 (Open Source) Software.....	379

An Ontology-Based Intelligent Mobile System for Tourist Guidance

<i>Toby H.W. Lam, Raymond S.T. Lee, and James N.K. Liu</i>	381
1 Introduction.....	381
2 Background	383
2.1 The Semantic Web	383
2.2 Agent	386

3	Related Work	388
4	Ontology-Based Tourist Guide	389
4.1	iJADE Framework	389
4.2	Construction of the Travel Ontology	390
4.3	iJADE FreeWalker	395
4.4	iJADE System Architecture	397
5	Performance Evaluation	400
5.1	Precision Test	401
5.2	Usability Test	401
6	Conclusion and Further Work	402
	References	403
	Resources	405
1	Key Books	405
2	Key Survey/Review Articles	405
3	Websites	405
4	Key International Conferences/Workshops	406
5	(Open Source) Software	406
6	Data Bases	406

Part VI Intelligent Agents

Open Source Agent-Based Modeling Frameworks

	<i>Russell K. Standish</i>	409
1	Introduction	409
1.1	Artificial Life (Alife)	409
2	Applications	411
2.1	<i>Sugarscape</i>	412
2.2	The <i>Santa Fe</i> Artificial Stock Market	413
2.3	Heatbugs	416
2.4	Mousetrap	417
3	Software Modeling Tools	417
3.1	Open Source <i>versus</i> Freeware	417
3.2	Programming Languages	418
3.3	Reflection	421
3.4	User Interface and Scripting	421
3.5	Discrete Event Scheduling	422
3.6	Random Number Library	422
3.7	Swarm	423
3.8	Repast	424
3.9	Mason	425
3.10	<i>EcLab</i>	426
3.11	The Logos, StarLogo and NetLogo	427
3.12	Cormas	428

4 Performance Comparisons 428
 5 Conclusion 431
 References 431

Resources 435
 1 ABM Platforms 435
 2 Discussion Fora 436

Agent-Based Grid Computing

Minjie Zhang, Jia Tang, and John Fulcher 439
 1 Introduction 439
 2 Computing Grids 440
 2.1 Development of Computing Grids 441
 2.2 Application-Oriented Metacomputing 442
 2.3 Service-Oriented Grid Computing 443
 2.4 Convergence of Grids and Peer-to-Peer Computing 444
 2.5 Research Questions of Grid Computing 445
 3 Grid Computing in Open Environments 446
 4 SmartGrid – A Hybrid Solution to Grid Computing
 in Open Environments 446
 4.1 Overall Architecture and Core Components 447
 4.2 The Task/Service Model 451
 4.3 The smartGRID Scheduling Process 453
 5 A Peer-to-Peer Solution to Grid Computing
 in Open Environments 462
 5.1 Overall Architecture and Core Components
 of smartGRID2 462
 5.2 Module – An Improved Task Model 465
 5.3 Peer-to-Peer Computing Architecture 467
 5.4 Resource Management and Scheduling Mechanisms 471
 5.5 Compatibility and Inter-Operability 475
 6 Conclusion and Further Work 475
 References 477

Resources 481
 1 Key Books 481
 2 Key Survey/Review Articles 481
 3 Journal 482
 4 Key International Conferences/Workshops 482
 5 Web Resources 482

**Decentralized Multi-Agent Clustering in Scale-free
 Sensor Networks**

*Mahendra Piraveenan, Mikhail Prokopenko, Peter Wang,
 and Astrid Zeman* 485
 1 Introduction 485

1.1	Multi-Agent Systems and Self-organization	485
1.2	Multi-Agent Networks	487
1.3	Adaptive Topologies and Dynamic Hierarchies	488
2	Dynamic Cluster Formation Algorithm	490
3	Regularity of Multi-Agent Communication-Volume	494
4	Experimental Results	496
5	An Application Scenario – Distributed Energy Management and Control	501
6	Conclusions	502
	References	503
	Decentralised Clustering Algorithm	507
	Predictor K_2	511
	Resources	513
1	Key Books	513
2	Key Survey/Review Article	513
3	Organisations, Societies, Special Interest Groups	513
4	Research Groups	514
5	Discussion Group, Forum	514
6	Key International Conferences/Workshops	514
	Computational Intelligence in Agent-Based Computational Economics	
	<i>Shu-Heng Chen</i>	517
1	Introduction	517
1.1	What is <i>Agent-Based Computational Economics</i> (ACE)? ...	517
1.2	Algorithmic Foundations of ACE	518
2	Artificial Neural Networks	519
2.1	Multilayer Perceptron Neural Networks	520
2.2	Radial Basis Network	522
2.3	Recurrent Neural Networks	522
2.4	Auto-Associative Neural Networks	524
2.5	Support Vector Machines	528
2.6	Self-Organizing Maps and k-means	529
2.7	K Nearest Neighbors	532
2.8	Instance-Based Learning	533
3	Evolutionary Computation	536
3.1	Evolutionary Strategies	538
3.2	Evolutionary Programming	540
3.3	Genetic Programming and Genetic Algorithms	540
4	Agent-Based Economic Simulations with CI	549
4.1	The Cobweb Model	549
4.2	Overlapping Generations Models	552

4.3	Foreign Exchange Rate Fluctuations	558
4.4	Artificial Stock Markets	562
4.5	Market/Policy Design	568
5	Pushing the Research Frontier with CI	570
5.1	Developments in Agent Engineering	570
5.2	Distinguishing Features	572
5.3	Future Directions	576
6	Concluding Remarks	579
	References	580
Resources		591
1	Key Books	591
2	Key Survey/Review Articles	592
3	Journals	592
4	Key International Conferences/Workshops	592
4.1	Economics	592
4.2	Agents	593
5	(Open Source) Software	593
6	Data Bases	594

Part VII Fuzzy Systems

**Semantics and Perception of Fuzzy Sets
and Fuzzy Mappings**

	<i>Witold Pedrycz</i>	597
1	Semantics of Fuzzy Sets: Some General Observations	597
2	Domain Knowledge and Problem-Oriented Formation of Fuzzy Sets	599
2.1	Fuzzy Set as a Descriptor of Feasible Solutions	599
2.2	Fuzzy set as a Descriptor of the Notion of Typicality	601
2.3	Membership Functions in the Visualization of Preferences of Solutions	603
3	User-Centric Estimation of Membership Functions	605
3.1	Horizontal Membership Function Estimation Scheme	605
3.2	Vertical Membership Function Estimation Scheme	606
3.3	Pairwise Membership Function Estimation Scheme	607
4	Fuzzy Sets as Granular Representatives of Numeric Data	609
5	From Multidimensional Numeric Data to Fuzzy Sets: Membership Estimation via Fuzzy Clustering	614
6	Main Design Guidelines	620
7	Nonlinear Transformation of Fuzzy Sets	621
8	Reconciliation of Information Granule Perception	625
9	The Optimization Process	626

10	An Application of the Perception Mechanism to Rule-Based Systems	627
11	Reconciliation of Granular Mappings	629
12	Conclusions	634
	References	634
	Resources	637
1	Key Books	637
2	Key Survey/Review Articles	637
3	Organisations, Societies, Special Interest Groups	638
4	Research Groups	638
5	Key International Conferences/Workshops	639
	Evolutionary Multiobjective Design of Fuzzy Rule-Based Classifiers	
	<i>Hisao Ishibuchi, Yusuke Nojima, and Isao Kuwajima</i>	641
1	Introduction	641
2	Fuzzy Rule-Based Classifiers	644
2.1	Pattern Classification Problems	644
2.2	Fuzzy Rules	644
2.3	Fuzzy Reasoning	648
2.4	Fuzzy Rule Extraction	650
2.5	Comparison Between Fuzzy and Interval Rules	653
3	Evolutionary Multiobjective Optimization (EMO)	655
3.1	Genetic Algorithms (GAs)	655
3.2	Multiobjective Optimization (MO)	657
3.3	Evolutionary Multiobjective Optimization (EMO)	658
4	Two Approaches to Evolutionary Multiobjective Design of Fuzzy Rule-Based Classifiers	661
4.1	Problem Formulation	662
4.2	Multiobjective Fuzzy Rule Selection	663
4.3	Multiobjective Fuzzy Genetics-Based Machine Learning	667
4.4	Computational Experiments on Test Problems	669
5	Future Research Directions	673
6	Concluding Remarks	674
	References	675
	Resources	681
1	Key Books	681
1.1	Fuzzy Rule-Based Classification Systems	681
1.2	Genetic Algorithms	681
1.3	Genetic Fuzzy Systems	682
1.4	Evolutionary Multiobjective Optimization	682
1.5	Evolutionary Multiobjective Machine Learning and Knowledge Extraction	682

2	Conferences	682
2.1	Fuzzy Systems	682
2.2	Genetic Algorithms	683
2.3	Genetic Fuzzy Systems	683
2.4	Evolutionary Multiobjective Optimization	683
2.5	Hybrid Systems	683
2.6	Broader Areas, Including Fuzzy Systems and Genetic Algorithms	683
3	Journals	683
3.1	Fuzzy Systems	683
3.2	Genetic Algorithms	684
3.3	Broader Areas, Including Fuzzy Systems and Genetic Algorithms	684
4	Websites	684
5	(Open Source) Software	685
6	Data Bases	685

Part VIII Artificial Neural Networks

Data Mining in QoS-Aware Media Grids

Xiuju Fu, Xiaorong Li, Lipo Wang, David Ong,

and Stephen John Turner

1	Introduction	689
2	Related Work	691
2.1	Network Bandwidth Prediction	691
2.2	Brief Overviews on Neural Networks	692
3	System Model of Data Analysis over Media Grid	694
3.1	Architecture	694
3.2	System Components	696
4	Data Mining Strategy for Bandwidth Prediction	697
4.1	Multi-Layer Perceptron Neural Network	697
4.2	Data Mining Strategy	699
4.3	Performance Metrics	701
5	Experimental System and Performance Evaluation	702
5.1	System Hardware and Software	702
5.2	Request Arrival Pattern	702
5.3	Results and Analysis	703
6	Conclusions	704
	References	709

Resources

1	Key Books	713
2	Key Survey/Review Articles	713
3	Organisations, Societies, Special Interest Groups	714

4	Key International Conferences/Workshops	714
5	(Open Source) Software.....	714
The Self-Organizing Maps: Background, Theories, Extensions and Applications		
	<i>Hujun Yin</i>	715
1	Introduction.....	715
2	Background	716
2.1	Biological Background: Lateral Inhibition and Hebbian Learning	716
2.2	From Von Marsburg and Willshaw's Self-Organization Model to Kohonen's SOM	721
2.3	The SOM Algorithm	725
3	Theories	726
3.1	Convergence and Cost Functions	726
3.2	Topological Ordering	729
4	Extensions and Links with Other Learning Paradigms	731
4.1	SOM, Multidimensional Scaling and Principal Manifolds ...	732
4.2	SOM and Mixture Models	738
4.3	SOM and Kernel Method	740
5	Applications and Case Studies	743
5.1	Vector Quantization and Image Compression.....	743
5.2	Image Segmentation	744
5.3	Density Modeling.....	745
5.4	Gene Expression Analysis	747
5.5	Data Visualization.....	749
5.6	Text Mining and Information Management	750
6	Summary and Future Directions	753
	References	754
	Resources	761
1	Key Books	761
2	Key Survey/Review Articles.....	761
3	Key International Conferences/Workshops	762
4	(Open Source) Software.....	762
Neural Systems Engineering		
	<i>Steve Furber and Steve Temple</i>	763
1	Introduction.....	763
1.1	The Neuron	764
1.2	Neural Microarchitecture	766
1.3	Engineering with Neurons	766
1.4	Scoping the Problem	767
1.5	The Research Agenda.....	768
1.6	Chapter Structure	769

- 2 Neural Computation 769
 - 2.1 Processing 770
 - 2.2 Communication 771
 - 2.3 Storage 771
- 3 The Neuron as a Component 772
 - 3.1 Communicating with Spikes 772
 - 3.2 Point-Neuron Models 773
 - 3.3 The Spike Response Model 774
 - 3.4 The Izhikevich Model 774
 - 3.5 Axons: The Hodgkin-Huxley Model 776
 - 3.6 Dendritic Trees and Compartmental Models 776
 - 3.7 The Synapse 777
- 4 Engineering Neural Systems 777
 - 4.1 Neural Models 778
 - 4.2 Population Encoding 778
 - 4.3 Spatio-Temporal Spike Neurons 780
 - 4.4 Defining ‘Connectivity’ 780
 - 4.5 Implementing Connectivity 781
 - 4.6 Learning, Adapting, and Tuning 781
 - 4.7 Example Neural Systems 782
 - 4.8 Neuromorphic Systems 782
- 5 Large-Scale Projects 783
 - 5.1 Blue Brain 783
 - 5.2 SPINN 784
 - 5.3 SpiNNaker 785
 - 5.4 Virtual Communication 786
 - 5.5 Diverse Approaches 789
- 6 Future Prospects 789
- References 790

- Resources** 795
 - 1 Key Books 795
 - 2 Key Reference Source 795
 - 3 Research Groups 796
 - 4 Key International Workshop 796
 - 5 (Open Source) Software 796

Artificial Brains: An Evolved Neural Net Module

- Approach**
- Hugo de Garis* 797
 - 1 Introduction 797
 - 2 Related Work 799
 - 2.1 Some Recent Artificial Brain Projects 800
 - 2.2 Some Other Recent Artificial Brain Projects 803
 - 3 The Evolution of Neural Network Modules 804

3.1	The Evolutionary Tasks	805
3.2	Our Evolutionary Approach	805
3.3	The Standard Genetic Algorithm	806
4	The Celoxica Board	806
5	Experimental Results	808
5.1	IMSI (Inter Module Signaling Interface)	809
5.2	How Many Modules?	811
6	The Robot and Brain-Robot Interface	812
7	Artificial Brain Architectures	814
7.1	A Simple Artificial Brain Architecture	815
7.2	Incrementing the Design	823
7.3	Why Not Just Program Everything?	826
7.4	Evolving Individual Modules	827
8	The Need for Generic Evolution	830
8.1	Limitations of Our Approach	832
8.2	Evolvability – A Key Issue	832
8.3	Book-Keeping of Modules and Circuits	833
9	Future Work	834
9.1	The ‘China Brain’ Project	836
10	Conclusion	837
10.1	Final Word	839
	References	839
	Resources	841
1	Key Books	841
1.1	Artificial Brain Architectures	841
1.2	Brain Theory	842
1.3	Cognitive Modeling	842
1.4	Evolvable Hardware (EHW)	843
1.5	Gerald Edelman	843
1.6	Ethology	844
1.7	Genetic Algorithms (GA)	844
2	Key Journals	845
3	Artificial Brain Research Groups	845
3.1	Markram’s ‘Blue Brain’ Project	845
3.2	Adaptive Development’s ‘CCortex’	845
3.3	Edelman’s ‘Darwin IV’ Robot Brain	845
4	Key International Conferences/Workshops	846
4.1	Congress on Evolutionary Computation – CEC (IEEE)	846
4.2	GECCO – Genetic and Evolutionary Computation Conference	846
4.3	ICES – International Conference on Evolvable Systems	847
4.4	NASA/DoD Conferences on Evolvable Hardware	847

Part IX Evolutionary Approaches

Evolving Artificial Neural Network Ensembles

Md. Monirul Islam and Xin Yao 851

1 Introduction 851

2 Evolutionary Ensembles 852

 2.1 An Evolutionary Design System for ANNs – EPNNet 853

 2.2 Combination Methods 855

 2.3 Experimental Studies 856

3 Automatic Modularization 859

4 Negative Correlation Learning 860

 4.1 Evolutionary Ensembles with Negative
 Correlation Learning 862

 4.2 Experimental Studies 864

5 Constructive Approaches to Ensemble Learning 865

 5.1 Experimental Studies 868

6 Multi-Objective Approaches to Ensemble Learning 870

 6.1 Experimental Studies 871

7 Conclusions 872

References 872

Resources 877

1 Key Books 877

2 Key Survey/Review Articles 877

3 Organizations, Societies, Special Interest Groups 878

4 Research Groups 878

5 Discussion Groups, Forums 878

6 Key International Conferences and Workshops 878

7 (Open Source) Software 879

8 Data Bases 879

**An Order Based Memetic Evolutionary Algorithm
for Set Partitioning Problems**

Christine L. Mumford 881

1 Introduction 881

2 A Brief History of Genetic Algorithms 883

3 A Generic Genetic Algorithm 883

4 Order Based GAs 886

5 A Simple Steady-State GA 891

6 Set Partitioning Problems 892

 6.1 The Graph Coloring Problem 893

 6.2 The Bin Packing Problem 894

 6.3 The Examination Timetabling Problem 894

 6.4 Other Set Partitioning Problems 895

7	Motivation for the Present Study	896
8	Culberson and Luo's Grouping and Reordering Heuristics	898
9	Modifications to a Standard Order Based GA for Set Partitioning	902
9.1	Performance Measures/Fitness Values	904
9.2	Comparing Order Based Crossovers	906
9.3	The Genetic Simulated Annealing (GSA) Algorithm	906
10	Results on Literature Benchmarks	911
10.1	Graph Coloring	912
10.2	Bin Packing	914
10.3	Timetabling	917
11	Summary	919
	References	920
	Resources	923
1	Key Books	923
2	Key International Conferences	923
3	Interest Groups/Web sites	924
4	(Open Source) Software	924
5	Data Sets used in the Chapter	925
	Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications <i>William B. Langdon, Riccardo Poli, Nicholas F. McPhee, and John R. Koza</i>	927
1	Introduction	927
1.1	GP in a Nutshell	928
1.2	Overview of the Chapter	929
2	Representation, Initialization and Operators in Tree-Based GP ...	929
2.1	Representation	929
2.2	Initializing the Population	931
2.3	Selection	934
2.4	Recombination and Mutation	934
3	Getting Ready to Run Genetic Programming	936
3.1	Step 1: Terminal Set	937
3.2	Step 2: Function Set	937
3.3	Step 3: Fitness Function	940
3.4	Steps 4 and 5: Parameters and Termination	942
4	Example Genetic Programming Run	943
4.1	Preparatory Steps	943
4.2	Step-by-Step Sample Run	944
5	Advanced Tree-Based GP Techniques	948
5.1	Automatically Defined Functions	948
5.2	Program Architecture and Architecture-Altering Operations	949

5.3	Genetic Programming Problem Solver	949
5.4	Constraining Syntactic Structures	950
5.5	Developmental Genetic Programming	954
5.6	Strongly Typed Autoconstructive GP – PushGP	954
6	Linear and Graph-Based GP	955
6.1	Linear Genetic Programming	955
6.2	Graph-Based Genetic Programming	957
7	Applications	958
7.1	Curve Fitting, Data Modeling, and Symbolic Regression	959
7.2	Human Competitive Results – <i>The Humies</i>	962
7.3	Image and Signal Processing	965
7.4	Financial Trading, Time Series Prediction and Economic Modeling	966
7.5	Industrial Process Control	967
7.6	Medicine, Biology and Bioinformatics	968
7.7	Mixing GP with Other Techniques	969
7.8	GP to Create Searchers and Solvers – Hyper-Heuristics	969
7.9	Artistic	969
7.10	Entertainment and Computer Games	970
7.11	Where can we Expect GP to Do Well?	970
8	Tricks of the Trade	971
8.1	Getting Started	971
8.2	Presenting Results	972
8.3	Reducing Fitness Evaluations/Increasing their Effectiveness	973
8.4	Co-Evolution	975
8.5	Reducing Cost of Fitness with Caches	976
8.6	GP Running in Parallel	977
8.7	GP Trouble-Shooting	981
9	Genetic Programming Theory	982
9.1	Mathematical Models	983
9.2	Search Spaces	984
9.3	Bloat	986
10	Conclusions	987
	References	989
	Resources	1025
1	Key Books	1025
2	Videos	1026
3	Key Journals	1026
4	Key International Conferences/Workshops	1026
5	Online Resources	1027

The Particle Swarm Algorithm

<i>Tim Hendtlass</i>	1029
1 Introduction	1029
2 The Basic Particle Swarm Optimization Algorithm	1030
2.1 Pseudo Code Algorithm for the Basic PSO	1034
3 Enhancements to the Basic Particle Swarm Algorithm	1034
3.1 Constriction Factors	1034
3.2 Adding Controlled Diversification	1035
3.3 Handling Problem Constraints	1035
4 Particle Swarm Optimization of Multiple Optima	1036
4.1 Exploring Multiple Optima	1037
4.2 Achieving Parallel Exploration of Several Positions of Interest (niching)	1037
4.3 Achieving Serial Exploration of Many Positions of Interest (WoSP)	1038
5 Controlling the Computational Expense	1041
5.1 Using a Dynamic Swarm Size	1041
5.2 Fitness Estimation	1041
6 Dynamic Optimization Problems	1043
6.1 Ways to Achieve these Adaptations	1044
6.2 Preventing Total Convergence	1045
6.3 Refreshing the Best Positions	1045
6.4 Forcing Explorer Particles	1046
6.5 Adapting WoSP to Dynamic Problems	1046
7 Particle Swarm and Quantized Problem Spaces	1046
8 Some Sample Results	1048
8.1 Problems used as Examples in this Chapter	1048
8.2 Experimental Details	1050
9 Sample Results	1050
9.1 Minimizing the Distance to the Origin in 100 Dimensions	1051
9.2 Rastrigin's Function in 100 Dimensions	1052
9.3 Schwefel's Function in 30 Dimensions	1054
10 Concluding Remarks	1059
References	1059

Resources 1061

1 Key Books	1061
2 Organisations, Societies, Special Interest Groups, Journals	1061
3 Key International Conferences/Workshops	1061
4 (Open Source) Software	1062

Part X DNA and Immunity-Based Computing

DNA Computing and its Application

<i>Junzo Watada</i>	1065
1 Introduction	1065
2 DNA Computing	1065
2.1 Encoding Scheme	1067
3 Comparison with Conventional Computing	1069
4 Applications of DNA Computing	1071
5 Approaches to Optimization and Scheduling	1071
6 Elevator Management System	1072
6.1 Restrictions on Elevator Movements	1074
6.2 Elevator Scheduling	1076
7 Bio-Soft Computing Based on DNA Length	1077
8 Bio-Soft Computing with Fixed-Length DNA	1079
8.1 Empirical Study	1081
9 Conclusion	1084
References	1085

Resources
 1087 |

1 Key Books	1087
1.1 DNA Computing	1087
1.2 Elevator Management	1087
2 Key Survey/Review Articles	1088
3 International Organization	1088
4 Discussion Groups, Forums	1088
5 Research Groups	1088
6 Key International Conferences and Workshops	1089
7 Web Resource	1089

**The Next Generation of Immunity-Based Systems:
From Specific Recognition to Computational Intelligence**

<i>Yoshiteru Ishida</i>	1091
1 Introduction	1091
2 Impact of Recognition	1093
2.1 An Impact of Recognition is a Double-Edged Sword	1094
3 Immunity-Based Systems: Evolved Recognitions	1095
3.1 Definition of Immunity-Based Systems	1095
3.2 Networked Recognition	1096
3.3 Adaptive Recognition	1101
4 Antibody-Based Computing: Arrayed Recognition	1105
4.1 Definition of Antibody-Based Computing	1106
4.2 Solving a Combinatorial Problem: The Stable Marriage Problem	1106

4.3	Mapping the Stable Marriage Problem to Antibody-Based Computing	1107
5	Toward a General Problem Solver: Immunity-Based Problem Solver	1110
6	Conclusion	1113
	References	1114
	Resources	1117
1	Key Books	1117
2	Key Survey/Review Articles	1118
3	Organisations, Societies, Special Interest Groups	1119
4	Research Groups	1119
5	Key International Conferences/Workshops	1119
	Index	1121

Overview, Background

Computational Intelligence: An Introduction

John Fulcher

Intelligent Systems Research Centre, University of Wollongong NSW 2522,
Australia, john@uow.edu.au

1 Introduction, Overview, Definitions

The Artificial Intelligence field continues to be plagued by what can only be described as ‘bold promises for the future syndrome’, often perpetrated by researchers who should know better.¹ While impartial assessment can point to concrete contributions over the past 50 years (such as automated theorem proving, games strategies, the LISP and Prolog high-level computer languages, Automatic Speech Recognition, Natural Language Processing, mobile robot path planning, unmanned vehicles, humanoid robots, data mining, and more), the more cynical argue that AI has witnessed more than its fair share of ‘unmitigated disasters’ during this time – see, for example [3, 58, 107, 125, 186]. The general public becomes rapidly jaded with such ‘bold predictions’ that fail to live up to their original hype, and which ultimately render the zealots’ promises as counter-productive.

To lay claim to having developed an ‘intelligent’ system is to open a hornet’s nest of debate – a debate which has raged since the early days of AI. This is not surprising, since one can very quickly stray into the realms of philosophy, metaphysics and/or religion (and as the adage goes, never discuss politics or religion with your fellows!). We do not propose to add to the debate here, but to simply make our contribution by way of *practical* methods which have proven effective over time in dealing with real-world applications. It is a moot point then as to whether the ‘intelligence’ in these systems can be attributed to some ‘Ghost in the Machine’, as it were, or alternately to their creator(s) – in other words, to the ‘cleverness’ of the system developer(s). For readers interested in such philosophical issues, they are referred to the extensive literature in this area (for starters, see [33, 44, 45, 67, 112, 245]).

So what then is the focus of the present Handbook, and in particular the Chapters which follow? In short, the techniques herein described originated

¹ Fulcher J, Jain LC (2004) *Applied Intelligent Systems*, Springer: Preface (VII–X).

in several different fields, including ‘Cybernetics’, ‘Machine Learning’ (ML), ‘Artificial Intelligence’ (AI) (together with its later offshoot ‘Connectionism’), ‘Data Mining’ (DM), ‘Knowledge Engineering’ (KE), ‘Intelligent Systems’, ‘Soft Computing’, and in more recent times, ‘Computational Intelligence’ (CI). Actually these approaches share more in common than one might at first suspect, going on their names alone.

Historically, AI has progressed over time by way of ‘quantum leaps’, beginning with traditional (philosophy/psychology-based) AI, through Artificial Neural Networks (inspired by neurobiology), Evolutionary Computation (inspired by genetics and biology), complex systems (inspired by economics and biology), to present-day CI. What the next leap (‘wave’) will be is open to speculation, but in this author’s view it stands a very good chance of being ‘Nature-inspired’ (see Sect. 3). The common characteristic of *each* such ‘wave’ is the aspect of *computation*. Accordingly, any attempt to define ‘intelligence’ must necessarily bear this in mind. However Conrad makes a valid point in this regard, that “no system can be at once highly structurally programmable, evolutionary efficient, and computationally efficient.” [55]

One definition of CI emphasizes (a) the ability to learn, (b) to deal with new situations, and (c) to reason [80]. Some early definitions of CI restricted themselves to ‘intelligent agents’ [16, 47, 247]. Another early definition is more typical of current-day attitudes: “Artificial Neural Networks (ANNs), Evolutionary Computation (EC), and Fuzzy Systems” (although this same author has since widened their definition to “the study of adaptive mechanisms to enable or facilitate intelligent behaviors in complex and changing environments” [82]). Pedrycz emphasizes the synergy between granular computing (in particular fuzzy sets), ANNs, and evolutionary optimization [240]. Further, he maintains the *order* in which the techniques are applied is important – more specifically, using a top-down approach, we commence with granular computing (say fuzzy sets), then refine the system using neural networks on numeric data.

According to Fogel,

“These technologies of neural, fuzzy and evolutionary systems were brought together under the rubric of CI, a relatively new term offered to generally describe methods of computation that can be used to adapt solutions to new problems and do not rely on explicit human knowledge.” [94]

Likewise Karplus states:

“CI substitutes intensive computation for insight into how the system works. Neural Networks, Fuzzy Systems and Evolutionary Computation were all shunned by classical system and control theorists. CI umbrellas and unifies these and other revolutionary methods.” [151]

Bezdek is more specific, characterizing CI thus:

“A system is computationally intelligent when it: deals with only numerical (low-level) data, has pattern recognition components, does not use knowledge in the AI sense; and additionally when it (begins to) exhibit (1) computational adaptivity; (2) computational fault tolerance; (3) speed approaching human-like turnaround, and (4) error rates that approximate human performance.” [16, 17]

According to the *IEEE Computational Intelligence Society*,² “CI is a field that greatly evolved in the last quarter century; from initial steps in the direction of understanding the mechanisms of human reasoning towards the study of all aspects of natural intelligence and behavior. The ultimate goal of researchers in this field was mimicking Nature with artificial technologies to replicate the basic mechanisms of Nature in engineering systems for the benefit of humanity. . . CI technologies are living approaches to tackle real-world problems. . . created as answers to the needs of applications.”

Duch characterizes CI as “that branch of Computer Science studying “problems for which there are no effective computational algorithms”. He further suggests that AI should be regarded as a sub-set of CI, which will no doubt upset traditionalists! More specifically, “AI is that part of CI that focuses on problems that require higher cognition and are at present easier to solve using symbolic knowledge representation.” [78]

Nowadays, most authors would agree on a core definition of fuzzy, neural and evolutionary (data-driven) methodologies, but some extend this to cover granular computing [190, 240, 326, 330], probabilistic reasoning, Bayesian (belief) networks [147, 161, 216], fuzzy Petri nets, constrained reasoning, case-based reasoning [231, 304], Support Vector Machines [1, 270, 297], rough sets [140, 189, 237], learning/adaptive classifiers, fractals [85, 200], wavelets [198, 242], and/or chaos theory [228, 282], not to mention the intelligent agents [229] alluded to earlier. [296] make some further observations as to how the CI field has *evolved* during the previous decade, with the emergence of differential evolution, Particle Swarm Optimization (Sect. 8.4), multi-objective evolutionary optimization (Sect. 8) – such as NSGA-II (fast elitist Non-dominated Sorting Genetic Algorithm) – and Support Vector Machines (Sect. 7.3).

Henceforth in this Handbook, we use the term CI in its most generic sense, and take it to mean the use of Artificial Neural Network, evolutionary and/or Fuzzy techniques, and more especially *hybrids* or synergistic combinations/ensembles of these complementary approaches (as well as occasionally incorporating rule-based and/or statistical ones). Moreover, the resulting technique(s) will usually be iterative in nature, with successive solutions delivering

² CIS President’s Forum, *IEEE World Congress CI*, 19 July 2006, Vancouver, Canada.

improved performance/accuracy, to a user-specified degree. As such, CI is potentially capable of solving problems which remain intractable to solution by any *individual* technique – truly, the whole is greater than the sum of its parts.

Application of CI methods will typically result in so-called ‘black box’ solutions to problems of interest, which while they may be effective, are not always welcomed by users – simply because it is difficult to provide justification/rationalization/explanation of any decisions thus made (in contrast say to Decision Trees (DTs), from which it is a relatively straightforward matter to extract explanatory rules). A strong selling point with CI systems is nevertheless their superior ability to model complex real-world systems which have proved intractable using classical (conventional mathematical/logic) methods – hence their popularity with researchers and practitioners alike.

As a first approximation, we can discriminate between CI methods which have an algorithmic or logical rule basis (model-driven) and those best described as being inherently *non*-algorithmic. With the latter, there is the added implication of a data-driven, iterative process, as well as some form of inspiration from biology, or more generally, Nature, although Teuscher cautions against biological inspiration for its own sake – rather he advocates inspiration in the broadest sense, including unconventional and novel paradigms, that need not be biological at all. These issues are elaborated upon in Sect. 3.

We could explore this a little further in the formulation of an alternative definition of CI. A classic text from the era of procedural programming (and the Pascal HLL) was Wirth’s *Algorithms + Data Structures = Programs*.³ Two decades later, Michalewicz attempted to similarly define ‘evolutionary programming’ thus⁴:

“Genetic Algorithms + Data Structures = Evolution Programs.”

A decade further on, we could define CI as follows:

Nature-inspired method(s) + real-world (training) data = Computational Intelligence.

Let’s examine each of these three components in our defining ‘equation’. Firstly, ‘Nature-inspired method’ does not necessarily imply an algorithmic basis; often heuristics suffice. On the other hand, the backpropagation (BP)

³ Wirth N (1976) *Algorithms + Data Structures = Programs*. Prentice Hall, Englewood Cliffs, NJ.

⁴ Michalewicz Z (1996) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

algorithm and Genetic *algorithms*, as their names suggest, have an algorithmic foundation. The various sources of inspiration from Nature are further discussed in Sect. 3.1. Secondly, data *structure* is not nearly as important as a sufficient amount of training (validation, and testing) data, notwithstanding bit-string chromosome representation in EAs, and set membership in Fuzzy Logic. Thirdly, since we of necessity use real-world data, then CI *ipso facto* facilitates real-world problem solving (see the earlier *IEEE CI Society* definition of Computational Intelligence). We shall explore this idea further in Sect. 3.1.

2 Historical Background

Researchers with an historical bent often regard particular scientific gatherings as constituting the origin of various disciplines: in the case of ‘Software Engineering’ this took the form of NATO-sponsored workshops during the late 1960s; with ‘Artificial Intelligence’ – a term coined by John McCarthy, then of MIT, to mean the science/engineering of constructing intelligent machines – this can be traced backed to a Workshop at Dartmouth College in the US in 1956.⁵

2.1 Artificial Intelligence (AI)

Much debate has taken place over the ensuing decades attempting to define ‘Artificial Intelligence’. Before we consider ‘AI’ specifically, we need to come to some understanding of the more general concept of ‘intelligence’. The following are typical dictionary definitions:

- ‘understanding’
- ‘the collection of information’
- ‘capacity for understanding and other forms of adaptive behavior’
- ‘aptitude for grasping facts, truths or meaning’
- ‘the ability to plan, reason, solve problems, think abstractly, comprehend ideas and language, and learn’

In 1996, the *IEEE Neural Networks Council* defined AI as:

“the study of how to make computers do things at which, at the moment, people are better.”

Others go further, citing both complexity and randomness (chaos) as being inherent attributes of AI [80]. Stair and Reynolds expand upon this basic definition by including the processing and manipulation of symbols, the use of heuristics (in other words, benefiting from experience), as well as notions

⁵ Although Alan Turing had published an earlier paper on ‘Computing machinery and intelligence’ (*Mind*, 1950, 59:433–460).

of ‘creativity’ and ‘imagination’ (and perhaps also ‘inspiration’?) [283]. Some authors take this much further, suggesting that one day artificial intelligence will surpass that of human intelligence [171, 202]! At the other extreme there are those who believe human intelligence is invariably superior to that of machines (<http://www.mturk.com>).⁶

The original focus of AI centered not only around making better (more ‘intelligent’?) computers, but also on computational psychology and/or philosophy. The goal with the former was to glean an understanding of (intelligent?) human behavior by creating programs which behaved in the same manner as people (such that they satisfied the Turing test) [244]. The goal with the latter was to formulate a ‘computational understanding’ of such behavior (it is debatable however as to whether intelligence can be consider a *computational* entity) [261].

Much early effort in AI was directed towards automated theorem proving – for example, **Logic Theorist** and the later **General Problem Solver**, **SOAR**, **SAINT** (**S**ymbolic **L**ogic **I**NTerpreter), and the **Geometry Theorem Prover** [261]; more recent examples are **SAM**, **AURA** and **OTTER** [109]. Another popular area has traditionally been game theory – such as checkers (draughts) [262] and chess (<http://www.research.ibm.com/deepblue>). Other early work centered around Microworlds [234], and high-level languages, in particular the **LIS**t **P**rocessing language (**LISP**) and the **LOG**ic **P**ROgramming language **Prolog** (a major thrust of the Japanese 5th Generation Project), although **C/C++** remains the preferred choice for many researchers (for instance, the **C-Language Inference Production System** or **CLIPS**).

Real-world applications of AI have subsequently tended to concentrate in the areas of robotics, computer vision, and most especially Expert Systems (Sect. 5). The approach taken with the latter can be characterized as the use of knowledge and reasoning in order to solve complex problems.

Knowledge Representation and Ontology

The challenge for a Knowledge Engineer (KE) is to first extract knowledge from a (human) domain expert, then encode it into an appropriate format – namely one that facilitates not only efficient storage within a Knowledge Base (KB), but also efficient searching of this KB when the system is presented with new user queries. Finally, the KE needs to consider how best to present/display system responses to such queries (along with corresponding confidence levels) to the user – in other words, in an easily understandable and aesthetically pleasing manner. Not only that, but users typically also expect some

⁶ For a good summary of open questions pertaining to AI, the reader is referred to <http://www.openquestions.com/oq-te019.htm>

form of rationale/justification for the decision(s) arrived at by the system – an especially difficult proposition for an Artificial Neural Network (Sect. 7), but a straightforward one for a Decision Tree (Sect. 2.3).

Several different techniques exist to assist a KE with the gathering of knowledge pertinent to a particular domain, including structured or unstructured interviews, focus groups, direct observation, case studies, and so forth [27]. Likewise, various alternatives are available for representing knowledge thereby obtained. These include **if..then** production rules (or alternatively, fuzzy rules if the knowledge is inexact, imprecise, or indeed at times, contradictory), formal logic (for example, First-Order Logic), frames (which bear some resemblance to a ‘record’ data structure), semantic networks (a graphical technique), Artificial Neural Networks, and so on [281].

These days a KE will also most likely be concerned with constructing an ontology of the relevant application domain – ‘ontology’ being that branch of metaphysics concerned with the nature of being, fundamental principles, and categorizations. In an AI context, ‘ontology’ is taken to mean the specification of a system of concepts and the relationships between them, usually in machine-readable form.

Logic and Reasoning

Formal logic, as its name suggests, concerns itself with ‘form’ (syntax) but not necessarily meaning (semantics). Propositional logic is a formal system in which propositions can be generated by combining atomic (‘true’ or ‘false’) propositions using logical (for example, Boolean) operators and formal ‘proofs’ in order to establish ‘theorems’. First-Order Logic (FOL) or Predicate Calculus is an extension of propositional logic which utilizes not just statements and connecting operators, but also variables, functions and predicates, in order to support quantification over individuals within a given domain (universe of discourse). FOL is sufficiently expressive to be able to formalize *most* mathematical concepts. Second-Order Logic extends FOL in order to support *sets* of individuals. Higher-Order Logic is a further extension which allows additional constructs (higher-order predicates which take more than a single predicate as arguments) according to an underlying type theory formalism. HOLs allow the quantification not only of objects, but also of relations between them, functions, and ontologies, however reasoning within a HOL is not straightforward (it is for this reason that AI/Machine Learning has traditionally used FOL and/or the λ -calculus to represent knowledge in a manner suited to both storage within and searching of a Knowledge Base (KB)). λ -calculus is a formal system developed to support function definition, application and recursion; it in turn influenced the subsequent development of functional languages such as LISP and ML. By contrast, FOL facilitates the use of PROLOG for the processing of user queries.

Reasoning about knowledge is possible using logical inferences, and by incorporating either forward- or backward-chaining. With the former, we start with the known facts (data/information), and proceed towards the goal; hence we characterize this approach as a ‘data-driven’ one. We proceed in the reverse direction with the latter; hence this approach can be characterized as being ‘goal-directed’. In either case, searching of the state (solution) space could proceed in one of several different ways. If we represent knowledge about a particular domain as a tree structure, then we could search from the root node down to the leaf nodes in either a breadth-first or depth-first manner, by using exhaustive search, or by employing some heuristic(s) (in other words, to exploit *a priori* knowledge about the domain in question). Whatever approach we take, we will invariably reach ‘dead ends’ and need to backtrack up the branches of the ‘knowledge tree’ in order to explore a new search path.

We should point out however that an over-reliance on sequential causality probably contributes to the ‘brittleness’ of logic-based AI. For example, in Immunity-Based Computing (Sect. 9.1), we also need to take into account *circular* causality.

Belief (Bayesian) Networks [147, 161, 216] are directed acyclic (or cyclic) graphs which can be used to represent entities and the (causal) relationships between them, as well as their conditional probabilities. They allow efficient reasoning about uncertainty. An alternate ‘world view’ (ontology) can be described in terms of intelligent (cognitive) beliefs-desires-intentions or BDI agents.

Classifiers

We can state the basic classification problem as being a transformation:

$$\Gamma^n \implies \Gamma^m \tag{1}$$

where n -dimensional input data is transformed into m discrete classes (categories), with $m \ll n$ typically.

Usually we will train a classifier on the available data, although *unsupervised* classifiers are also possible (however the clusters they form may not always be sensible ones!). Once a classifier has been trained, it will be capable of generalizing what it has learnt and be able to correctly classify input patterns it has not previously met, but which are nevertheless ‘similar to’ (in a geometric or vector sense) patterns it knows about, providing of course that the new pattern is drawn from the same overall population. Several different approaches can be followed in creating supervised classifiers, including statistical, Decision Tree, SVM, ANN, GA, to mention but a few.

Now a classifier will only be able to discriminate between a finite number – p – of different (orthogonal?) pattern classes before they begin interfering

with each other (leading to pattern crosstalk). In the case of Artificial Neural Networks (Sect. 7), the patterns are stored in the minima (‘valleys’) of the solution space (energy landscape), assuming that the network capacity has not been exceeded. Different types of classifier are capable of forming different shaped discriminants (separators, boundaries) between pattern classes – more specifically, linear in the case of statistical regression, rectangular in the case of Decision Trees (Sect. 2.3), or arbitrary in the case of ANNs (providing a sufficient number of hidden layers and/or nodes are used, in the case of MLP/BP). Figure 1 shows the iris data set, which comprises sepal and petal length and width measurements taken from three different flower species: *setosa*, *versicolor*, and *virginica*. We observe that separation of the lower class (*setosa*) can be performed by inspection (and a straight line separator/discriminant drawn accordingly, as in Fig. 2). The other two classes are more intermixed, and hence more difficult to separate (certainly using a *linear* discriminant).

Some classifiers are only capable of forming linear discriminants (straight line decision boundaries in the case of 2D data, such as eXclusive-OR – see Fig. 2 – top); others are capable of handling non-linear input data. For instance, both the ADALINE and Rosenblatt’s original 2-layer Perceptron were only capable of acting as linear discriminators; the later Multi-Layer Perceptron was not so restricted. In general, for n -dimensional data, the discriminant (decision boundary) is an $(n - 1)$ -dimensional hyperplane (Fig. 2 – bottom).

$$P(H | X) = \frac{P(X | H)P(H)}{P(X)} \quad (2)$$

Naïve Bayesian classifiers [284] perform as simple linear statistical classifiers, and are trained using tuples of data attributes and the classes to which they belong. The basis for constructing such a classifier is the Bayes formula (Eqn. (2)), which enables the *a posteriori* conditional probability – of an input pattern H belonging to a particular class X – to be determined based on *a priori* (known) probabilities, or estimates thereof. The ‘naïve’ here refers to the inherent assumption of class conditional independence (in order to reduce computational cost); likewise, if prior class probabilities are not known, they are assumed to be equally likely. Notwithstanding these simplifications, the Naïve Bayesian classifier is often used as a benchmark with which to compare other approaches.

Mathematical Function Approximation

The fundamental curve fitting (modeling) problem is to fit a mathematical function which passes through as many of the given data points as possible, with minimal overall error. In fitting such a function to the data, one must be mindful of not *over*-fitting – more specifically, while a higher-order function may pass through more data points, a lower-order function

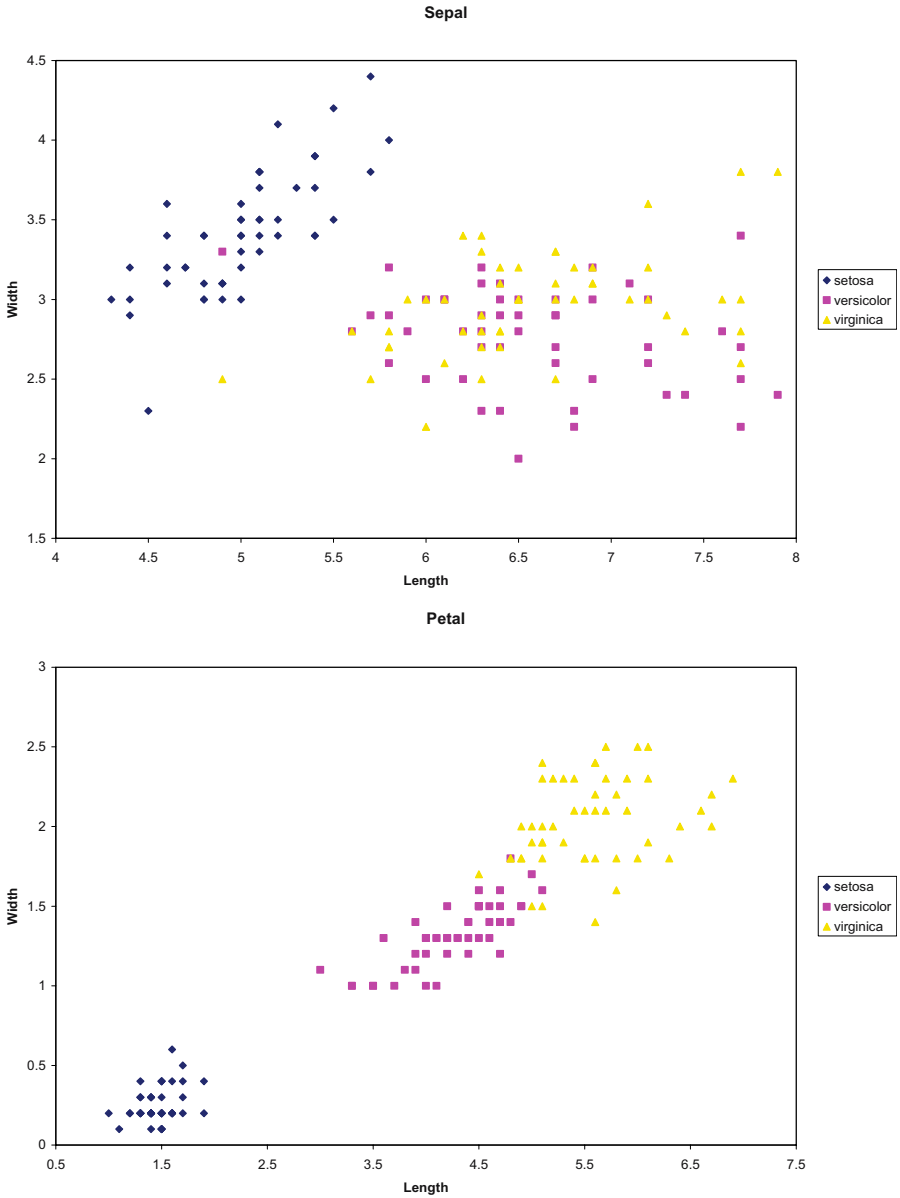


Fig. 1. UCI-ML iris data set

(such as a quadratic or set of cubic splines) may result in a better graphical representation of the data in question. For instance, in the case of foreign exchange rate or stock market time series data, it may not be so much the intermediate values that may be of interest, but more the daily, weekly or monthly *average* values.

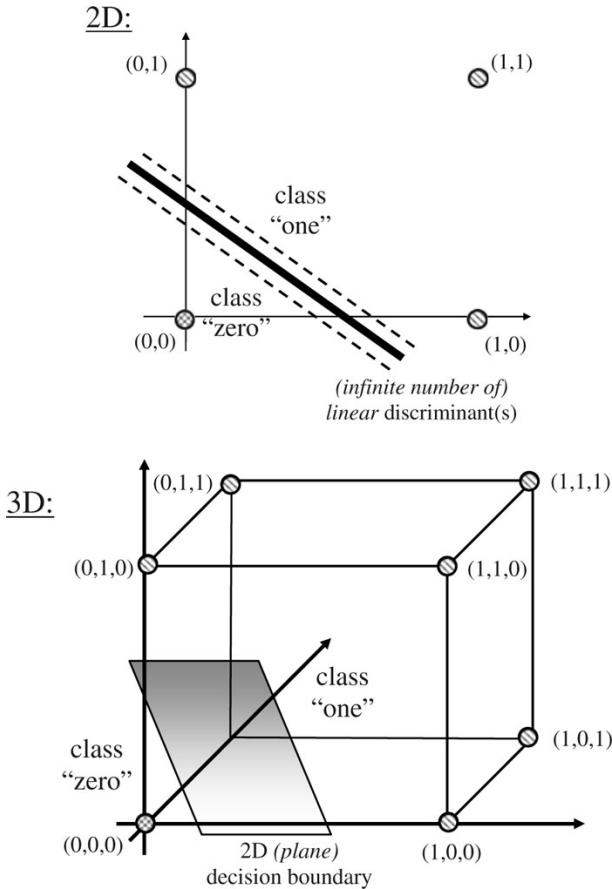


Fig. 2. 1D (linear) and 2D (planar) discriminants/decision boundaries (XOR)

Standard statistical approaches to mathematical function approximation and/or time series modeling include (simple) regression, Auto-Regressive Moving Average (ARMA) – and variants thereof – and in more recent times, Support Vector Machines [1, 270, 297] and rough sets [140, 189]. ANN approaches include Multi-Layer Perceptron/BackPropagation, Radial Basis Function, Higher-Order Neural Networks [328], recurrent networks and Time Delay Neural Networks, with the latter two incorporating time delay (memory) elements, in order to retain knowledge of *previous* network weights in order to better predict *future* ones. Higher-order ANNs (HONNs) incorporate not just the familiar sigmoid and summation neurons, but also ones with multiplicative activation functions, to better model highly complex, non-linear, discontinuous real-world data [328].

Modeling of a given data set is often only half the story however; often we are more interested in predicting future values, based on having first modelled

them as accurately as possible – not always easy if the system in question is chaotic rather than deterministic (or even stochastic), let alone in the face of discontinuities (such as a Stock Market ‘black tuesday’, say).

Figure 3 shows a long-term (top) and medium-term (bottom) view of the exchange rate between the Australian and US dollars. The former may be of use to Historians in discussing the effects of de-regulating the Australian dollar in December 1993, for instance. The latter may be of more interest to regulators in setting official interest rates, say. Yet again, shorter-term views (monthly, weekly, daily...) may be more useful for financial market speculators. In short, on some occasions the short-term variations in such time series will be important; at other times average values moreso.

2.2 Machine Learning (ML)

Any discussion of Machine Learning (ML) necessarily raises the question as to what constitutes ‘learning’ in general (in a similar way that ‘AI’ necessarily led to a consideration of the concept of ‘intelligence’). Again, such questions border on the philosophical. Typical dictionary definitions include:

- ‘to gain knowledge of or skill in, by study or experience’
- ‘to commit to memory’
- ‘to be told about, be informed of’
- ‘to become aware of’
- ‘to receive instruction, be taught’
- ‘the modification of behavior through interaction with the environment’, and so on.

We can define information as being data (facts) plus meaning; likewise knowledge we can regard as being information coupled with understanding. Similarly, we could define wisdom as knowledge plus experience together with insight (and so on, for concepts such as consciousness, self-awareness, and the like). Accordingly, we contrast people versus machines as follows:

- human = body + mind + soul (*consciousness*)
- machine = body + mind (*or just brain?*)

If the latter possesses just a brain (the physical organ) and not a mind,⁷ then it makes sense to talk about logic, reason, rules, ‘knowledge’ (providing it can be expressed in the form of **if..then** rules), even ‘learning’, but what about ‘intelligence’? As to the ‘Ghost in the Machine’ (or soul), again we shall leave this as an exercise for the reader. On the basis of the above definitions, Machine Learning would appear to be a valid concept; by contrast, the concept of ‘AI’ is much more problematic (see earlier). Indeed Naus is quite skeptical:

⁷ We chose not to enter into a debate here as to whether these terms are synonymous – in other words, whether the mind is nothing more than a biological (physical) mechanism; readers so inclined are referred to [33, 44, 45, 63, 245].

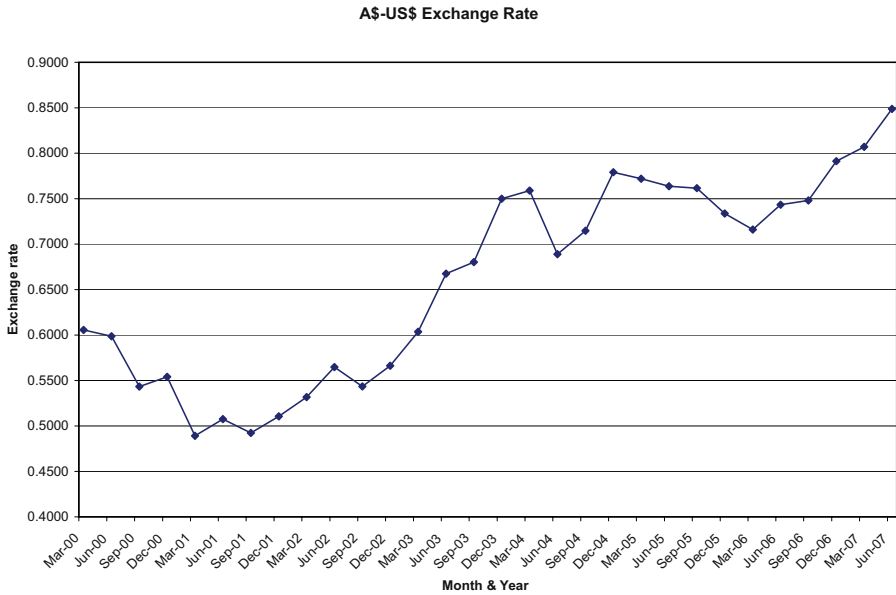
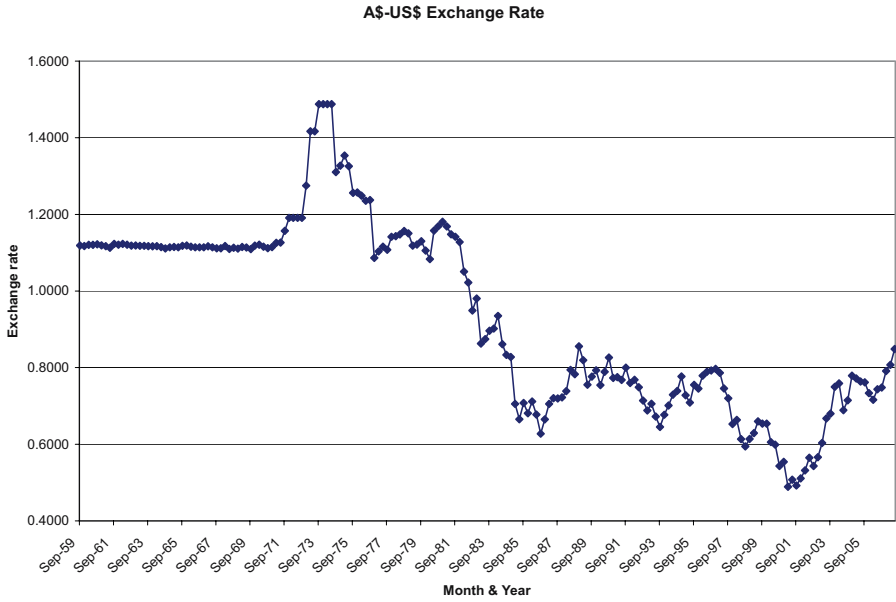


Fig. 3. Financial time series (Australian Bureau of Statistics)

“Present-day authors argue about mental life from totally defect, confused cognitive notions, in terms such as ‘consciousness’, ‘knowledge’, ‘language’, ‘intelligence’, ‘concept’, that denote nothing clearly, and moreover that William James’s insight into human mental life as presented in his *Principles of Psychology*⁸ is unknown.” [211]

Machine Learning takes an algorithmic approach to learning, adaptation and optimization; by contrast, Computational Intelligence focuses on *non*-algorithmic, data-driven approaches, as previously mentioned in Sect. 1. These ML algorithms can range from supervised learning, through unsupervised learning, reinforcement learning, and more. ML incorporates both deductive and inductive techniques. With the former, we reason from premises (assumptions) to conclusions (axioms or certainties). In the latter, we attempt to extract rules and/or patterns from the available data (typically using statistical or Data Mining [271] techniques); the results are probabilities (likelihoods) rather than certainties. The classic example in this regard is the following:

- *premise*: all men are mortal
- *premise*: Socrates is a man
- *conclusion*: (therefore) Socrates is mortal.

2.3 Decision Trees

Decision Trees – DTs – are a classification technique which grew out of both the statistical [24] and ML fields [250, 251]. They are supervised learning methods, with the tree being constructed (‘grown’) according to the attributes that characterize a particular data set. A test is performed on the most significant attribute (determinant) of a data record at the root node, and a new branch grown, depending on the result of this (Y/N) test. The next most important attribute is then tested, and so on, until we terminate at a leaf node – which corresponds to a specific class to which this particular training datum belongs. One could liken the fundamental divide-and-conquer approach of DTs to that of Principal Component Analysis (PCA), in the sense that the most significant data attributes determine the uppermost ‘limbs’ of the tree.

Classification of *new* data corresponds to finding the leaf node (solution) which most closely matches the input data (‘most closely matching’ here in terms of either mean squared error, Euclidian distance, vector (dot) product, or some other geometric/Cartesian measure). Searching of the solution space (tree) can be undertaken in either a breadth-first or depth-first manner (with or without backtracking when we find ourselves proceeding down a ‘dead end’), exhaustively (in other words, by visiting *all* branches in turn), or by invoking some form of heuristic in the decision making (branching) process.

⁸ James W (1890) *The Principles of Psychology*. Henry Holt, USA (reprinted by Dover, 1950).

Pre-processing of the data *prior* to commencement of tree growth is critical (in other words, so that noise does not negatively impact the growth process). More specifically, tree ‘pruning’ is crucial in reducing training times, which typically vary as $O(2^p)$, where p is the number of branches, to guard against combinatorial explosion.

Once grown, a DT can be used to classify data it has not previously encountered (but which nevertheless is drawn from the same population as that used to construct the tree). Put another way, they are able to generalize, but perhaps not as well as ANNs (see Sect. 7).

3 Approaches to CI

The two fundamental approaches to AI can be characterized as left- *versus* right-brained; the former refers to the traditional logical, rule-based, model-driven algorithmic approach, while the latter (data-driven) connectionist approach mimics the intuitive/creative/artistic side of our brains. Connectionism, intelligent systems, soft computing, Computational Intelligence is the primary focus of this Compendium.

Modeling complex systems using formal (higher-order) mathematical equations is not always feasible (or even possible) in many real-world situations, hence the appeal of *non*-algorithmic approaches. In other words, if modeling a certain complex system is proving intractable, why not instead try to *learn* the system characteristics (for instance, the Input-Output describing/ transfer/system function in the case of an industrial plant or process)? This will only be feasible if we have a sufficient number and diversity of (labeled) input-output training data (examples) at our disposal for learning purposes however.

3.1 The Intuitive Appeal of Nature

Traditional AI attempts to mimic human thought processes by means of models, logic, reasoning, heuristics, encapsulated knowledge, symbolic logic and the like. CI (modern-day AI?) instead attempts to create intelligent machines and/or processes by copying biological behavior – ‘mimicking Nature for problem solving’, as so succinctly put by the *IEEE Computational Intelligence Society*. Some of these natural structures/processes which have provided inspiration in the past have been [277]:

- evolution; natural selection (*phylogeny*) \implies evolutionary approaches
- multi-cellular organisms (*embryology; ontogeny*) \implies cellular automata [192, 276, 279, 292, 310]
- biological brains; cerebral cortex; the nervous, immune and endocrine systems (*epigenesis*) \implies ANNs; immunity-based systems [61, 201]
- social organisms; insect swarms, bird flocks, shoals of fish \implies swarms; artificial life [185]

This list is not exhaustive, and no doubt more natural phenomena will inspire researchers in the years to come.

There is a qualitative difference between biological and artificial systems. The former can be characterized as being probabilistic, inexact/imprecise, capable of performing 1-to-many mappings between input and output, and memory not being exact but even deliberately ‘forgotten’. By contrast, the latter are typically characterized as being deterministic, exact/precise, restricted to 1:1 I/O mappings, and with ‘perfect’ memory/recall (in other words, no forgetting).

There are numerous structures and processes in Nature which can serve as inspiration for computational methods which may be capable of superior performance compared with conventional statistical (and/or logic-based and/or algorithmic) techniques. We hasten to add that such natural structures (‘hardware’) and processes (‘software’) are not in the main fully understood, even by experts in their respective disciplines. Nevertheless, such structures/processes can serve as the inspiration for alternative computing paradigms, in a similar way that birds served as the inspiration for the development of aeroplanes, but with lift being achieved in the latter via a different mechanism than the flapping of wings (namely, by differential air pressure either side of *fixed/rigid* aeroplane wings).

As a sideline to the aforementioned, we have also managed to develop methods which better reflect human thought processes, ones capable of handling ‘grey areas’, and not just (‘black-and-white’) Boolean logic constructs – one such example of course is Fuzzy Logic (see Sect. 6 of this Chapter, as well as Part-VII of the Compendium proper).

Traditional computation has always been hampered by the so-called ‘semantic gap’, meaning the gap between the high-level language (HLL) constructs of human users which map natural language (linguistics) onto the low(machine)-level of the underlying hardware on which computer programs execute. Compilers – specialist software/programs – attempt to bridge this gap. Traditional AI, being underpinned by formalism and logic, while being able to capture *syntax* (form) has not always been successful in capturing *semantics* (meaning). This, by the way, has been a common failing with Software Engineering (SE) – hence the emphasis there on requirements elicitation (problem definition and/or specification). As we shall see in Sect. 5, this has also hampered many efforts in Expert Systems, inasmuch as it is not always easy to capture, let alone adequately represent, a domain expert’s knowledge. Fuzzy logic can assist to a degree in this process, due to its ability to translate between linguistic terms and computational terms (see Sect. 6).

Indeed, this could be one reason why Fuzzy Logic, in particular, has found widespread use in real-world applications – in other words, serving as the *interface* between human language constructs and practical problem-oriented languages.

Now CI has the potential to bridge this semantic bottleneck (gap) – which, by the way, has crippled many traditional computational and/or AI efforts – and interact *directly* with the application domain (due to its inherent application-oriented, data-driven, bottom-up nature). In bypassing this (substantial) problem, CI can learn a domain (discipline) ontology *indirectly*.

Another potential advantage of CI over both traditional AI and/or computation more generally is the possibility of producing a *range* of solutions to any given problem. The latter approaches have invariably focused on maximum performance at all costs (in other words, irregardless of the available resources), and often on *virtual* problems – to use a driving analogy, with the driver’s foot ‘planted firmly on the accelerator pedal’. CI offers the possibility of trading off performance against resources, and moreover with regard to *real-world* problems, as indicated in Fig. 4.

For instance, relaxing the termination criteria for a ANN/EA (number of epochs/generations) in order to realize a sub-optimal, yet adequate solution to the real-world problem of interest. Similarly, ANN convergence to a *local*, rather than *global*, minimum in the energy (solution) landscape can oftentimes nevertheless lead to an acceptable solution in practice (Sect. 7.2).

Lastly, CI has much to contribute as simply a *pre-processing* method, rather than a complete solution *per se* to real-world problems, inasmuch as system parameters/attributes can be learnt rather than formally modelled (again, due to its inherent data-driven, bottom-up nature). Indeed, this is often the approach taken with hybrid systems (see Sect. 12). The data-driven nature of CI (in contrast to the symbolic-oriented nature of traditional, first-generation AI) has the potential to deal with data transformation *directly* – in fact this can be viewed as a corollary to the semantic gap issue discussed earlier in this section.

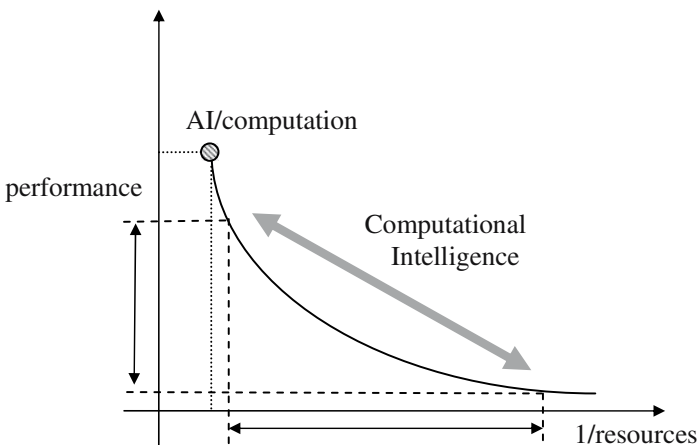


Fig. 4. Performance *vs.* resources for CI systems

Table 1. Brains *versus* computers

Brains	Computers
analog	digital
<i>(massively)</i> parallel	sequential
slow neurons	fast switches
sub-symbolic processing	symbolic processing
<i>(bottom-up)</i> data-driven	<i>(top-down)</i> model-driven
trained	programmed
fuzzy logic	precise (<i>crisp, brittle</i>) logic
fault tolerant	precise/exact (<i>fault intolerant</i>)
noise tolerant	intolerant of errors (noise)

3.2 Brains *versus* Computers

Despite the relatively slow response times of individual neurons (typically milliseconds), the brain as a whole is capable of outperforming even the fastest supercomputer (which boasts picosecond clock cycle time), and by several orders of magnitude on some tasks. It is also well known that the brain is organized into local regions/neighbourhoods which perform specific functions (such as sight, hearing, motor movement, and so forth), and that a substantial amount of pre-processing takes place in the cerebral cortex, prior to electrical signals reaching other parts of the brain. Indeed, this has served as inspiration for work on Hierarchical Temporal Memory [123] (see Sect. 13).

Now brains are able to perform some tasks much better than digital computers, and vice versa. One such task is pattern recognition, where the pattern of interest could be visual, sound, or derived from some other source. For example, if we encounter a person some distance away walking towards us down a street, we reach a certain point where we *instantly* recognize our friend; we do *not* embark upon a process of elimination – searching the state (solution) space, as it were – performing a depth search (with backtracking), in order to reach a leaf node (pattern class) which best matches our friend!

By contrast, humans in the main fare poorly at highly mathematical and/or logical tasks – ones which a digital computer can be programmed to handle with ease. Table 1 compares and contrasts brains with computers.

4 CI Paradigms

In Sect. 1 we stated that our use of the term ‘Computational Intelligence’ implied iterative, adaptive techniques inspired by Nature, and which possess the ability to learn, to deal with new situations, and to ‘reason’. In Sect. 3 we characterized this approach to AI as a right-brained, intuitive/creative, connectionist one. Thus, CI is viewed as incorporating ANN, evolutionary, and/or

Fuzzy techniques, and most especially *hybrids* of these. Moreover, as previously observed, some researchers cast their nets a little wider in defining CI, and encompass learning theory, probabilistic reasoning, constrained reasoning, case-based reasoning, Support Vector Machines, rough sets, learning/adaptive classifiers, Bayesian networks, intelligent agents, and other techniques.

4.1 Pre-Processing

Irrespective of the specific CI technique used, data pre-processing is an essential consideration (pre-condition); the adage ‘garbage in, garbage out’ certainly applies in this regard. As a first step, data visualization is strongly recommended, as we can often gain insights by so doing (the iris example of Fig. 1 is a good example of this; another pertinent example is the use of scatter plots to reveal data dependencies, since many techniques assume data *independence*). Not only do we need to concern ourselves with error bounds checking, noise filtering, dealing with missing data, normalizing, re-formatting, transformation and so forth, but reduction of the available data to a ‘minimum yet sufficient’ number is often vital. This is especially the case with ANN training (or GA evolution), since computation times typically explode as a function of data set size n , in other words $O(e^n)$. Hence data ‘cleaning’ is essential prior to training (evolution). Chapter 2 of this Compendium discusses data reduction in considerably more detail.

CI methods not only require a considerable amount of pre-processing, but also oftentimes quite a deal of parameter tuning – this will be emphasized in our separate discussions of Fuzzy Systems (Sect. 6), ANNs (Sect. 7), and Evolutionary Computation (Sect. 8).

5 Expert Systems

Georgeff and Azarmi rightly observe that:

“Although we are very far from achieving the ultimate goal of AI, which is the building of the intelligent machine, AI research and development has so far led to a myriad of spinoff technologies and techniques that are used in a large variety of applications. For example, Expert Systems...” [107]

Expert (or Knowledge-based) Systems (ES/KBS) came to the fore during the 1980s, initially in the context of interpreting molecular structure from mass spectrograms (*Dendral*), medical diagnosis (*Mycin*), DEC mini-computer configurations (*XCON*) – around 30 times faster than humans, with no errors – assisting students to solve complex symbolic math (*MACSYMA*), oil and gas exploration (Schlumberger’s *Dipmeter Adviser*), gold prospecting (<http://www.SPSS.com/clementine/hugin.htm>), machine vibration data

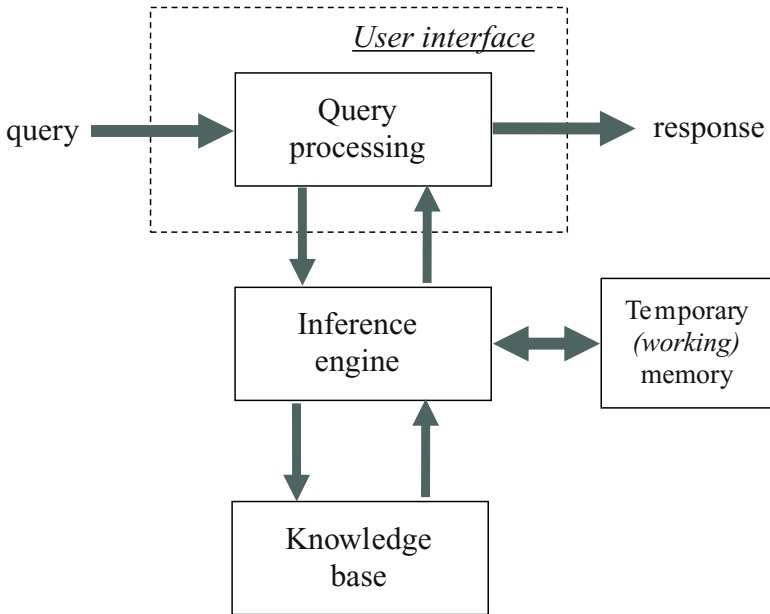


Fig. 5. Expert system

analysis (General Motors' **Charley**), and bank loan approvals, to mention but a few [79, 139, 142, 194]. The **Emycin** shell in fact became the foundation for many subsequent commercial Expert Systems.

An ES comprises an Input-Output module for handling both user queries as well as responses to same (in a similar manner to that used in a DataBase Management System), an inference engine (for the processing of user queries), a knowledge base, and a working memory for the storage of temporary (intermediate) results, as shown in Fig. 5. The *same* inference engine could in theory be used with *different* knowledge bases in order to realize a range of Expert Systems appropriate for different application domains (as with **Emycin**).

So how do we firstly encode and then store knowledge in the Knowledge Base of an Expert System? Many alternatives exist, including:

- (typically hundreds or even thousands of) **if...then** production rules,
- propositional/first-order predicate calculus (or other suitable formal logic),
- (crisp/precise) Boolean logic,
- (imprecise, inexact) Fuzzy Rules,
- Association Rules,
- frames/schema (which map naturally onto the Object-Oriented Programming paradigm) (such as **OWL**, **FRAIL**, **KODIAK**),
- graphically, in the form of a Semantic Network (such as **SNePS**, **NETL**),

- Belief (Bayesian) Networks,
- an Ontology,
- Beliefs-Desires-Intentions (BDI) within a Multi-Agent System (MAS),
- in the internal weights of a trained Artificial Neural Network,
- in the fixed length strings of a Genetic/Evolutionary Algorithm,
- or by some other means.

The inference engine uses the information stored in the Knowledge Base to develop a line of reasoning to solve the problem at hand, as already noted in Sect. 2.1. This process is referred to as ‘forward chaining’. It is also possible to proceed from the goal back to the necessary set of initial conditions, as with logic proofs, in which case it is referred to as ‘backward chaining’. In practice a mixture of both forward- and backward chaining proves more effective; likewise methods are usually incorporated for expanding the line of reasoning. Furthermore, if reasoning under uncertainty is appropriate (and more often than not this will be the case), then fuzzy logic is probably more appropriate than (crisp, exact) Boolean logic (Sect. 6).

Now both the inference engine and knowledge base could be implemented either as rule-based or neural network sub-systems. For instance, in the former knowledge is stored in the form of **if..then** production rules, whereas in the latter it is stored within the network weights.

The most difficult part in developing an ES is often in quantifying not only a domain expert’s knowledge, but also their reasoning processes and the logic behind what they do (which is not *always* logical!). The adage ‘garbage in, garbage out’ certainly applies in this context [73,312]. Experts are not always able to explain/justify why they do certain things in the way they do – just that they’ve always done it this way, and it works! A good illustration of this is James Dyson, of vacuum cleaner fame:

“But a business philosophy is a difficult thing to distil out of the daily workings of a company, because you never really know how you do it, you just do it. It’s like asking a horse how it walks.”⁹

This is an even more daunting prospect when attempting to extract and encode knowledge from *several* different domain experts, since there are bound to be disagreements, inconsistencies and contradictions in what each individual expert relates to the KE (hence an opportunity here for Fuzzy Logic, Fuzzy Cognitive Maps (FCM) [162] and the like, which are capable of handling such conflicting and/or contradictory information).

Later developments in ES involved the incorporation of Belief (Bayesian) Networks, which enable the generation of sound probabilistic inferences from the available evidence.

⁹ Dyson J (1997) *Against the Odds: An Autobiography*. Orion Business Books, London, UK: 256.

Several ES shells exist – both commercial and Open Source – to assist with the development of application-specific ES. A couple of the more popular are the C/C++ based CLIPS (<http://www.ghg.net/clips/CLIPS.html>), FuzzyCLIPS (http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyClips/fuzzyCLIPS/index.html), and Jess (a Java version of CLIPS), as well as OpenExpert (<http://www.openexpert.org/>).

6 Fuzzy Systems

Picture yourself in the early evening sitting in your favourite armchair watching television, prior to preparing the evening meal. The thought crosses your mind that soon you will enter the kitchen and begin slicing and dicing vegetables ready for steaming. Before that however, since it has become somewhat cooler since you sat down to watch television, you will need to turn on the room heater. Notice that usage of linguistic terms such as ‘soon’, ‘steaming’ and ‘cooler’ come naturally to us as humans, but that the computers used to control the TV (clock), hotplate and room heater are usually designed to handle *exact* (precise) terms like 1850 hours (10 minutes hence); 100°C, and 28°C, respectively. Fuzzy Logic allows us to cater for such ‘fuzzy’ but more natural terms (linguistic variables) in the development of computer (control) systems. Fuzzy membership functions – μ – relate linguistic variables to numeric values, by way of simple graphical shapes (typically triangular or trapezoidal). Fig. 6 shows a simple example where the same input value exhibits 30% ‘membership’ of fuzzy set-A, simultaneously with 90% membership of fuzzy set-B (in other words, x ‘belongs’ not just to A or B , but to both).

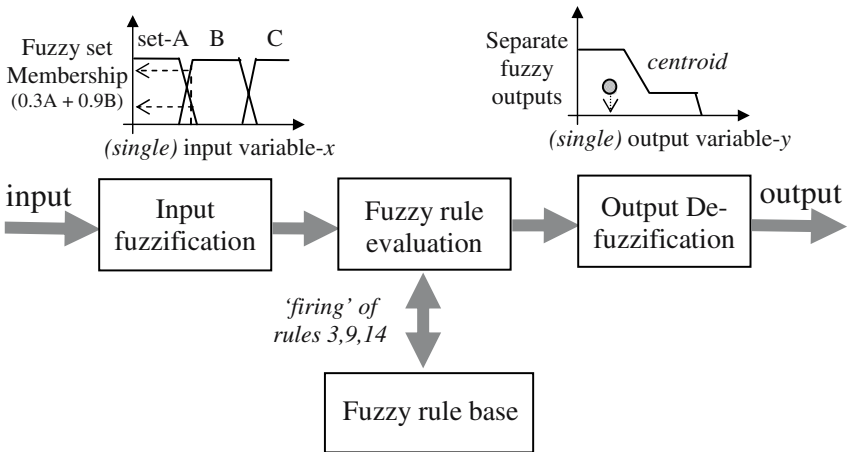


Fig. 6. Fuzzy inference system

The roots of Fuzzy Logic can be traced back to the work of Lukasiewicz on multi-valued logic (together with fuzzy set structure and the relationship to conventional logic) [195], and to that of Black on ‘quasi-fuzzy’ sets [20]. However the ‘Father of Fuzzy Logic’ is undoubtedly Lofti Zadeh, who was the first to introduce the idea of membership sets and fuzzy operators (MAX, MIN) [323]. MAX and MIN are the fuzzy equivalents of the crisp (precise, Boolean) operators logical-OR and logical-AND, respectively. Again referring to Fig. 6, these fuzzy operators could be used to derive a fuzzy rule set pertaining to this example, as follows:

$$\mu_{A \vee B} = \max\{\mu_A, \mu_B\} = \max\{0.3; 0.9\} = 0.9 \quad (3)$$

$$\mu_{A \wedge B} = \min\{\mu_A, \mu_B\} = \min\{0.3; 0.9\} = 0.3 \quad (4)$$

$$\mu_{\neg A} = 1 - \mu_A = 1 - 0.3 = 0.7 \quad (5)$$

Despite some early successes with the development of fuzzy controllers during the late 1970s and early 1980s (for instance, [15] [77] [199] [287]), research funding basically dried up during the ensuing decade [57] [205]. We can attribute the resurgence of Fuzzy Logic as a popular control method to the persistence of Japanese white goods manufacturers during the 1980s [133].

Fuzziness refers to the inexact or imprecise nature of common, everyday terms [323]. Fuzzy set membership allows the description of such fuzzy terms, by enabling them to belong to more than one set (30% of set-A, concurrent with 90% of set-B, say), in contrast to ‘crisp’ logic which only allows membership (yes/no) of a *single* set [324]. Fuzzy logic is said to be a generalization of conventional (two-valued) logic which enables us to perform operations on fuzzy sets – in other words, it constitutes a form of ‘approximate reasoning’ [325].

Figure 6 shows the basic components of a Fuzzy Inference System (FIS), these being Input Fuzzification, the Fuzzy Rule Base, Fuzzy Inference Engine (rule evaluation), and Output De-Fuzzification. In the example of Fig. 6, the preconditions (premises) of rules number 3, 9 and 14 have been satisfied, and so these rules are ‘fired’ (the antecedents/consequents activated). Converting discrete (exact) inputs into their fuzzy equivalents is performed on the basis of their membership of one or more fuzzy sets, as described earlier. Conversion back from numerical values to linguistic terms (de-fuzzification) is accomplished using geometrical constructs like ‘centroid’, as indicated.

As emphasized in Sect. 4.1, pre-processing is a critical consideration in implementing a Fuzzy System (as it is with *any* CI technique) – more specifically, the choice of linguistic variable terms, the number of fuzzy sets, and the set membership functions/shapes (triangular, trapezoidal, and the like). As is also the case with other CI methods, there is typically no proscriptive formula for determining optimum operating parameters – this more comes as a result of trial-and-error and/or experience.

7 Artificial Neural Networks

The origins of Artificial Neural Networks (ANNs) go back even further than those of Artificial Intelligence (AI). The simplified individual neuron model most favoured by researchers and users alike stems from the 1940s [204] (Fig. 7). Theories of ANN learning [14,309] and indeed the prospect of building ‘brain-like’ computers [302] likewise stem from the mid 20th Century.

7.1 ANN Types

ANNs can be classified along many dimensions, including supervised *versus* unsupervised, feedforward *versus* feedback, and so forth. Supervised networks require a ‘sufficient’ number of labeled Input/Output data pairs (exemplars), not only for training, but also for testing the network once it has converged. Such data is not always available, hence the appeal of *unsupervised* networks, which only require input data, and which form their own output classes (but which may or may not be meaningful!) [40,160]. In practice, such unsupervised networks are often used as a preprocessing stage prior to a supervised network classifier proper. In feedforward networks, connections (weights) are only present in the forward direction, from input layer to output layer; no connections exist in the reverse direction. If such connections *are* present, as with recurrent networks (such as Hopfield), then the network behavior is considerably more complex, since we now have the possibility of resonance occurring in the network.

Biological Plausibility

It is well known that the McCulloch & Pitts neuron model is quite a simplistic one. By contrast, in biological neurons, outputs are produced in the form of

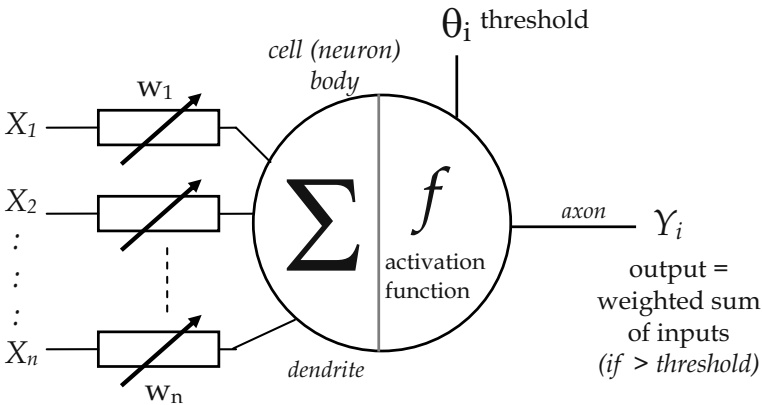


Fig. 7. McCulloch & Pitts neuron model

pulse trains as a result of *electrochemical* processes (reactions), rather than simple (electrical) level shifts – indeed, this has prompted some researchers to pursue the development of Pulsed ANNs [196]. Likewise, synapses are not usually restricted to forward connections in the brain, as is the case with MLPs, or symmetrical weights, as assumed in the Hopfield model (a constraint employed in order to simplify the mathematics) [137]. There is however *some* justification for the localized (and topology preserving) organization and behavior of SOMs [160]. The point here is that biological plausibility is not essential, provided the technique in question works (refer back to our earlier comments in Sect. 3.1 regarding inspiration from Nature).

7.2 Multi-Layer Perceptron/BackPropagation

During the 1960s, networks were favored which were suited to linear systems – typical of these being the ADALINE [308] and the (2-layer) Perceptron [257, 258]. From the former came the Least-Mean Square (LMS) or Delta learning rule, which was later revamped into the Generalized Delta learning rule (BackPropagation). Enthusiasm for Perceptrons began to wane when it became apparent they could only be successfully applied to linearly separable data; they could not be used on linearly *inseparable* data, such as eXclusive-OR. Indeed, funding for ANN research underwent a so-called ‘neural winter’ as a result of the publication of [209].

Interest in the Perceptron – or more especially the *Multi-Layer* Perceptron – was rekindled a decade or so later with the development of the BackPropagation (BP) learning algorithm [37, 235, 260, 305, 306].

Figure 8 shows a 3-layer MLP. Presentation of input-output training pairs (exemplars) will generate errors at the output layer between the actual and desired output values. The difference between these two values is then used to adjust the weights – firstly those connecting the hidden layer to the output layer, followed by those connecting the input layer to the hidden layer – in order to minimize this error or difference signal. In this manner, the error propagates *backwards* from the output layer towards the input layer, adjusting the network weights as it does so. The problem with this procedure is that once we have presented all training exemplars (one epoch), the weight values will have been changed in completely different directions from the first weight changes. In practice, *many* passes through the training data (epochs) are typically required in order to reduce the overall weight error value to a minimum – at which point the network is said to have ‘converged’ to a solution (in other words, it has been ‘trained’). This process can be likened to ‘gradient descent in weight space’, which is an optimization technique known to suffer from several limitations, namely (i) oscillation about a minimum (rather than convergence *perse*), and/or (ii) convergence to a local, rather than a global minimum (in the energy/error landscape or solution space).

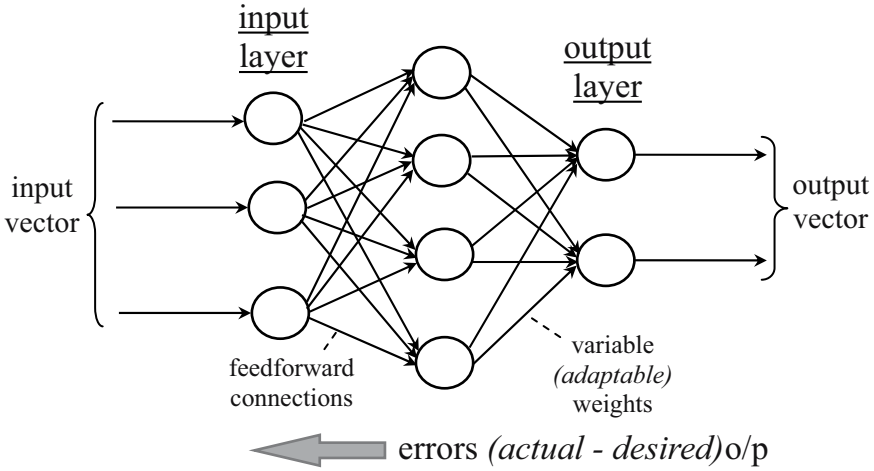


Fig. 8. Multi-Layer Perceptron/BackPropagation

Now whereas previously Perceptrons proved to be of limited use, MLPs have been shown to be suited to solving numerous pattern recognition and/or classification problems, despite their being hampered by long training times (although it has been proven that BP will *eventually* converge). They are also particularly well suited to mathematical function approximation (curve fitting), and by extension to time series modeling, simulation and/or forecasting. We have to be careful here though not to *over-fit* the data though, as previously mentioned (in other words, not to over-train the network).

Interest in ANNs in general was also fostered by the publication of [5, 131, 137, 160], to name but a few. Since that time, MLPs have been applied to many different problem domains, sometimes indiscriminately so (in other words, ‘throwing’ an ANN at a problem when conventional techniques fail!). Indeed, when the lay person refers to ‘neural networks’, this is usually synonymous with MLP/BP. For instance, [314] reported that over 95% of ANN business applications use MLP/BP. A typical pattern classification application to which MLP/BP has been applied is the discrimination between cancerous and benign breast tumours on mammograms [300].

It should be noted that often the most difficult part of applying ANNs in practice is transforming the available training data into an appropriate format prior to training, most especially by removing noise, reducing the dimensionality of the data (since training times increase exponentially as a function of the number of network weights), feature extraction, and so on (see Sect. 4.1 on Pre-Processing).

Optimal network parameters (numbers of layers/nodes/weights) are typically determined in practice by a process of trial-and-error, complemented by

experience. Alternatively, various researchers have devised heuristics like the following [124]:

- number of hidden neurons $H = \text{geometric mean of } I \text{ \& } O$ (number of neurons in the input & output layers, respectively)
- no need for any more than 2 hidden layers (\rightarrow *arbitrary* decision boundaries) – Kolmogorov Representation Theorem [59]
- good generalization ability if training set size $N \geq W/\varepsilon$ (where $W =$ the number of network weights + thresholds (biases); and $\varepsilon =$ the maximum permissible classification error)
- $W \leq N \leq 10xW$
- $\eta = \frac{1.5}{\sqrt{n_1^2+n_2^2+\dots+n_m^2}}$ (where $\eta =$ the learning rate; $n_i =$ number of (training exemplars in class- i)

Now MLP/BP training times are notoriously slow, indeed [150] has shown that finding a set of ANN weights consistent with a set of examples is NP-complete! Accordingly, numerous attempts have been made to improve the convergence of BP – in other words, to speed up training times. These range from simply adding a momentum term (α) to the weight update rule,¹⁰ to more sophisticated approaches, such as:

- making use not only of the first derivative of the weight, but also of the *second* derivative (Conjugate Gradient;¹¹ QuickProp [84])
- using the *sign* only of the weight derivative, not the magnitude (Resilient Back Propagation – Rprop [255])
- making use of *previous* weight changes (Delta-Bar-Delta [143]; SuperSAB [293])
- dynamically *growing* a network of ‘minimum yet sufficient’ size (Cascade Correlation [84])

Often we find that some weights are quite small and moreover do not contribute much to the network behavior. Accordingly, some researchers have developed pruning techniques [121, 178], akin to those used in Data Mining (DM) [251, 313].

7.3 Other ANN Models

Many other ANN models have been developed over the years apart from MLP/BP, including learning vector quantization LVQ [159], radial basis

¹⁰ $w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j^P O_j^P$, where w_{ij} is the weight connecting node- i in the previous layer to node- j in the present layer; η is the learning rate ($0 \cdot \cdot 1$); δ_j^P is the delta (difference or error) between the actual and desired outputs, and O_j^P the output generated on node- j , following presentation of input pattern- P .

¹¹ <http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf>

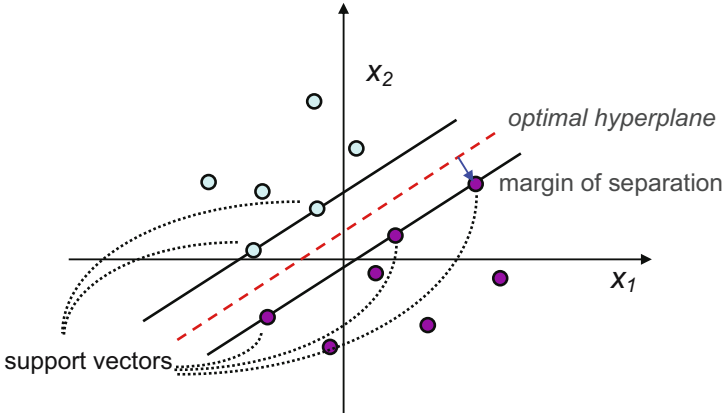


Fig. 9. Support vector machine (linear separator)

function RBF network [248], support vector machine SVM [1, 270, 297] (the latter cannot strictly be regarded as an ANN technique, but rather a statistical one; nevertheless, it has sparked a lot of interest in the ANN community). Now RBF and SVM, while utilizing the same general architecture as MLPs, behave quite differently internally. RBF activation functions are usually more complex than the simple sigmoid¹² typically employed by MLP/BP – typically Gaussian. Applying RBFs can be likened to fitting a mixture of Gaussians, and hence are quite suited to mathematical function approximation. Their training times are comparatively short, but they take a long time to fit test data (which is the exact reverse of MLP/BP).

SVMs use a non-linear mapping to first transform the data set of interest into a higher (feature) dimension, then to find an optimal linear separating hyperplane (decision boundary/discriminant) to separate one class from another. They find this hyperplane using ‘support vectors’ (attractors) and ‘margins’, as indicated in Fig. 9. These margins are defined as follows:

$$\mathbf{w}^T \mathbf{x} + b = 0 \quad (6)$$

where \mathbf{w} is the (adjustable) weight vector, \mathbf{x} is the input training vector, and b is the bias (or threshold) term. The distance from \mathbf{x} to the optimal hyperplane \mathbf{g} is

$$\mathbf{g}\mathbf{x} = \mathbf{w}_0^T \mathbf{x} + b_0 \quad (7)$$

The optimum parameters \mathbf{w}_0 and b_0 provide the maximum possible separation between positive and negative training exemplars, and are obtained by minimizing the Euclidian norm of \mathbf{w} . As it happens, SVMs are quite capable of classifying both linear and nonlinear data; indeed the latter can be extended to so-called ‘kernel methods’.

¹² $y(x) = \frac{1}{1+e^{-x}}$.

Now several researchers have shown that MLPs behave as universal approximators, namely:

“An MLP with an arbitrary bounded non-constant activation function is capable of universal approximation. More specifically, any suitably smooth function can be approximated arbitrarily closely by a single hidden layer MLP/BP. Furthermore, this approximation improves as the number of nodes in the hidden layer increases. In other words, a suitable network can always be found.” [138]

and

“A standard multilayer feedforward network with a locally bounded piecewise continuous activation function can approximate *any* continuous function to *any* degree of accuracy if and only if the network’s activation function is not a polynomial.” [182]

[329] subsequently proved a similar result for ANN groups:

“Consider a neural network piecewise function group, in which each member is a standard MLP, and which has a locally bounded, piecewise continuous (rather than polynomial) activation function and threshold. Each such group can approximate *any* kind of piecewise continuous function, and to *any* degree of accuracy.”

8 Evolutionary Methods

Evolutionary computation (EC) is a rather broad term which encompasses optimization techniques which employ evolutionary principles [65]. To the lay person this approach is synonymous with Genetic Algorithms (GAs); in reality, there are four distinct branches, albeit with GAs dominating in practice. More specifically, evolutionary computation encompasses genetic algorithms, evolutionary programming (EP), evolution strategies, and genetic programming (GP).

Obviously the inspiration for evolutionary methods goes back to the 1850s and the work of Charles Darwin with his theories of evolution, natural selection and ‘survival-of-the-fittest’. Some of the earliest work on the development of GAs took place during the late 1950s. Fraser first encoded the epistasis function parameters¹³ of certain *biological* systems as 15-bit strings, then chose as suitable parents those strings that produce function values within the prescribed range [96, 97].

Around the same time – the early 1960s – [135] developed the now familiar reproduction-crossover-mutation cycle within the context of adaptive (*artificial*) systems; accordingly, John Holland is nowadays regarded as the ‘Father

¹³ Epistasis measures the degree to which a particular gene is suppressed.

of GAs'. The term 'genetic algorithm' was coined during the late 1960s [8]. Interest in GAs took off during the 1970s and continued unabated into the 1980s and beyond [62, 114, 116, 136]. During this period DeJong devised a set of five GA test functions together with two performance metrics [64]. A decade or so later, [264] introduced the idea of using a multi- (rather than single-) GA objective function [54, 148, 289] (likewise in Fuzzy Systems). Allied to this are the relatively new fields of parallel EC [217] and memetic algorithms [119, 158].

Evolutionary programming (EP) emerged during the 1990s [94], after first appearing in the 1960s [92]. In a similar manner that Minsky and Papert received the blame for instigating the so-called 'neural winter' [209] (meaning the drying up of research funding during the ensuing decade), [92] has sometimes been blamed for the 'evolutionary computation winter'.

Evolutionary strategies can be traced back to the work of [253, 254] in the 1960s, in the context of engineering optimization. Genetic programming (GP) stems from around the same time [100, 101], but really came to the fore during the 1980s due to work of Koza [163–165].

8.1 Genetic Algorithms

In Sect. 3.1 we saw how Nature has been the source of inspiration for several CI techniques. In the case of Genetic Algorithms (GAs), it is evolution itself [114]. Darwinism or natural selection involves sexual attraction between male and female, mating, birth, and child rearing (the extent of which can vary from non-existent in the case of catfish, to many years in the case of elephants and humans), hopefully to maturity – in other words to ensure survival of the species (in the face of numerous obstacles, such as danger from various predators, variations in the weather, disease and the availability of a steady food source). Figure 10 summarizes these evolutionary processes, with crossover of genetic information being illustrated in Fig. 11.

GAs work on populations of (potential) solutions to problems of interest. These solutions need to be first encoded in the form of (fixed width) 'genetic strings'. Obviously this initial encoding process is critical. As discussed earlier (Sects. 4.1, 6 and 7.2), pre-processing is a vital consideration, as with *any* CI technique. For instance, how many bits should be used to adequately represent chromosomes (problem solutions) in the GA?

As in Nature, the aim with GAs is to evolve 'stronger' population members in successive generations, while at the same time retaining diversity within the population as a whole (in other words, to avoid inbreeding). This can be likened to Nature in which strong genes are passed from parents to offspring, while weak ones tend to die off over successive generations. We start the evolutionary process with *random* strings, then proceed to select potential mates from the current generation based on some fitness (objective) function; GAs have no knowledge of the solution (search) space.

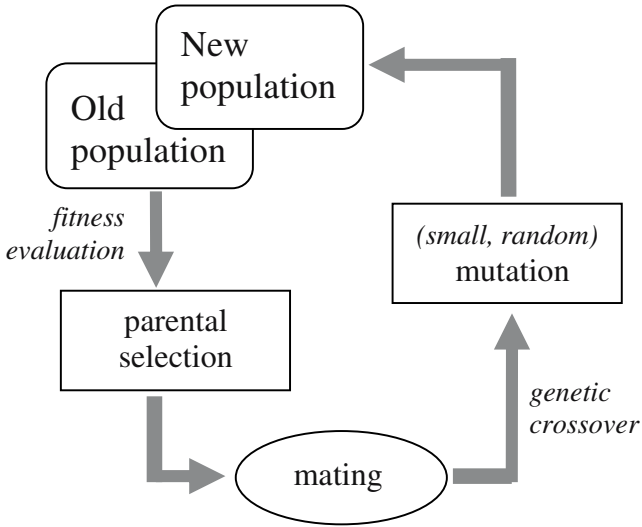


Fig. 10. The steps in evolution

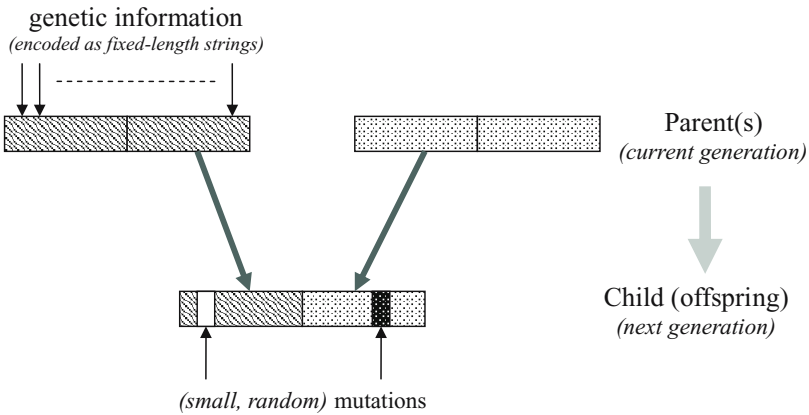


Fig. 11. Crossover of genetic information

We have just seen that a major difficulty with applying GAs in practice is the encoding of potential solutions to a problem into an appropriate form (namely, ‘genetic string’ or ‘bit chromosome’ representation). This is entirely in keeping with the general preprocessing principles outlined earlier in Sect. 4.1. Assuming this *can* in fact be done, then evolving an acceptable solution to a problem is a straightforward process, albeit a lengthy one (much longer than the time needed to train an ANN, typically) [62, 114].

Beale and Pryke coupled GAs with interactive 3D dynamic visualization in their *Haiku* system in preference to conventional rule-generation approaches (in other words, knowledge discovery becomes an interactive process, with what constitutes ‘interesting’ being established by the system user gradually over time) [12].

Evolvable Hardware

Rather than using *fixed* connections between logic gates in an integrated circuit (IC), Programmable Logic Devices (PLDs) allow such connections to be altered, either once only (as with Programmable Read-Only Memory) or repeatedly (as with EPROM or EEPROM). One such small-scale PLD is the Programmable Logic Array (PLA) shown in Fig. 12.

Since a PLA essentially comprises just one large AND-gate array followed by a similarly large OR-gate array, then potentially *any* logic function can be realized (in so-called ‘sum-of-products’ form). Addition of latches/flip flops – as in Programmable or Gate Array Logic (PAL/GAL) – further enables the fabrication of Finite State Machines (in other words, by incorporating time delay/memory elements in order to track internal state transitions). The capacities of small-scale devices like PLAs and PALs/GALs limits their usefulness however. By contrast, a Field Programmable Gate Array (FPGA) [68, 113, 241] boasts much higher chip (gate) counts – comparable with commodity off-the-shelf (COTS) general-purpose and/or Digital Signal Processors – and at a fraction of the cost of custom VLSI chip fabrication [206] (<http://www.altera.com>; <http://www.latticesemi.com>; <http://www.xilinx.com>).

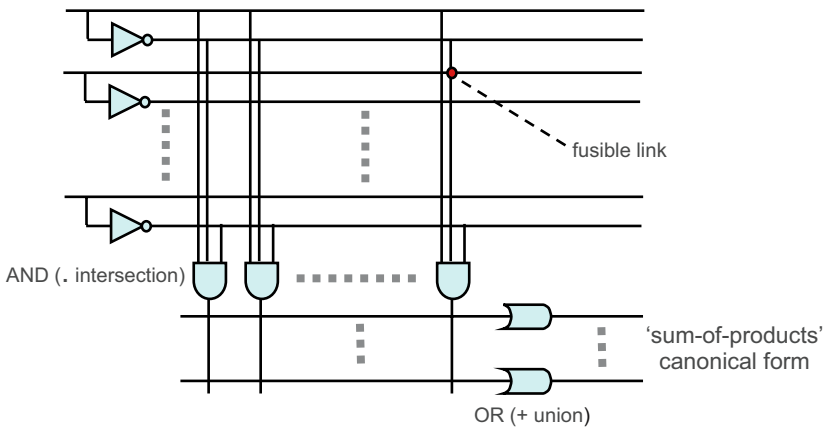


Fig. 12. Programmable logic array (PLA)

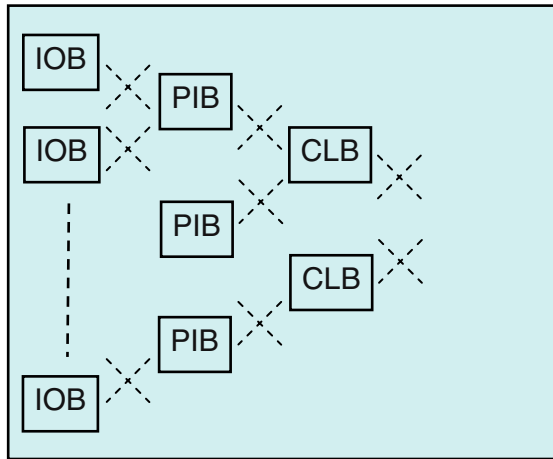


Fig. 13. Generic FPGA layout

FPGAs incorporate programmable I/O blocks (IOB), programmable interconnect blocks PIB (also known as programmable switch matrices PSM), and configurable logic blocks (CLB), with connections between these being created *in situ* by the user, appropriate for the application at hand (Fig. 13). The latter contain lookup tables (LUT), multiplexers/switches, latches, memory, and various low-level combinatorial logic ‘glue’ (recent offerings even boast entire CPUs and/or DSPs). Note that LUTs find extensive use in FPGAs since they provide much faster computation than algorithmic (software) solutions.

Designs can be developed using either hardware (building block schematics) or software (hardware description languages) form. Vendor-specific development platforms support both alternatives, together with timing simulation, testing and de-bugging facilities. Once a designer is satisfied, a bit stream is generated for downloading into the FPGA device proper. The beauty of FPGAs is that they can be *re-programmed* should the system performance not meet expectations. Some key aspects which appeal generally about FPGA designs are their (i) inherent parallelism and (ii) fast speed (both being typically much higher than COTS processors).

Such dynamic, re-programmable logic devices open up the possibility of realizing *adaptive* hardware – in other words the development of ANNs and Evolutionary Algorithms in hardware form as opposed to software simulations (currently the dominant approach). More specifically, FPGA inputs can be altered during successive training iterations according to some learning rule (ANNs) or evolutionary strategy (EAs). In the case of ANNs, the inherent parallelism of FPGAs can be exploited at the iteration, layer, neuron or weight level (the first two correspond to coarse-grain parallelism, while the last two correspond to fine-grain parallelism). Moreover, while earlier FPGAs were

limited to fixed-point arithmetic, in recent times we have seen the emergence of floating-point devices. Omondi and Rajapakse describe FPGA fabrication of ANNs – not only MLP/BP, but also Associative Memory, SOM and the Neocognitron [226]. Earlier hardware ANN implementations (not necessarily FPGA) include [109], [133] and [286].

Evolvable hardware [193, 263, 278, 332], or ‘the combination of soft computing and reconfigurable hardware’, is seen by [266] as leading to the development of ‘computational machines’. [322] define evolvable hardware (EHW) as ‘one particular type of hardware whose architecture, structure, and function change dynamically and autonomously in order to improve its performance in performing certain tasks’. Such adaptable hardware – usually realized by way of FPGAs – then has the potential to better match real world *dynamic* problems [320]. Typical of such efforts in EHW are [22, 105, 110, 129, 132, 259]. Chapters 18 and 19 of this Compendium provide more extensive, in-depth coverage of EHW.

8.2 Evolutionary Programming

Unlike with GAs, in evolutionary programming (EP), the *only* evolutionary principle used is mutation – crossover is not employed. Fogel contrasts evolutionary programming as being a ‘top-down process of adaptive behavior’, in contrast to GAs, which he likens to a ‘bottom-up process of adaptive genetics’ [94]. In this sense, it can be thought of as ‘survival-of-the-*most skillful*’, rather than ‘survival-of-the-fittest’.

8.3 Genetic Programming

Genetic Programming differs from Genetic/Evolutionary Algorithms in that the population of solutions comprise entire *programs* – in other words, automatic program generation/evolution [10, 163–165, 175]. A comprehensive coverage of GP is provided in Chap. 18 of this Compendium.

8.4 Swarms

Collective (Swarm) Intelligence (SI) [21, 83, 154] takes its inspiration from social insects (such as ants, termites, bees, wasps), as well as the swarming, flocking, herding and/or shoaling behavior common in some vertebrates. The ‘collective intelligence’ of such swarms is reflected in the ability of large groups of relatively *unintelligent* individuals to achieve feats far beyond any single individual, as a direct result of their interactions. Observation of social insects (ants, bees, and the like) suggests that intelligent group behavior emerges out of simple interactions between individuals, which otherwise

exhibit limited capabilities. Swarm Intelligence focuses on local, rather than global interactions.

Swarms differ from GAs/EAs inasmuch as in the latter individual behavior *directly* influences the behavior of future generations; in swarms, this influence is *indirect*, since individuals transmit general messages, rather than ‘peer-to-peer’ messages intended for specific individuals in the community, and which only relate to the present generation. These messages can be in the form of:

1. chemicals (namely, pheromones),
2. audio (sounds),
3. dance/stylised movements,
4. altering the surrounding environment (such as by removing all available food sources).

‘Stigmergy’ is defined as any indirect communication that allows the activities of social insects to be directed towards a common goal – for instance, ants leaving a (volatile) pheromone trail – which reinforces one particular (preferred) path over time. Collective Intelligence constraints can be summarized as follows:

- intra-generation learning only (in contrast to GAs/EAs, which foster *inter*-generational learning),
- inter-changeability and simplicity of individual population members (that is, identical form, function, and status) – although heterogeneous swarms also exist,
- reliance on indirect communication only.

Apart from intelligent swarms (as above), other variants include Swarm Intelligence (SI), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) [71]. Generally speaking, SI can be thought of as a population-based stochastic method or ‘meta-heuristic’ approach well suited to optimization problems. PSO is a global optimization method in which solutions to problems of interest are represented as points in n -D space. Particles are assigned initial *velocities*, then move towards points (solutions/attractors) over time, according to some fitness evaluation criterion. ANTS – Autonomous NanoTechnology Swarm – is a more recent development by NASA in which 100s (1000s) of small, lightweight ‘pico class’ spacecraft are able to be deployed in order to undertake exploration, provide backup, and ensure survival in space [130].

SI algorithms have been applied to both static (for example, the Traveling Salesman Problem – TSP), as well as dynamic problems (such as load balancing in telecommunications networks). In TSP, the salesman needs to travel through each city once only, in the shortest possible overall tour distance. Figure 14 shows both a sub-optimal tour of 14 cities in Burma (top), together with the optimal 14-city tour (bottom). The starting point in both

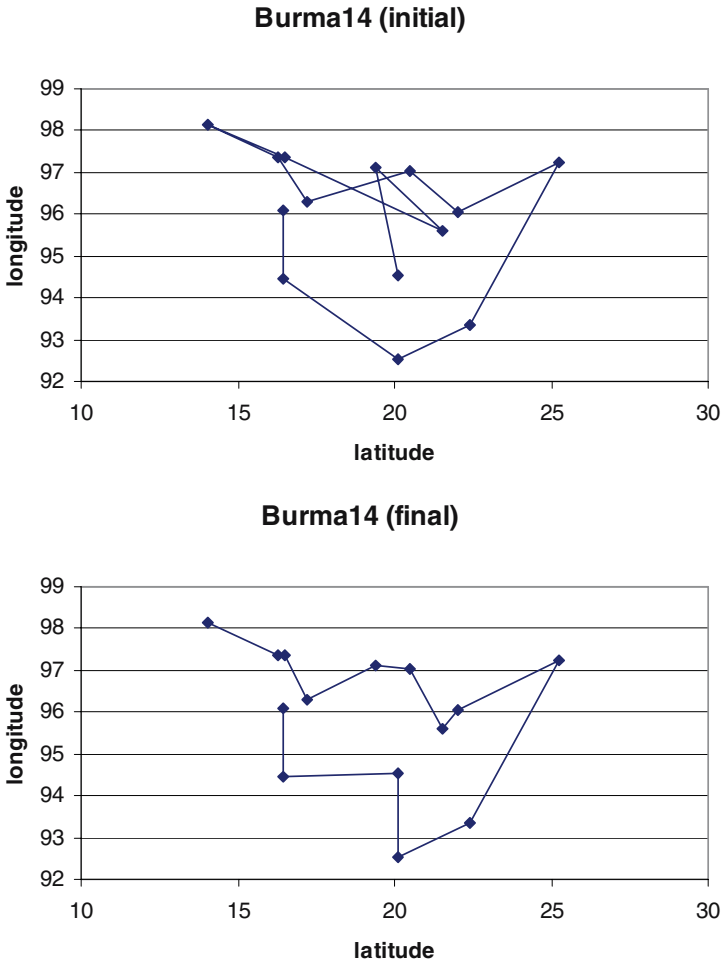


Fig. 14. Travelling Salesman Problem: (*top*) sub-optimum Burma14 tour; (*bottom*) optimum Burma14 tour

cases is [16.47, 96.10], but the initial (non-optimal) tour finishes at city#14 [20.09, 94.55], whereas the optimal tour finishes at city#10 [14.05, 98.12]. Likewise, Fig. 15 shows two tours of the larger Berlin52 city data set, both sub-optimal (left) and optimal (right).

Dedicated websites exist which compare various optimization algorithms and also serve as a TSP data repository.¹⁴

¹⁴ See for example, <http://www.research.att.com/~dsj/chtsp/>

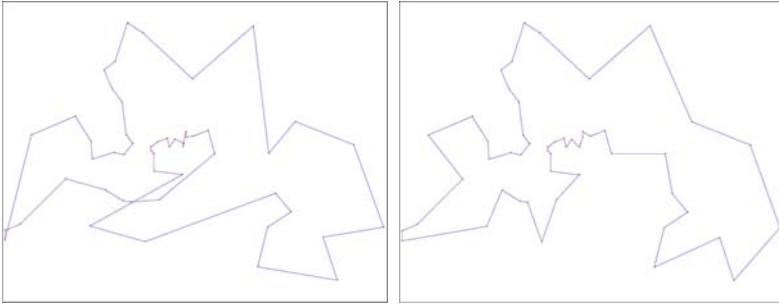


Fig. 15. Travelling Salesman Problem: (*left*) sub-optimum tour; (*right*) optimum Berlin52 tour

Hendtlass describes the use of both Particle Swarm Optimization (PSO) and Ant Colony Optimization to solve TSP, by way of the following algorithm (ACO) [127]:

Algorithm 1 Ant Colony Optimization TSP Algorithm (after [127])

1. initialize pheromone on each path segment, and randomly distribute N ants among C cities.

repeat

2. each ant decides which city to move to next (provided they have not previously visited it)

until they return to their original city

3. each ant calculates the length of its tour, then updates its information about the shortest tour found to date.

4. the pheromone levels on each path segment are refreshed.

5. all ants that have completed their assigned maximum number of tours (typically one) die and are replaced by new ants at randomly chosen cities.

6. return to 2. and continue until some termination criterion is met (for example, best path length < threshold, or maximum number of tours reached – similar to BP training).

Sharkey and Sharkey show how swarms are well suited to the study of collective robotic behavior – via so-called ‘swarm robotics’ [269].

9 Immunity-Based and Membrane-Based Computing

9.1 Immunity-Based Computing

Artificial Life (Alife) was a term first coined by Langton [176]. It has subsequently served as the inspiration not only for a lot of research effort in evolutionary computing (EC), and most especially swarms, but also more

recently with Immunity-based and Membrane-based Computing. Moreover, ‘Nature-Inspired Computing’ (NIC) emphasizes self-organization and complex systems principles [191]. What all these computing paradigms share in common is an approach to computing inspired by (based upon) processes observed in Nature. Immunity-Based Computing (IBC) Systems take as their inspiration the memory, learning and self-organization ability of biological immune systems (more specifically, those of invertebrates) [87, 141]. The ability of ‘antibodies’ to discriminate between *self* and *non-self*, and to self-repair in the face of bodily infections are implemented in Artificial Immune Systems (AIS) by way of computer simulations (in which the attributes of ‘antigens’ and ‘antibodies’ are encoded as strings within an appropriate data structure).

[23] pointed out the similarity between immune systems and intrusion detection systems, inasmuch as they both need to first identify then respond to malicious ‘agents’. An immunological selection mechanism was employed by [38] in their agent-based optimization of neural network architectures.

9.2 Membrane-Based Computing

The inspiration for membrane-based computing paradigm is, as its name suggests, biological membranes, which house multiple sets of objects within various compartments, and which evolve over time according to certain (non-deterministic) ‘reaction rules’ acting in parallel [27]. These objects are able to pass through membranes, and the membranes themselves are able to change shape, divide, dissolve, and/or change their permeability. Such characteristics (attributes) can be used in turn to define not only system state transitions, but also of state transition *sequences*, which together can serve as the basis for (stochastic) computations, optimizations and the like.

10 DNA Computing

Biologically-inspired CI methods – most especially ANNs and GAs – are usually realized in practice by way of software simulations on digital (silicon-based) computers, although we saw in Sect. 8.1 how success has also been achieved by way of *hardware* implementations (most commonly, via Field Programmable Gate Arrays). In other words, carbon-based techniques are simulated in silicon. The basic idea behind DNA computing is to do the exact reverse – namely to realize in ‘wetware’ algorithms derived from the world of ‘software’ (or ‘firmware’), although some researchers do in fact focus on implementation of DNA computing algorithms in silicon [2, 4]. Such an approach conjures up science fiction visions of the future, more specifically, rather than build computers, why not *grow* them (that is, in a test tube)?

DeoxyriboNucleic Acid (DNA) consists of four bases: A(denine), G(uanine), C(ytosine), T(hymine) – in addition, some constraints apply as to

which bases are allowed to connect to which others. In a DNA computer, data are represented using strands of DNA, next chemicals mixed in a test tube, then reactions allowed to take place (in parallel), and finally the results of these reactions extracted and interpreted. Such carbon-based or ‘wet’ computation (as opposed to conventional, silicon-based computing) has more in common with Quantum Computing (QC) [256,311], in that computations are inherently massively parallel, but where the encoding (input) and decoding (output) – in other words, system I/O – is far from straightforward. More specifically, how do we first encode problems of interest into DNA strand representation, and once the reactions (computations) have completed, how do we efficiently extract the results of these calculations into an intelligible form? Another problem that both DNA- and Quantum Computing share is the (lack of) *repeatability* of experiments, inasmuch as errors, being inherently non-deterministic, can have the cumulative effect of producing different outcomes on different experimental runs!

So what then is the basic appeal of DNA computing? In short it is the inherently (massive) *parallel* nature of the simultaneous chemical reactions taking place within the test tube, despite the reaction times of each being relatively slow (a situation akin to that of *slow* neurons within an artificial neural network nevertheless leading to *fast* overall network performance – Sect. 7).

11 Intelligent Agents

During the 1970s and 80s, procedural programming was the norm. This was largely superseded by Object-based and Object-Oriented Programming (OOP) during the 1980s and 90s. An alternative approach – agent-oriented programming [14,53,146] – began to emerge during the 1990s. This paradigm is founded on the concept of software ‘agents’, as opposed to software ‘objects’ (methods + data). Agents are touted as being better able to interact with complex, real-world situations than objects, since they are network-centric, adaptive and self-modifying (indeed, *self-repairing*) – unlike objects, which are fixed and unable to modify their behavior over time, being constrained to obey the Boolean logic rules which underpin them [19,183,197,210,317]. Now unlike Java applets say, which require a Java Virtual Machine to be executing on all hosts, mobile agents are free to move at will between the nodes (hosts) within a heterogeneous network. They only consume network bandwidth when moving between hosts, and they continue to execute once relocated on to the new host. This necessarily raises the issue of security and access permissions, lest a host regards the incoming agent as a virus, worm, or other malicious form of software, rather than a *bona fide* application [72]. In fact, some go so far as to regard intelligent agents as ‘both wrong and evil’ [177].

The autonomous software agent concept came out of work in distributed AI during the 1980s [89,315–317]. Fundamentally, an agent is an entity that

perceives its environment through sensors, then takes ‘appropriate’ actions through ‘effectors’ (actuators). Moreover, an agent is able to *automate* this mapping between perception and action(s) [261]. An ‘intelligent’ agent is one that takes the ‘best’ possible action in any given situation. Ideally, such actions should be ‘rational’ (although humans are often notoriously *irrational!*); ‘rationality’ in this context is taken to mean that an agent’s behavior at any point in time is a function of:

1. knowledge about its environment,
2. its repertoire of possible actions,
3. its ‘perceptual history’ (in other words, everything it has perceived through its senses to this point in time – which in turn could be used to modify/update/adapt its Knowledge Base), and
4. some performance measure (feedback) of the success or otherwise of its actions.

In the BDI model, intelligent agents are said to possess beliefs, desires and intentions [106]. In [275], BDI agents are developed to interact with humans in a multi-player game environment, using the JACK [6] toolkit.

From a CI perspective, intelligent agents are viewed as reactive, proactive, autonomous, social entities (the latter leading naturally to the concept of *Multi-Agent Systems* – MAS). In practice, they are software architectures and programs (methods) which perform the aforementioned ‘rational’ functions. Many different types of agents have been developed over the years, tailored to specific applications (such as ‘selfish’ versus ‘cooperative’). In order for intelligent agents to communicate their knowledge and conceptualizations with their fellow agents, they need to ‘speak a common language’, as it were. To this end, agent communication languages have been developed (for example, KQML [90]). In [9], coloured Petri Nets are used to coordinate agent interactions in open environments.

We also need to define an MAS ontology, by which we mean the formal representation of the knowledge pertaining to a particular domain, typically by specifying commonly used terms, together with explanations of how they interact with each other, in a computer-readable form (Sect. 2.1). Such ontologies can be either general or specific to a particular application domain. Accordingly, ontology generation is facilitated by formal languages such as OIL [88].

12 Hybrid Methods

Back in Sect. 1 we formulated a working definition of Computational Intelligence as ‘neural network, evolutionary and/or fuzzy techniques, and more especially *hybrids* or synergistic combinations/ensembles of these complementary approaches.’ It is the latter aspect which is the focus of the

present Section. The basic premise is that should a *single* CI method fail to deliver the desired performance, the perhaps a *mixture* may be able to.

In this regard, we must necessarily examine issues previously considered by researchers in data/sensor fusion [35, 60, 91, 117, 157], multi-modal user interfaces [102, 285] [as well as <http://research.microsoft.com/mmui> and <http://almaden.ibm.com/u/turaman/chi-2003/mmi-position.html>],¹⁵ biometric identification systems [170], and on a more pragmatic level, classifier combinations [111, 156, 169], neural network ensembles [118, 120, 243, 268, 298, 331], and indeed parallel processing in the most general sense [70, 115]. For instance, some practitioners have been guilty in the past of inappropriately ‘throwing’ a parallel computer at problems of interest, in the vain hope of rendering a solution more tractable and/or improving performance. In doing so, they often lose sight of Amdahl’s Law:

$$\text{Speedup} = S + P/n \quad (8)$$

where S is that part of the problem of interest that must be executed *sequentially*, P is that (often small) part which can be solved concurrently, and n is the number of computers (processors) available in the parallel system.

To take a roadwork analogy, there is no point in having 10 workmen armed with shovels (‘processors’) if only *one* shovel can fit in a hole in the ground at any one time, with the other 9 workmen standing by idle. In other words, this is an inherently *sequential* task.

Nevertheless, much effort has been placed in recent times into the development of *hybrid* CI systems [307]. The naïve hope is that by incorporating several CI techniques into a hybrid solution, we will improve system performance. Obviously this is not necessarily the case – it really depends on whether the respective techniques are complementary and/or enhance the performance of each in isolation.

There are several challenges in developing hybrid systems, in particular (i) selecting the most appropriate technique(s), depending on the attributes pertaining to (characteristics exhibited by) the problem under study, and (ii) how best to combine these techniques [26, 219, 227]. Ho characterizes the latter thus: “Instead of looking for the best set of features and the best classifier, now we look for the best set of classifiers and then the best combination method.” [134]. Before we rush off to develop *new* methods though, Kuncheva counsels us “to make the best use of the tools and methodologies that we have at present, before setting off for new complicated designs.” [169].

[318] combined multiple classifiers for handwriting recognition, while Miller and Yan did so in the context of signal processing [208], [46] for speaker identification, and [108] for intruder detection in computer networks.

¹⁵ See also *J. Multimodal Interfaces* (Springer).

Table 2. CI technique comparison (after [219])

CI technique	Learning	Explanation	Adaptation	Discovery	Flexibility
ANN	excellent	poor	excellent	fair	excellent
EC	excellent	fair	good	excellent	good
Fuzzy	poor	fair	poor	poor	excellent

Kuncheva identifies the following four approaches to building classifier ensembles: (a) at the data level, (b) at the feature level, (c) at the classifier level, and (d) at the combination level.¹⁶ Furthermore, there are two main strategies that can be employed for combining classifiers, these being *fusion* and *selection* [169]. With the former approach, each classifier has knowledge about the *entire* feature space, whereas with the latter each classifier has knowledge about (and responsibility for) part of the feature space only.

There are also approaches which fall part way between these two, namely ‘fusion-selection’, also known as the ‘ensemble-modular’ approach [268], or ‘multiple-hybrid’ topology [172] – a typical example is the so-called ‘Mixture-of-Experts’ [144, 149, 224].

Example *fusion* techniques include majority voting [11, 168, 174], plurality voting [66, 189], naïve Bayesian [284], bagging (or ‘bootstrap sampling’) [25, 28, 69, 252], boosting – the combination of rough, inaccurate ‘rules-of-thumb’ to produce accurate predictors – [74–76, 99], and fuzzy integral [50, 166, 299]. In [49] fuzzy logic provided the fusion; in [167], it was Genetic Algorithms.

In the context of hybrid CI systems, [219] attempt to characterize neural, evolutionary and fuzzy (‘intelligent system’) techniques, as well as Knowledge-based Expert Systems, along dimensions of learning, explanation capability, adaptation (in response to changes in the environment), knowledge discovery, and flexibility (that is, decision making ability in the face of imprecise, incomplete and/or new input data) – Table 2.

We leave it as an exercise for the interested reader to expand Table 2 to characterize intelligent agents, swarms, immunity-based systems, and the many other ancillary CI methods mentioned earlier in this Chapter (refer to Sect. 1 in particular).

Michalewicz and Fogel caution against attempting to add *too many* components (elements, ‘ingredients’) into the mix (‘stew’), lest we overload the hybrid system and the techniques begin interfering with each other, leading to degraded overall system performance [207]. Nevertheless, it is behoven upon us to cite here some studies which *have* managed to produce hybrid systems which exhibit superior performance over that of stand-alone CI methods. One

¹⁶ See also the *Annual Intl. Workshops on Multiple Classifier Systems (MCS)*, sponsored by the International Association for Pattern Recognition.

such hybrid approach involves the combination of ANNs and Fuzzy Logic, another the combination of ANNs and GAs. We briefly describe a couple of representative examples below.

Fuzzy Expert Systems

[294] observed that if the knowledge at our disposal can be expressed in the form of linguistic rules, then we can readily construct a Fuzzy Inference System (FIS). More specifically, we need to specify fuzzy set membership, the fuzzy operators and the Knowledge Base.

The Fuzzy ES of [80] utilizes user-defined triangular and/or non-linear membership functions. The three input parameters to the system are: (a) how the antecedents are handled, (b) how output membership values are formed, and (c) how defuzzification is performed.

Now rather than fine tune the fuzzy membership functions manually, we could alternatively *learn* these using an ANN.

NeuroFuzzy

Ordinarily, the standard MAX and MIN operators used in Fuzzy Systems are unable to be differentiated, as in the BP algorithm. In order to link the nodes in an MLP with Fuzzy Logic rules, one approach is to use a Fuzzy Associative Memory (FAM) [319]. More specifically, a Fuzzy Associative Memory (FAM) incorporates both a fuzzy logic rule and an associated (adaptable) weight. Von Altrock shows how a Fuzzy Inference System (FIS) can be mapped to an MLP, and where it is possible to use a modified form of BP – more specifically, with input fuzzification mapped to the input layer, the Inference Engine to the hidden layer(s), and the output de-fuzzification to the output layer [301].

In NeuroFuzzy systems, the Knowledge Base (Fuzzy Inference System – FIS) at the bottom of Fig. 5 is replaced by a neural network in which the rules are encoded within the network weights. Fuzzy rule evaluation then amounts to determining the output pattern (response) which most closely matches the input pattern (query). To put it another way, the ANN learning algorithm is used to determine the parameters [294].

The Adaptive-Network Fuzzy Inference System (ANFIS) of Fig. 16 [145] implements a 2-input Sugeno model with 9 inputs. There are three fuzzy sets to which the input linguistic variables (x , y) can belong. Input fuzzification forms the **if** (premise) parameters, which in turn fire certain fuzzy rules, which drive one or more of the **then** (consequent) variables, which are then finally converted back to linguistic form during output de-fuzzification.

There have been numerous Neuro-Fuzzy hybrid systems cited in the literature, including those of [13, 36, 42, 48, 103, 153, 184, 187, 212–215, 230, 233, 267, 288, 295].

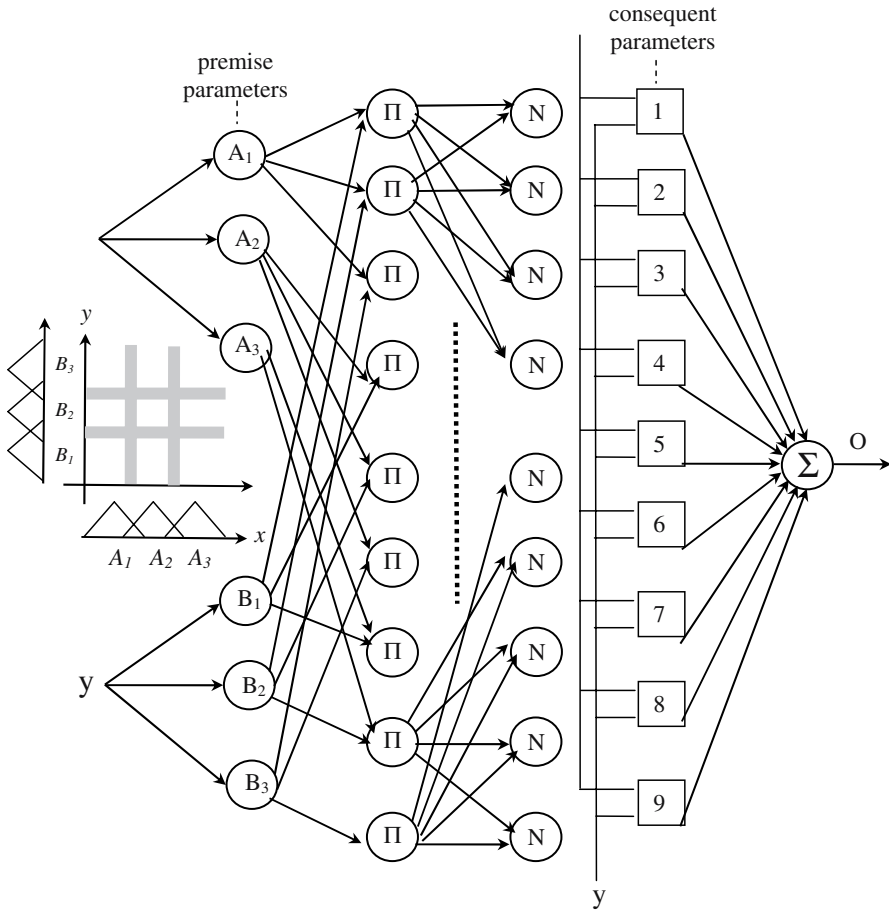


Fig. 16. ANFIS (after [145]: 339) – ©1997, reprinted by permission of Pearson Educational Inc., Upper Saddle River, NJ

Fuzzy Neuro

The basic approach here is to incorporate Fuzzy techniques in the adaptation of network weights, in order to improve the performance of ANNs. Alternatively, Enbutsu has applied fuzzy rule extraction to a trained MLP [81].

In Fuzzy BackPropagation, heuristics based on first- (Change-of-Error) and second- (change of CE) derivative weight changes can be used to develop a Fuzzy Rule Base for both learning rate (η) and momentum (α). Using such an approach, some researchers have found both a speedup of network convergence and smaller resulting mean square error (MSE) [124].

In FuzzyART, stable recognition categories self-organize in response to arbitrary sequences of either binary (ART1 [40]) or analog (ART2 [41]) input

patterns. The Boolean AND (intersection) operator is replaced with the Fuzzy MIN operator inside a ‘fuzzy cube’ [162]. FuzzyARTMAP is a supervised extension comprising two FuzzyART networks, in which a MIN-MAX learning rule controls category structure.

Fuzzy min-max NNs [80, 273, 274] use unsupervised pattern clustering, realized by way of hypercube fuzzy sets. The hypercube contains all patterns, and moreover defines a membership region in n -dimensional pattern space. The MIN-MAX points, together with the hyperplane membership functions define a fuzzy set (cluster). Learning in a Fuzzy Min-Max (Cluster) neural network corresponds to the creation and adaptation of hypercubes in pattern space – in other words, a form of *un*-supervised clustering [80].

Other Fuzzy-Neuro systems are described in [39, 238, 327].

Evolution of Neurons/ANNs

Now rather than *train* ANNs, we could alternately *evolve* them. While this is possible, it is often computationally prohibitive. What is more feasible however is to evolve the network architecture, rather than the weights, which was the approach taken by [265] with his use of ‘blueprints’ (which describe the number of nodes, starting with the input layer, together with their fan-in); crossover occurs at common points in such blueprint representations. Other researchers investigating the evolution of ANNs include [203, 232]. Cho combined ANNs using GAs [52]. Chapter 20 of this Compendium examines this topic in considerably more detail.

GAs and Fuzzy

We saw earlier that rather than fine tune fuzzy membership functions manually, we could alternatively *learn* these using an ANN; we could just as readily evolve them using a GA/EA [152]. [80] describes just such a Fuzzy ES in which the rule set is evolved using a GA. Conversely, the basic GA crossover and mutation operations can be determined by reference to a fuzzy rule base (lookup table). [179] take this approach a step further in their development of an ontology-based genetic fuzzy agent.

There has been considerable activity in GA-Fuzzy hybrid systems, including [18, 56, 86, 104, 128, 180, 181, 220, 221, 249, 272, 303].

Swarms and Fuzzy

[155] combined Particle Swarm Optimization and the Taguchi method for identifying optimum fuzzy models in the control of a rapid Ni-Cd battery charger. By contrast, [43] combined swarms and k -nearest neighbours.

Other Hybrid Approaches

In [280], fuzzy multivariate auto-regression is used to model multivariate time series data, in particular interest rates and gas furnace measurements. [225] combined *three* CI methods in their immunity-based, multi-agent ANNs.

13 Conclusion

We made the observation in the Introductory Section that CI encompasses more than just intelligent agents – indeed, most researchers nowadays agree on the core technologies of neural network, evolutionary and fuzzy logic. Duch characterizes CI as incorporating “all non-algorithmic processes that humans (and sometimes animals) can solve with various degrees of competence.” [78]. Will we see CI deliver where AI has failed during the past five decades? Some researchers, including Duch, hold the view that the early activities of AI were somewhat misguided – indeed, that the problems themselves were ill-formed. For instance, Ford and Hayes argue that “the traditional view of the goal of AI – create a machine that can successfully imitate human behavior – is wrong.” [95]. Moreover, the claim that ‘all intelligence comes from symbol manipulation’ [222,223] has been largely misinterpreted. Despite this, McCarthy argues that one way out of this (self-inflicted?) mire could be *more* formalism [202].

Brooks postulates the following as being possible explanations to the rhetorical question ‘What is going wrong?’ (namely with AI):

1. wrong parameter models,
2. working below some complexity threshold,
3. lack of computing power, and/or
4. we’re missing something fundamental and unimagined.

We had much to say earlier in this Chapter regarding model-driven *versus* data-driven approaches (with CI belonging to the latter camp). Postulates 2 and 4 above would therefore appear to be most plausible.

According to Pollack, AI has stalled because of its preoccupation with simulating the human mind and/or mimicking human intelligence [246], coupled with a fixation on symbolic [202,222,223], rather than sub-symbolic processing. Indeed, he makes the pertinent observation that Moore’s Law of *itself* (in other words, raw computational power) should have been able to deliver us with ‘real AI’ by now; likewise [29–32] pointed out that ‘massive parallelism adds absolutely zero in terms of expressivity’, and by the way, casts doubt on Brooks’ third postulate above. Instead, AI applications suffer from the prevailing curse afflicting software generally nowadays, namely that of ‘software bloat’ – by which we mean an increase in software *quantity* (size of programs) quite unmatched by software *quality*! Pollack further observes that:

“...many intelligent processes in Nature perform more powerfully than human symbolic reasoning, even though they lack any of the mind-like mechanisms long believed necessary for human ‘competence’.” [246]

Instead, he advocates the study of what he terms ‘mindless intelligence’, citing the success of TD-Gammon [290] over both rule-based systems and human backgammon players as being indicative of what can be achieved by following such an approach. This is reminiscent of Brook’s subsumption architecture [29,31,32]. Another potential advantage of mimicking such simple organisms – even ones without a nervous system – is that we may also be able to produce *artificial* systems which are likewise capable of self-repair.

As Hawkins rightly observes, it is clear the brain works in a very different manner to digital computers [123]. In order to build intelligent machines we therefore need to first understand how brains work, then attempt to replicate them. In his case he has focused on the neocortex, which uses time and hierarchy to create and perceive world models. He subsequently developed the ‘Hierarchical Temporal Memory (HTM)’, which is said to learn in much the same manner as children do – more specifically by exposure to sensory data.¹⁷ Will this approach prove to be a more fruitful path to an ‘intelligent machine’? More to the point, is this old chestnut an *appropriate* pursuit for CI, or should we relegate this to the dustbin of history?

The present author advocates the use of CI techniques simply for their own sake, and not as justification to continue tilting at windmills *à la* Don Quixote. Should this lofty goal eventuate then all well and good; in the meantime, let’s simply enjoy CI techniques for their own inherent interest (and beauty?). Let Nature continue to inspire us now and into the future!

Acknowledgements

The author gratefully acknowledges the helpful feedback on earlier chapter drafts from Professor Witold Pedrycz, Professor Tim Hendtlass (likewise for the TSP figures), Associate Professor Russell Standish, Dr. Christine Mumford, and especially the ‘think tanks’ undertaken with Professor Yoshi Ishida whilst on sabbatical during October 2007 (on CI definitions and hybrid systems). The financial support of the Intelligent Systems Research Centre at the University of Wollongong is also greatly appreciated. I would also like to thank the Faculty of Informatics and School of Computer Science and Software Engineering at the University of Wollongong for allowing me to take a six-month sabbatical during Semester-2 of 2007 in order to collaborate with contributing authors and to bring this Compendium to fruition.

¹⁷ <http://www.www.numanta.com>

References

1. Abe S (2005) *Support Vector Machines for Pattern Classification*. Springer-Verlag, New York, NY.
2. Adelman L (1994) Computing with DNA. *Scientific American*, 279(2): 54–61.
3. Allen J (1998) AI growing up: the changes and opportunities. *AI Magazine*, Winter: 32–45.
4. Amos M (2005) *Theoretical and Experimental DNA Computation*. Springer-Verlag, Berlin.
5. Anderson JA, Rosenfeld E (eds.) (1988) *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA.
6. AOS (2002) JACK intelligent agents. *Agent Oriented Software P/L* (available online at <http://www.agent-software.com.au> – last accessed November 2006).
7. Arotaritei D, Negoita GM (2002) Optimisation of recurrent NN by GA with variable length genotype. In: McKay B, Slaney J (eds.) *AI2002: Advances in Artificial Intelligence*. Springer-Verlag, Berlin.
8. Bagley JD (1967) The behavior of adaptive systems which employ genetic and correlation algorithms. *PhD Thesis*, University of Michigan, Ann Arbor, MI.
9. Bai Q, Zhang M (2006) Coordinating agent interactions under open environments. In: Fulcher J (ed.) *Advances in Applied Artificial Intelligence*. Idea Group, Hershey, PA: 52–67.
10. Banzhaf W, Nordin P, Keller RE, Francone FD (1998) *Genetic Programming, An Introduction: On the Automatic Evolution of Computer Programs and its Application*. Morgan Kaufmann, San Francisco, CA.
11. Battiti R, Colla AM (1994) democracy in neural networks: voting schemes for classification. *Neural Networks*, 7: 691–707.
12. Beale R, Pryke A (2006) Knowledge through evolution. In: Fulcher J (ed.) *Advances in Applied Artificial Intelligence*. Idea Group, Hershey, PA: 234–250.
13. Benitez JM, Blanco A, Delgado M, Requena I (1996) Neural methods for obtaining fuzzy rules. *Mathware Soft Computing*, 3: 371–382
14. Bergenti F, Giezes M-P, Zambonelli F (2004) *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Springer-Verlag, Berlin.
15. Bezdek JC (1981) *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, NY.
16. Bezdek JC (1994) What is computational intelligence? In: Zurada JM, Marks II RJ, Robinson CJ (eds.) *Computational Intelligence Imitating Life*. IEEE Press, Piscataway, NJ: 1–12.
17. Bezdek JC (1998) Computational intelligence defined – by everyone! In: Kaynak O, Zadeh LA, Türksen B, Rudas IJ (eds.) *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*. Springer-Verlag, Berlin: 10–37.
18. Bezdek JC, Hathaway RJ (1994) Optimization of fuzzy clustering criteria using genetic algorithms. In: *Proc. World Congress Computational Intelligence (WCCI'94)*, June, Orlando, FL. IEEE Computer Society Press, Los Alamitos, CA: 589–594.
19. Bigus JP, Bigus J, Bigus J (2001) *Constructing Intelligent Agents Using Java (2nd ed)*. Wiley, New York, NY.
20. Black M (1937) Vagueness: an exercise in logical analysis. *Philosophy of Science*, 4: 427–455.

21. Bonabeau E, Dorigo M, Theaulaz G (1999) *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, UK.
22. Botros NM, Abdul-Aziz M (1994) Hardware implementation of an ANN using field programmable gate arrays (FPGAs). *IEEE Trans. Industrial Electronics*, 41(6): 665–667.
23. Boukerche A, Jucá KRL, Sobral JB, Notare MSMA (2004) An artificial immune based intrusion detection model for computer and telecommunication systems. *Parallel Computing*, 30(5–6): 629–646.
24. Breiman L, Friedman J, Olsh R, Stone CJ (1984) *Classification and Regression Trees*. Chapman and Hall, New York, NY.
25. Breiman L (1996) Bagging predictors. *Machine Learning*, 26(2): 123–140.
26. Breiman L (1999) Combining predictors. In: Sharkey AJC (ed.) *Combining Artificial Neural Networks: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, Berlin: 31–50.
27. Brewka G (1997) *Principles of Knowledge Representation*. CSLI Publications, Stanford, CA.
28. Brill R, Guiterrez-Osuna, Quek F (2003) Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern recognition*, 36(6): 1291–1302.
29. Brooks RA (1986) A robot layered control system for a mobile robot. *IEEE J. Robotics and Automation*, RA-2: 14–23.
30. Brooks RA (1991) Intelligence without representation. *Artificial Intelligence*, 47(1–3): 139–159.
31. Brooks RA (1991) Intelligence without reason. In: *Proc. 12th Intl. Joint. Conf. Artificial Intelligence – IJCAI*. August, Sydney, Australia: 569–595.
32. Brooks RA (1991) How to build complete creatures rather than isolated cognitive simulators. In: van Lehn K (ed.) *Architectures for Intelligence*. Lawrence Erlbaum Associates, Hillsdale, NJ: 225–239.
33. Brooks RA, Kurzweil R, Gelernter D (2006) Gelernter, Kurzweil debate machine consciousness. (available online at <http://www.kurzweilai.net/meme/frame.html?m=4> – last accessed April 2007).
34. Brooks RA (2007) The relationship between matter and life. *Nature*, 409(6818): 409–410.
35. Brooks RR, Ivengar SS (1997) *Multi-Sensor Fusion: Fundamentals and Applications with Software*. Prentice Hall, Upper Saddle River, NJ.
36. Brown M, Harris CJ (1994) *Neuro-fuzzy Adaptive Modeling and Control*. Prentice Hall, Englewood Cliffs, NJ.
37. Bryson AE, Ho Y-C (1969) *Applied Optimal Control*. Blaisdell, New York, NY.
38. Byrski A, Kisiel-Dorohinicki M (2005) Immune-based optimization of predicting neural networks. In: Sunderam VS et al. (eds.) *Proc. Workshop Intelligent Agents in Computing Systems – ICCS 2005*, 22–25 May, Atlanta, GA, Lecture Notes in Computer Science 3516. Springer-Verlag, Berlin.
39. Calado JMF, Ss da Costa JMG (1999) An expert system coupled with a hierarchical structure of fuzzy neural networks for fault diagnosis. *J. Applied Mathematics and Computer Science*, 3(9): 667–688.
40. Carpenter GA, Grossberg SA (1987) A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Understanding*, 37: 54–115.
41. Carpenter GA, Grossberg SA (1987) ART2: self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23): 4919–4930.

42. Castellano G, Castiello C, Fanelli AM, Mencar C (2003) Discovery prediction rules by a neuro-fuzzy modeling framework. In: Palade V, Howlett JR, Jain LC (eds.) *Knowledge-Based Intelligent Information and Engineering Systems*. Springer-Verlag, Berlin, 2: 1243–1248.
43. Cedeño W, Agrafiotis DK (2003) Combining particle swarms and k-nearest neighbors for the development of qualitative structure-activity relationships. *Bicom Magazine*: 43–53.
44. Chalmers DJ (1997) Moving forward on the problem of consciousness. *Consciousness Studies*, 4(1): 3–46.
45. Chalmers DJ (1998) *On the Search for the Neural Correlate of Consciousness*. MIT Press, Cambridge, MA.
46. Chen K, Wang L, Chi H (1997) Methods of combining multiple classifiers with different features and their application to text-independent speaker identification. *Intl. J. Pattern Recognition and Artificial Intelligence*, 11(3): 417–445.
47. Chen Z (2000) *Computational Intelligence for Decision Support*. CRC Press, Boca Raton, FL.
48. Chimmanee S, Wipusitwarakun K, Runggeratigul S (2003) Adaptive per-application load balancing with neuron-fuzzy to support quality of service over IP in the internet. In: Palade V, Howlett JR, Jain LC (eds.) *Knowledge-Based Intelligent Information and Engineering Systems*. Springer-Verlag, Berlin, I: 533–541.
49. Cho S-B, Kim JH (1995) Pattern recognition with neural networks combined by genetic algorithm. *Fuzzy Sets and Systems*, 103: 339–347.
50. Cho S-B, Kim JH (1995) Combining multiple neural networks by fuzzy integral and robust classification. *IEEE Trans. Systems, Man, and Cybernetics*, 25: 380–384.
51. Cho S-B, Kim JH (1995) Multiple network fusion using fuzzy logic. *IEEE Trans. Neural Networks*, 6: 497–501.
52. Cho S-B (1999) Pattern recognition with neural networks combined by genetic algorithm. *Fuzzy Sets and Systems*, 103: 339–347.
53. Ciancarini P, Wooldridge MJ (eds.) (2000) Agent-oriented software engineering. In: *Proc. 1st Intl. AOSE Workshop*, June, Limerick, Ireland, Lecture Notes in Computer Science 1957, Springer-Verlag, Berlin.
54. Cohon JL (2004) *Multiobjective Programming and Planning*. Dover Publications, Mineola, NY.
55. Conrad M (1989) The brain-machine disanalogy. *Biosystems*, 22(3): 197–213.
56. Cordon O, Herrera F, Lozano P (1997) On the combination of fuzzy logic and evolutionary computation: a short review and bibliography. In: Pedrycz W (ed.) *Fuzzy Evolutionary Computation*. Kluwer Academic Publishers, Boston, MA: 41–42.
57. Cox E (1994) *The Fuzzy System Handbook*. AP Professional Books, Boston, MA.
58. Crox T (2007) Stop cahsing the AI illusion. *Communications ACM*, 50(4): 7–8.
59. Cybenko G (1989) Approximation by superposition of a sigmoidal function. *Math Control, Signals, Systems*, 2: 303–314.
60. Dasarthy BV (1997) Sensor fusion potential exploitation – innovative architectures and illustrative applications. *Proc. IEEE*, 85: 24–38.

61. Dasgupta D, Attou-Okine N (1997) Immunity-based systems: a survey. In: *Proc. IEEE Intl. Conf. Systems, Man and Cybernetics*. Orlando, FL. IEEE Computer Society Press, Los Alamitos, CA: 326–331.
62. Davis L (ed.) (1991) *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY.
63. Deasy H (2007) Consciousness and computers. *IEEE Computer*, 40(10): 7.
64. DeJong KA (1975) An analysis of the behavior of a class of genetic adaptive systems. *PhD Thesis*, University of Michigan, Ann Arbor, MI.
65. DeJong KA (2006) *Evolutionary Computation: A Unified Approach*. Bradford/MIT Press, Cambridge, MA.
66. Demirekler M, Altincay H (2002) Plurality voting-based multiple classifier systems: statistically independent with respect to dependent classifier sets. *Pattern Recognition*, 35: 2363–2379.
67. Dennett D (1991) *Consciousness Explained*. Little, Brown and Co., Lebanon, IN.
68. Deschamps JP, Bioul GJA, Sutter GO (2006) *Synthesis of Arithmetic Circuits: FPGAs, ASIC and Embedded Systems*. Wiley, New York, NY.
69. Dietterich T (2000) An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 40(2): 139–157.
70. Dongarra J, Foster I, Fox GC, Gropp W, Kennedy K, Torczon L White A (2003) *The Sourcebook of Parallel Computing*. Morgan Kaufman, San Francisco, CA.
71. Dorigo M, Stützle T (2004) *Ant Colony Optimization*. MIT Press, Cambridge, MA.
72. Dowling C (2000) Intelligent agents: some ethical issues and dilemmas. In: *Proc. Australian Institute Conf. Computer Ethics – AICE2000*, Canberra, 11–12 November, Australian Computer Society, Darlinghurst, NSW: 28–32.
73. Dreyfus H, Dreyfus S (1986) Why expert systems do not exhibit expertise. *IEEE Expert*, 1(2): 86–90.
74. Drucker H, Schapire RE, Simard P (1992) Improving performance in neural networks using a boosting algorithm. In: Hanson SJ et al. *Advances in Neural Information Processing Systems 5*, 30 November–3 December, Morgan Kaufman, San Mateo, CA: 42–49.
75. Drucker H, Cortes C, Jackel LD, LeCun Y, Vapnik V (1994) Boosting and other ensemble methods. *Neural Computation*, 6: 1289–1301.
76. Drucker H (1999) Boosting using neural networks. In: Sharkey AJC (ed.) *Combining Artificial Neural Networks: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, Berlin.
77. Dubois D, Prade H (1980) *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, NY.
78. Duch W (2007) What is computational intelligence and where is it going? In: Duch W, Mandziuk J (eds.) *Challenges for Computational Intelligence*. Springer-Verlag, Berlin.
79. Durkin J (1994) *Expert Systems: Design and Development*. Macmillan, New York, NY.
80. Eberhart R, Simpson P, Dobbins R (1996) *Computational Intelligence PC Tools*. Academic Press, Boston, MA.

81. Enbutsu I, Baba K, Hara N (1991) Fuzzy rule extraction from a multilayered network. In: *Proc. Intl. Joint Conf. Neural Networks (IJCNN'91)*, 8–12 July, Seattle, WA. IEEE Computer Society Press, Los Alamitos, CA: 461–465.
82. Engelbrecht AP (2003) *Computational Intelligence: An Introduction*. Wiley, New York, NY.
83. Engelbrecht AP (2005) *Fundamentals of Computational Swarm Intelligence*. Wiley, New York, NY.
84. Fahlman SE, Lebiere C (1990) The cascade learning architecture. In: Touretzky DS (ed.) *Advances in Neural Information Processing Systems*. Morgan Kaufmann, San Mateo, CA: 524–532.
85. Falconer K (2003) *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, New York, NY.
86. Fagarasan F, Negoita GM (1995) A genetic-based method for learning the parameter of a fuzzy inference system. In: Kasabov N, Coghill G (eds.) *Artificial Neural Networks and Expert Systems*. IEEE Computer Society Press, Los Alamitos, CA: 223–226.
87. Farmer JD, Packard NH, Perelson AS (1986) The immune systems: adaptation and machine learning. *Physica A*, 22: 187–204.
88. Fensel D, Hermalen F, Horrocks I, McGuinness D, Patel-Schneider P (2001) OIL: an ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2): 38–45.
89. Ferber J (1999) *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley, Reading, MA.
90. Finin T, Labrou Y, Mayfield J (1997) KQML as an agent communication language. In: Bradshaw JM (ed.) *Software Agents* MIT Press, Cambridge, MA.
91. Fisher R, Fulcher J (1998) Improving the inversion of ionograms by combining neural network and data fusion techniques. *Neural Computing and Applications*, 7: 3–16.
92. Fogel LJ, Owens AJ, Walsh MJ (1966) *Artificial Intelligence through Simulated Evolution*. Wiley, New York, NY
93. Fogel D (1995) Review of *CI: Imitating Life*, In: *IEEE Trans. Neural Networks*, 6: 1562–1565.
94. Fogel LJ (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ.
95. Ford K, Hayes P (1998) On computational wings: rethinking the goals of AI. *Scientific American*, 9/4: 78–84.
96. Fraser AS (1957) Simulation of genetic systems by automatic digital computers. *Australian J. Biological Science*, 10: 484–499.
97. Fraser AS (1960) Simulation of genetic systems by automatic digital computers. In: Kempthorne O (ed.) *Biometrical Genetics*. Macmillan, New York, NY: 70–83.
98. Freitas AA (2002) *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, Berlin.
99. Freund Y, ScahapiRE RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *J. Computer and System Sciences*, 55(1): 119–139.
100. Friedberg RM (1958) A learning machine: part I. *IBM J. Research and Development*, 2: 2–13.
101. Friedberg RM, Dunham B, North JH (1959) A learning machine: Part II. *IBM J. Research and Development*, 3: 282–287.

102. Fulcher J (2008) User interfaces. In: Pagani M (ed.) *Encyclopedia of Multimedia Technology (2nd ed)*. Information Sciences Reference, Hershey, PA (in press).
103. Fuller R (1999) *Introduction to Neuro-Fuzzy Systems*. Springer-Verlag, Berlin.
104. Furuhashi T, Nakaoka K, Uchikawa Y (1994) A new approach to genetic based machine learning and an efficient finding of fuzzy rules: proposal of Nagoya approach In: *Proc. IEEE/Nagoya University World Wisepersons Workshop on Advances in Fuzzy Logic, Neural Networks, and Genetic Algorithms*, Lecture Notes in Computer Science 1011, Springer-Verlag, Berlin: 173–189.
105. Gallagher JC, Virraham S, Kramer G (2004) A family of compact genetic algorithms for intrinsic evolvable hardware. *IEEE Trans. Evolutionary Computation*, 8(2): 111–126.
106. Georgeff M, et al. (1999) The belief-desire-intention model of agents. In: Müller JP, Singh MP, Rao AS (eds.) *Proc. 5th Intl. Workshop Intelligent Agents V, Agent Theories, Architectures, and Languages (ATAL-98)*. Lecture Notes in Computer Science 1555. Springer-Verlag, Berlin: 1–10.
107. Georgeff M, Azarmi N (2003) What has AI done for us? *BT Technology J.*, 21(4): 15–22.
108. Giacinto G, Roli F, Didaci L (2003) Fusion of multiple classifier for intrusion detection in computer networks. *Pattern Recognition Letters*, 24: 1795–1803.
109. Giarratano JC, Riley G (2005) *Expert Systems: Principles and Programming (4th ed)*. Thomson, Boston, MA.
110. Girau B (2000) FPNA: interaction between FPGA and neural computation. *Intl. J. Neural Systems*, 10(3): 243–259.
111. Ghosh J (2002) Multiclassifier systems: back to the future. In: Roli F, Kittler J (eds.) *Proc. 3rd Intl. Workshop Multiple Classifier Systems (MCS'02)*, Cagliari, Italy. Lecture Notes in Computer Science 2364, Springer-Verlag, Berlin: 1–15.
112. Gladwell M (2005) *Blink: The Power of Thinking without Thinking*. Little, Brown and Co., Lebanon, IN.
113. Gokhale MB, Graham PS (2005) *Reconfigurable Computing: Computation with Field-Programmable Gate Arrays*. Springer-Verlag, Berlin.
114. Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA.
115. Grama A, Karypis G, Kumar V, Gupta A (2003) *An Introduction to Parallel Computing: Design and Analysis of Algorithms (2nd ed)*. Addison Wesley, Reading, MA.
116. Greffenstette JJ (1984) A user's guide to GENESIS. *Technical Report CS-84-11*, Department of Computer Science, Vanderbilt University, Nashville, TN.
117. Hall DL, Llinas J (1997) An introduction to multisensor data fusion. *Proc. IEEE*, 85(1): 6–23.
118. Hansen LK, Salamon P (1990) Neural network ensembles. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(10): 993–1001.
119. Hart WE, Krasnogor N, Smith JE (eds.) (2005) *Recent Advances in Memetic Algorithms*. Springer-Verlag, Berlin.
120. Hashem S (1997) Optimal linear combinations of neural networks. *Neural Networks*, 10(4): 599–614.
121. Hassibi B, Stork DG, Wolff GJ (1992) Optimal brain surgeon and general network pruning. In: *Proc. IEEE Intl. Joint Conf. Neural Networks I*, San Francisco, CA. IEEE Computer Society Press, Piscataway, NJ: 293–299.
122. Haupt RL, Haupt SE (2004) *Practical Genetic Algorithms*. Wiley, New York, NY.

123. Hawkins J (2007) Learn like a human: why can't a computer be more like a brain? *IEEE Spectrum*, 44(4): 17–22.
124. Haykin SY (1999) *Neural Networks: a Comprehensive Foundation (2nd ed)*. Prentice Hall, Englewood Cliffs, NJ
125. Hearst M, Hirsh H (2000) AIs greatest trends and controversies. *IEEE Intelligent Systems*, January/February: 8–17.
126. Hebb DO (1949) *The Organization of Behavior*. Wiley, New York, NY
127. Hendtlass T (2004) An introduction to collective intelligence. In: Fulcher J, Jain LC (eds.) *Applied Intelligent Systems: New Directions*. Springer-Verlag, Berlin: 133–178.
128. Herrera F, Lozano M, Verdegay IL (1993) Tuning fuzzy logic controllers by genetic algorithms. *Technical Report DECSai-93102*, June, Universidad de Granada, Spain.
129. Higuchi T, et al. (1999) Real-world applications of analog and digital evolvable hardware. *IEEE Trans. Evolutionary Computation*, 3(3): 220–235.
130. Hinchey MG, Sterritt R, Rouff C (2007) Swarms and swarm intelligence. *IEEE Computer*, 40(4): 111–113.
131. Hinton GE, Anderson JA (1981) *Parallel Models of Associative Memory*. Lawrence Erlbaum Associates, Potomac, MD.
132. Hirai Y (1993) Hardware implementations of neural networks in Japan. *Neurocomputing*, 5: 3–16.
133. Hirota K (1995) History of Industrial Applications of Fuzzy Logic in Japan. In: Yen J, Langari R, Zadeh L (eds.) *Industrial Applications of Fuzzy Logic and Intelligent Systems*. IEEE Press, Piscataway, NJ: Chapter 2.
134. Ho TK (2002) Multiple classifier combination: lessons and the next steps. In: Kandel A, Bunke H (eds.) *Hybrid Methods in Pattern Recognition*. World Scientific, Singapore: 171–198.
135. Holland JH (1962) Outline for a logical theory of adaptive systems. *J. ACM*, 3: 297–314.
136. Holland JJ (1992) *Adaptation in Natural and Artificial Systems (2nd ed)*. MIT Press, Cambridge, MA.
137. Hopfield JJ (1984) Neurons with graded response have collective computational properties like those in two-state neurons. *Proc. National Academy Science*, 81: 3088–3092.
138. Hornik K (1991) Approximation capabilities of multi-layer feed-forward networks. *Neural Networks*, 4: 2151–2157.
139. Ignizio J (1991) *Introduction to Expert Systems*. McGraw-Hill, New York, NY.
140. Inuiguchi M, Hirano S, Tsumoto S (eds.) (2003) *Rough Set Theory and Granular Computing*. Springer-Verlag, Berlin.
141. Ishida Y (2004) *Immunity-Based Systems: A Design Perspective*. Springer-Verlag, Berlin.
142. Jackson P (1999) *Introduction to Expert Systems (3rd ed)*. Addison Wesley, Reading, MA.
143. Jacobs RA (1988) Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1: 295–307.
144. Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE (1991) Adaptive mixture of local experts. *Neural Computation*, 3: 79–87.
145. Jang J-S R, Sun C-T, Mizutani E (1997) *Neuro-Fuzzy and Soft Computing: a Computational Approach to Learning and Machine Intelligence*. Prentice Hall, Englewood Cliffs, Upper Saddle River, NJ.

146. Jennings NR, Faratin P, Norman TJ (2000) On agent-oriented software engineering. *Artificial Intelligence*, 117(2): 277–296.
147. Jensen FV (2001) *Bayesian Networks and Decision Graphs*. Springer-Verlag, Berlin.
148. Jin Y (ed.) (2006) *Multi-Objective Machine Learning*. Springer-Verlag, Berlin.
149. Jordan MI, Jacobs RA (1994) Hierarchical mixture of experts and the EM algorithm. *Neural Computation*, 6(2): 181–214.
150. Judd JS (1990) *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, MA.
151. Karplus W (1998) cited in: Kaynak O, Zadeh LA, Türksen B, Rudas IJ (eds.) *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*. Springer-Verlag, Berlin.
152. Karr C (1991) Applying genetics to fuzzy logic. *AI Expert*, 6(3): 38–43.
153. Kasabov N (1996) Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems. *Fuzzy Sets and Systems*, 82: 135–149.
154. Kennedy J, Eberhart RC, Yuhui S, Shi Y (2001) *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA.
155. Khosla A, Kumar S, Aggarwal KK (2006) Swarm intelligence and the Taguchi method for identification of fuzzy models. In: Fulcher J (ed.) *Advances in Applied Artificial Intelligence*. Idea Group, Hershey, PA: 273–295.
156. Kittler J, Hatef M, Duin RPW, Matas J (1998) On combining classifiers. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(3): 226–239.
157. Klein LA (2004) *Sensor and Data Fusion: A Tool for Information and Decision Making*. Intl. Society for Optical Engineering (SPIE), Bellingham, WA.
158. Knowles J, Corne D (2004) Memetic algorithms for multiobjective optimization: issues, methods and prospects. In: Krasnogor N, Smith JE, Hart WE (eds.) *Recent Advances in Memetic Algorithms*. Springer-Verlag, Berlin.
159. Kohonen T (1986) Learning vector quantization for pattern recognition. *Technical Report TKK-F-A601*, Helsinki University of Technology, Finland.
160. Kohonen T (2001) *Self-Organization and Associative Memory (3rd ed.)*. Springer-Verlag, Berlin.
161. Korb KB (2004) *Bayesian Artificial Intelligence*. CRC Press, Boca Raton, FL.
162. Kosko B (1992) *Neural Networks and Fuzzy Systems: a Dynamical Approach to Machine Intelligence*. Prentice Hall, Englewood Cliffs, NJ.
163. Koza J (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.
164. Koza J (1995) *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA.
165. Koza J (1999) *Genetic Programming III: Darwinian Inventions and Problem Solving*. Morgan Kaufmann, San Mateo, CA.
166. Kuncheva LI (2003) "Fuzzy" vs "non-fuzzy" in combining classifiers designed by boosting. *IEEE Trans. Fuzzy Systems*, 11: 729–741.
167. Kuncheva LI, Jain LC (2000) Designing classifier fusion systems by genetic algorithms. *IEEE Trans. Evolutionary Computation*, 4(4): 327–336.
168. Kuncheva LI, Whitaker CJ, Shipp CA, Duin RPW (2003) Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis and Applications*, 6: 22–31.
169. Kuncheva LI (2004) *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, New York, NY.

170. Kung SY, Mak MW, Lin SH (2004) *Biometric Authentication: A Machine Learning Approach*. Prentice Hall, Upper Saddle River, NJ.
171. Kurzweil R (1999) *The Age of Spiritual Machines: When Computers Exceed Human Intelligence*. Penguin, New York, NY.
172. Lam L (2000) Classifier combinations: implementations and theoretical issues. In: Kittler J, Roli F (eds.) *Multiple Classifier Systems*. Lecture Notes in Computer Science 1857, Springer-Verlag, Berlin: 78–86.
173. Lam L, Suen CY (1995) Optimal combination of pattern classifiers. *Pattern Recognition Letters*, 16: 945–954.
174. Lam L, Suen CY (1997) Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Trans. Systems, Man, and Cybernetics*, 27(5): 553–568.
175. Langdon WB (1998) *Data Structures and Genetic Programming: GP + Data Structures = Automatic Programming!* Kluwer Academic Press, Boston, MA.
176. Langton CG (1984) Self-reproduction in cellular automata. *Physica D*, 10: 1–2.
177. Lanier J (1995) Agents of Alienation. *Interactions*, 2(3): 67–72.
178. LeCun Y, Denker JS, Solla SA (1990) Optimal brain damage. In: *Advances in NIPS 2*, Morgan Kaufman, San Mateo, CA: 598–605.
179. Lee CS, Jian CC, Hsieh TC (2005) Ontology-based genetic fuzzy agent. In: *Proc. IEEE Intl. Fuzzy Systems Conf.*, Reno, NV, 22–25 May: 331–335.
180. Lee MA, Takagi H (1993) Integrating design stages of fuzzy systems using genetic algorithms. In: *Proc. 2nd IEEE Intl. Conf. Fuzzy Systems (FUZZ-IEEE'93)*, 28 March–1 April, San Francisco, CA. IEEE Computer Society Press, Los Alamitos, CA. 1: 612–617.
181. Lee MA, Esbensen H (1997) Fuzzy/multiobjective genetic systems for intelligent design tools and components. In: Pedrycz W (ed.) *Fuzzy Evolutionary Computation*. Kluwer Academic Publishers, Boston, MA: 57–81.
182. Leshno M, Lin V, Pinkus A, Schoken S (1993) Multi-layer feed-forward networks with a non-polynomial activation function can approximate any function. *Neural Networks*, 6: 861–867.
183. Lesser V (1995) Multiagent systems: an emerging subdiscipline of AI. *ACM Computing Surveys*, 27(3): 340–342.
184. Leung SC, Fulcher J (1997) Classification of user expertise level by neural networks. *Intl. J. Neural Systems*, 8(2): 155–171.
185. Levy S (1997) *Artificial Life: A Report From the Frontier Where Computers Meet Biology*. Vintage Books, New York, NY.
186. Lighthill J (1972) Artificial intelligence: a general survey. *Scientific Research Council of Britain*. March, SRC: 72–27.
187. Lin C-T, Lee CSG (1991) Neural network based fuzzy logic control and decision system. *IEEE Trans. Computers*, 40(12): 1320–1336.
188. Lin X, Yacoub S, Burns J, Simske S (2003) Performance analysis of pattern classifier combination by plurality voting. *Pattern Recognition Letters*, 24(12): 1795–1969.
189. Lin YT, Cercone N (1997) *Rough Sets and Data Mining: Analysis of Imprecise Data*. Kluwer Academic Publishers, New York, NY.
190. Lin YT (1999) Granular computing: fuzzy logic and rough sets. In: Zadeh LA, Kacprzyk J (eds.) *Computing with Words in Information/Intelligent Systems*. Springer-Verlag, Berlin.
191. Liu J, Tsui KC (2006) Toward Nature-inspired computing. *Communications ACM*, 49(10): 59–64.

192. Lohn ID, Reggia JA (1997) Automatic discovery of self-replicating structures in cellular automata. *IEEE Trans. Evolutionary Computation*, 1(3): 165–178.
193. Lohn JD, Hornby GS (2006) Evolvable hardware: using evolutionary computation to design and optimize hardware systems. *IEEE Computational Intelligence Magazine*, 1(1): 19–27.
194. Lucas P, van der Gaag L (1991) *Principles of Expert Systems*. Addison Wesley, Reading, MA.
195. Lukasiewicz J (1963) *Elements of Mathematical Logic*. Macmillan, New York, NY.
196. Maass W, Bishop CM (eds.) (1999) *Pulsed Neural Networks*. Bradford/MIT Press, Cambridge, MA.
197. Mahmoud Q, Yu L (2006) Making software agents user-friendly. *IEEE Computer*, 39(7): 94–96.
198. Mallat S (1999) *A Wavelet Tour of Signal Processing*. Academic Press, Boston, MA.
199. Mamdani E, Assilian S (1975) An experiment in linguistic synthesis with a fuzzy logic controller. *Intl. J. Man-Machine Studies*, 7(1): 1–13.
200. Mandelbrot BB (1985) *The Fractal Geometry of Nature: Updated and Augmented*. WH Freeman, New York, NY.
201. Mange D, Tomassin M (1998) *Bio-Inspired Computing Machines*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland.
202. McCarthy J (2005) The future of AI: a manifesto. *AI Magazine*, 26: 39–40.
203. McCullagh J, Choi B, Bluff K (1997) Genetic evolution of a neural network's input vector for meteorological estimation. In: Kasabov N, Kozma R, Ko K, Coghill G, Gedeon T (eds.) *Progress in Connectionist-Based Information Systems*. Springer-Verlag, Berlin: 1046–1049.
204. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin Mathematical Physics*, 5: 115–117.
205. McNeill D, Thro E (1994) *Fuzzy Logic: A Practical Approach*. Academic Press, Boston, MA.
206. Mead C (1989) *Analog VLSI and Neural Systems*. Addison Wesley, Reading, MA.
207. Michalewicz Z, Fogel DB (2000) *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin.
208. Miller DJ, Yan L (1999) Critic-driven ensemble classification. *IEEE Trans. Signal Processing*, 47(10): 2833–2844.
209. Minsky M, Papert S (1969) *Perceptrons (2nd ed)*. MIT Press, Cambridge, MA.
210. Nardi BA, Miller JR, Wright DJ (1998) Collaborative, Programmable Intelligent Agents. *Communications ACM*, 41(3): 96–104.
211. Naur P (2007) Computing versus human thinking. *Communications ACM*, 50(1): 85–93.
212. Neagu C-D (2000) Toxicity prediction using assemblies of hybrid fuzzy neural models. In: *Proc. 6th Intl. Conf. Knowledge-Based Intelligent and Engineering Systems (KES2002)*, 16–18 September, Milan, Italy. IOS Press, Amsterdam, The Netherlands: 1093–1098.
213. Neagu C-D, Gini G (2003) Neuro-fuzzy knowledge integration applied in toxicity prediction. In: Jain R, Abraham A, Faucher C, van der Zwaag BJ (eds.) *Innovations in Knowledge Engineering*. Advanced Knowledge International, Magill, South Australia: 311–342.

214. Neagu C-D, Palade V (1999) Fuzzy computing in a multi-purpose neural network implementation. In: Reusch B (ed.) *Computational Intelligence: Theory and Applications*. Lecture Notes in Computer Science 1625, Springer-Verlag, Berlin: 697–700.
215. Neagu C-D, Palade V (2003) A neuro-fuzzy approach for functional genomics data interpretation and analysis. *J. Neural Computing and Applications*. 12(3-4): 153–159.
216. Neapolitan RE (2003) *Learning Bayesian Networks*. Prentice Hall, Englewood Cliffs, NJ.
217. Nedjah N, Alba E, Mourelle LdM (2006) *Parallel Evolutionary Computations*. Springer-Verlag, Berlin.
218. Negnevitsky M (2005) *Artificial Intelligence: A Guide to Intelligent Systems (2nd ed)*. Prentice Hall, Englewood Cliffs, NJ.
219. Negoita MG, Neagu D, Palade V (2005) *Computational Intelligence: Engineering of Hybrid Systems*. Springer-Verlag, Berlin.
220. Negoita M, Agapie A, Fagarasan F (1994) The fusion of genetic algorithms and fuzzy logic: Application in expert systems and intelligent control. In: *Proc. IEEE/Nagoya University WWW Conf. Fuzzy Logic and Neural Networks/Genetic Algorithms*, August, Nagoya, Japan. IEEE Computer Society Press, Alamitos, CA: 130–133.
221. Negoita M, Mihaila D (1995) Intelligent techniques based on genetic evolution with applications to neural networks weights optimization. In: *Proc. 14th Intl. Congress Cybernetics*, 21–25 August, Namur, Belgium. Intl. Association for Cybernetics, Namur, Belgium.
222. Newell A, Simon HA (1976) Computer Science as empirical enquiry: symbols and search. *Communications ACM*, 19(3): 113–126.
223. Newell A (1990) *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.
224. Nowlan SJ, Hinto GE (1991) Evaluation of adaptive mixtures of competing experts. In: Lippmann RP, Moody JE, Touretzky DS (eds.) *Advances in Neural Information Processing Systems 3*. Morgan Kaufman, San Mateo, CA: 774–780.
225. Oeda S, Ichimura T, Yamashita T, Yoshida K (2003) A proposal of immune multi-agent neural networks and its applicaiton to medical diagnostic system for hepatobiliary disorders. In: Palade V, Howlett JR, Jain LC (eds.) *Knowledge-Based Intelligent Engineering Information Systems*. Sprigner-Verlag, New York, NY, II: 526–532.
226. Omondi AR, Rajapakse JC (eds.) (2006) *FPGA Implementations of Neural Networks*. Springer-Verlag, Dordrecht, The Netherlands.
227. Opitz D, Maclin R (1999) Popular ensemble methods: an empirical study. *J. AI Research*, 11: 169–198.
228. Ott E (2002) *Chaos in Dynamical Systems*. Cambridge University Press, UK.
229. Padgham L, Winikoff M (2004) *Developing Intelligent Agent Systems: A Practical Guide*. Wiley, New York, NY.
230. Pagliosa A, de Sá CC, Sasse FD (2005) Obtaining membership functions from a neuron fuzzy system extended by Kohonen network. In: Nakamatsu K, Abe JM (eds.) *Advances in Logic Based Intelligent Systems (Selected Papers of LAPTEC'2005)*. IOS Press, Amsterdam, The Netherlands: 42–49.
231. Pal SK, Shiu S (2004) *Foundations of Soft Computer-Based Reasoning*. Wiley, Hoboken, NJ.

232. Palade V, Negoita M, Arifon V (1999) Genetic algorithms optimization of knowledge extraction from neural networks. In: *Proc. 6th Intl. Conf. Neural Information Processing (ICONIP'99)*, November, Perth, Australia. IEEE Computer Society Press, Los Alamitos, CA: 752–758.
233. Palade V, Patton RJ, Uppal FJ, Quevedo J, Daley S (2002) Fault diagnosis of an industrial gas turbine using neuro-fuzzy methods. In: *Proc. 15th Intl. IFAC World Congress*, 21–26 July, Barcelona, Spain. Federation for Automatic Control: 2477–2482.
234. Papert S (1980) *Mindstorms*. Basic Books, New York, NY.
235. Parker DB (1985) Learning logic. *Technical Report TR-47*, Centre for Computational Research in Economics and Management Science, MIT.
236. Paun G (2002) *Membrane Computing: An Introduction*. Springer-Verlag, Berlin.
237. Pawlak Z (1991) *Rough Sets: Theoretical Aspects of Reasoning About Data*. Kluwer, Dordrecht, The Netherlands.
238. Pedrycz W (1993) Fuzzy neural networks and neurocomputations. *Fuzzy Sets and Systems*, 56: 1–28.
239. Pedrycz W (1997) *Computational Intelligence: An Introduction*. CRC Press, Boca Raton, FL.
240. Pedrycz W (1999) Computational Intelligence: an introduction. In: Szczepaniak PS (ed.) *Computational Intelligence and Applications*. Physica-Verlag, Berlin: 3–17.
241. Pellerin D, Thibault S (2005) *Practical FPGA Programming in C*. Prentice Hall, Englewood Cliffs, NJ.
242. Percival DB, Walden AT (2000) *Wavelet Methods for Time Series Analysis*. Cambridge University Press, UK.
243. Perrone MP, Cooper LN (1993) When networks disagree: Ensemble methods for neural networks. In: Mammone RJ (ed.) *Artificial Neural Networks for Speech and Vision*. Chapman and Hall, New York, NY: 126–142.
244. Pfeifer R, Bongard J (2007) *How the Body Shapes the Way We Think: A New View of Artificial Intelligence*. MIT Press, Cambridge, MA.
245. Pinker S (2001) How the mind works. (available online at <http://www.kurzweilai.net/meme/frame.html?m=4> – last accessed April 2007).
246. Pollack JB (2006) Mindless intelligence. *IEEE Intelligent Systems*, 21(3): 50–56.
247. Poole D, Mackworth A, Goebel R (1998) *Computational Intelligence – A Logical Approach*. Oxford University Press, New York, NY.
248. Powell MJD (1985) Radial basis functions for multivariate interpolation: a review. In: *Proc. IMA Conf. Algorithms for the Approximation of Functions and Data*, RMCS, Shrivenham, UK: 143–167.
249. Pritchard D, Negoita G (2006) A fuzzy – GA hybrid technique for optimisation of teaching sequences presented in ITSs. In: Reusch B (ed.) *Computational Intelligence, Theory and Applications (Proc. 8th Fuzzy Days Conf.)*, 29 September–1 October, Dortmund, Germany. Lecture Notes in Computer Science 3505, Springer-Verlag, Berlin: 311–316.
250. Quinlan JR (1986) Induction of decision trees. *Machine Learning*, 1(1): 81–106.
251. Quinlan JR (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA.
252. Quinlan JR (1996) Bagging, boosting, and C4.5. In: *Proc. AAAI'96*, Portland, OR. AAAI Press, Menlo Park, CA: 725–730.

253. Rechenberg I (1973) *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, Germany.
254. Rechenberg I (1994) Evolution strategy. In: Zurada J, Marks II RJ, Robinson C (eds.) *Computational Intelligence – Imitating Life*. IEEE Press, Piscataway, NJ.
255. Reidmiller M, Braub H (1992) RPROP: a fast adaptive learning algorithm, In: *Proc. Intl. Symp. Computer and Information Sciences*, November, Antalya, Turkey: 279–285. (ISCIS-VII)
256. Rieffel EG, Polak W (2000) Quantum computing for non-Physicists. *ACM Computing Surveys*, 32(3): 300–335.
257. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65: 386–408.
258. Rosenblatt F (1962) *The Principles of Neurodynamics*. Spartan Books, Washington, DC.
259. Ruckert U (2002) ULSI architectures for ANNs. *IEEE Micro*, May-June: 10–19.
260. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by backpropagating errors. In: Rumelhart DE, McClelland JL (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition I*. MIT Press, Cambridge, MA.
261. Russell S, Norvig P (2001) *Artificial Intelligence: A Modern Approach (2nd ed)*. Prentice Hall, Englewood Cliffs, NJ.
262. Samuel A (1959) Some studies in machine learning using the game of checkers. *IBM J.*, 3(3): 210–229.
263. Sanchez E, Tomassini M (eds.) (1996) *Towards Evolvable Hardware: The Evolutionary Engineering Approach*. Springer-Verlag, Berlin.
264. Schaffer JD (1984) Some experiments in machine learning using vector evaluated genetic algorithms. *PhD Thesis*, Vanderbilt University, Nashville, TN
265. Schalkof RJ (1997) *Artificial Neural Networks: Application to Ecology and Evolution*. McGraw Hill, New York, NY.
266. Sekanina L (2004) *Evolvable Components: From Theory to Hardware Implementation*. Springer-Verlag, Berlin.
267. Shann JJ, Fu HC (1995) A fuzzy neural network for rule acquiring on fuzzy control systems. *J. Fuzzy Sets and Systems*, 71: 345–357.
268. Sharkey AJC (ed.) (1999) *Combining Artificial Neural Networks: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, Berlin.
269. Sharkey AJC, Sharkey N (2006) The application of swarm intelligence to collective robots. In: Fulcher J (ed.) *Advances in Applied Artificial Intelligence*. Idea Group, Hershey, PA: 157–185.
270. Shawe-Taylor J, Cristianini N (2000) *Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, UK.
271. Shearer C, Caron P (2002) *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, UK.
272. Shimojima K, Fukuda T, Hasewaga I (1995) Self-tuning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm. *J. Fuzzy Sets and Systems*, 71: 294–309.
273. Simpson PK (1992) Fuzzy MIN-MAX neural networks – part 1: classification. *IEEE Trans. Neural Networks*, 3(5): 776–786.
274. Simpson PK (1993) Fuzzy MIN-MAX neural networks – part 2: clustering. *IEEE Trans. Fuzzy Systems*, 1(1): 32–45.

275. Sioutis C, Urlings P, Tweedale J, Ichalkaranje N (2004) Forming human-agent teams within hostile environments. In: Fulcher J, Jain LC (eds.) *Applied Intelligent Systems: New Directions*. Springer-Verlag, Berlin: 255–279.
276. Sipper M (1997) *Evolution of Parallel Cellular Machines – The Cellular Programming Approach*. Springer-Verlag, Berlin.
277. Sipper M, Sanchez E, Mange D, Tomassini M, Perez-Urbe A, Stauffer A (1997) A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Trans. Evolutionary Computation*, 1(1): 83–97.
278. Sipper M, Mange D, Sanchez E (1999) Quo Vadis Evolvable Hardware? *Communications ACM*, 42(4): 50–59.
279. Sipper M (2002) *Machine Nature: The Coming Age of Bio-Inspired Computing*. McGraw-Hill, New York, NY.
280. Sisman-Yilmaz NA, Alpaslan FN, Jain LC (2004) Fuzzy multivariate auto-regression method and its application. In: Fulcher J, Jain LC (eds.) *Applied Intelligent Systems: New Directions*. Springer-Verlag, Berlin: 281–300.
281. Sowa JF (2000) *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks-Cole, Pacific Grove, CA.
282. Sprott JC (2003) *Chaos and Time Series Analysis*. Oxford University Press, UK.
283. Stair RM, Reynolds GW (1999) *Principles of Information Systems (4th ed)*. Thomson, Cambridge, MA.
284. Stevens M (1997) *Bayesian Methods for Mixtures of Normal Distributions*. Oxford University Press, Oxford, UK.
285. Stock O, Zancanaro M (eds.) (2005) *Multimodal Intelligent Information Presentation: Text, Speech and Language Technology*. Springer-Verlag, Berlin.
286. Sundarajan N, Satchandran P (1998) *Parallel Architectures for Artificial Neural Networks*. IEEE Press, Los Alamitos, CA.
287. Sugeno M (1985) *Industrial Applications of Industrial Control*. North Holland, New York, NY.
288. Takagi H (1994) Cooperative systems of neural networks and fuzzy logic and its applicaiton to consumer products. In: Yen J, Langari R, Zadeh LA (eds.) *Industrial Applications of Fuzzy Control and Intelligent Systems*. Van Nostrand Reinhold, New York, NY.
289. Tan KC, Khor EF, Lee TH (2005) *Multiobjective Evolutionary Algorithms and Applications*. Springer-Verlag, London, UK.
290. Tesauro G (1992) Temporal difference learning of backgammon strategy. In: Shafer G, Pearl J (eds.) *Proc. Intl. Conf. Machine Learning – ICML92*, July, Aberdeen, UK, Morgan Kaufmann, San Francisco, CA: 451–457.
291. Teuscher C (2006) Biologically un-inspired computational intelligence. *Communications ACM*, 49(11): 27–29.
292. Toffoli T, Margolus N (1987) *Cellular Automata Machines*. MIT Press, Cambridge, MA.
293. Tollenaere T (1990) SuperSAB: fast adaptive backpropagation with good scaling properties. *Neural Networks*, 7(5): 561–573.
294. Tran C, Abraham A, Jain LC (2006) Soft computing paradigms and regression trees in decision support systems. In: Fulcher J (ed.) *Advances in Applied Artificial Intelligence*. Idea Group, Hershey, PA: 1–28.
295. Uppal FJ, Patton RJ, Palade V (2002) Neuro-fuzzy based fault diagnosis applied to an electro-pneumatic valve. In: *Proc. 15th IFAC World*

- Congress*, 21–26 July, Barcelona, Spain. Intl. Federation for Automatic Control: 2483–2488.
296. van Eck J, Waltham L, van den Berg J, Kaymak V (2006) Visualizing the CI Field. *IEEE Computational Intelligence Magazine*, 1(4): 6–10.
 297. Vapnik VN (1998) *Statistical Learning Theory*. Wiley, New York, NY.
 298. Verikas A, Lipnickas A, Malmqvist K, Bacauskiene M, Gelzinis A (1999) Soft combination of neural classifiers: a comparative study. *Pattern Recognition Letters*, 20: 429–444.
 299. Verikas A, Lipnickas A (2002) Fusing neural networks through space partitioning and fuzzy integration. *Neural Processing Letters*, 16: 53–65.
 300. Verma B, Panchal R (2006) Neural networks for the classification of benign and malignant patterns in digital mammograms. In: Fulcher J (ed.) *Advances in Applied Artificial Intelligence*. Idea Group, Hershey, PA: 251–272.
 301. Von Altrock (1995) *Fuzzy Logic and Neurofuzzy Applications Explained*. Prentice Hall, Englewood Cliffs, NJ.
 302. Von Neumann J (1958) *The Computer and the Brain*. Yale University Press, New Haven, CT.
 303. Wang D, Fang S-C (1997) A genetics-based approach for aggregated production planning in a fuzzy environment. *IEEE Trans. Systems, Man and Cybernetics*, 27(5): 636–645.
 304. Watson I (1997) *Applying Base-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann, San Francisco, CA.
 305. Werbos P (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. *PhD Thesis*, Harvard University, Cambridge, MA.
 306. Werbos P (1994) *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley, New York, NY.
 307. Wermter S, Sun R (2000) *Hybrid Neural Systems*. Springer-Verlag, Berlin.
 308. Widrow B, Hoff ME (1960) Adaptive switching circuits. In: *Proc. IRE WESCON Convention Record: Part 4, Computers: Man-Machine Systems*, Los Angeles, CA: 96–104.
 309. Wiener N (1948) *Cybernetics*. Wiley, New York, NY.
 310. Wolfram S (1997) *Cellular Automata and Complexity – Collected Papers*. Addison Wesley, Reading, MA.
 311. Williams CP, Clearwater SH (2000) *Ultimate Zero and One: Computing at the Quantum Frontier*. Springer-Verlag, Berlin.
 312. Williams J (1990) When expert systems are wrong. In: *Proc. ACM SIGBDP Conf. – Trends and Directions in Expert Systems*. Orlando, FL, ACM Press, New York, NY: 661–669.
 313. Witten IH, Frank E (2005) *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufman, San Francisco, CA.
 314. Wong B, Lai V, Lam J (2000) A bibliography of neural network business applications research: 1994 – 1998. *Computer and Operations Research*, 23: 1045–1076.
 315. Wong HC, Sycara K (1999) Adding security and trust to multi-agent systems. In: *Proc. Autonomous Agents'99*, May, Seattle, WA: 149–161.
 316. Wooldridge M, Jennings NR (1995) Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2): 115–152.
 317. Wooldridge M (2002) *An Introduction to Multiagent Systems*. Wiley, Chichester, UK.

318. Xu L, Krzyzak A, Suen CY (1992) Methods of combining multiple classifiers and their application to handwriting recognition. *IEEE Trans. Systems, Man, and Cybernetics*, 22: 418–435.
319. Yager R (1992) Implementing fuzzy logic controller using a neural network framework. *Fuzzy Sets and Systems*, 48: 53–64.
320. Yao X (1999) Following the path to evolvable hardware. *Communications ACM*, 42(4): 47–49.
321. Yao YY (2000) Granular computing: basic issues and possible solutions. In: *Proc. 5th Joint Conf. Information Sciences*, 27 February–3 March, Atlantic City, NJ: 186–189.
322. Yao X, Higuchi T (1999) Promises and challenges of evolvable hardware. *IEEE Trans. Systems, Man and Cybernetics-Part C*, 29(1): 87–89.
323. Zadeh LA (1965) Fuzzy sets. *Information and Control*, 8: 338–353.
324. Zadeh LA (1978) Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1: 3–28.
325. Zadeh LA (1994) Fuzzy logic, neural networks, and soft computing. *Communications ACM*, 37(3): 77–84.
326. Zadeh LA (1997) Towards a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 19: 111–127.
327. Zhang J, Morris J (1996) Process modeling fault diagnosis using fuzzy neural networks. *Fuzzy Sets and Systems*, 79: 127–140.
328. Zhang M (2008) *Artificial Higher-Order Neural Networks for Economics and Business*. IGI, Hershey, PA.
329. Zhang M, Fulcher J, Scofield R (1997) Rainfall estimation using artificial neural network group. *Neurocomputing*, 16(2): 97–115.
330. Zhang Y-Q, Fraser MD, Gagliano RA, Kandel A (2000) Granular neural networks for numerical-linguistic data fusion and knowledge discovery. *IEEE Trans. Neural Networks*, 11(3): 658–667.
331. Zhou Z-H, Wu J, Tang W (2002) Artificial neural network ensembles. *Artificial Intelligence*, 137(1–2): 239–263.
332. Zykov V, et al. (2005) Self-reproducing machines. *Nature*, 435(7038): 163–164.

Resources

1 Key Books

In addition to the specific listings below, the following Springer book series are recommended for general reference: *Studies in Computational Intelligence*, *Studies in Fuzziness and Soft Computing*, and *Advances in Soft Computing*.

1.1 Computational Intelligence

Chen SH, Wang P, Wang PP (2006) *Computational Intelligence in Economics and Finance*. Springer-Verlag, Berlin.

Chen Z (2000) *Computational Intelligence for Decision Support*. CRC Press, Boca Raton, FL.

Dick S, Kander A (2005) *Computational Intelligence in Software Quality Assurance*. World Scientific, Singapore.

Duch W, Mandziuk J (eds.) (2007) *Challenges for Computational Intelligence*. Springer-Verlag, Berlin.

Eberhart R, Simpson P, Dobbins R (1996) *Computational Intelligence PC Tools*. Academic Press, Boston, MA.

Englebrecht AP (2003) *Computational Intelligence: An Introduction*. Wiley, New York, NY.

Fogel DB, Robinson CJ (eds.) (2003) *Computational Intelligence: The Experts Speak*. Wiley, New York, NY.

Kecman V (2001) *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. MIT Press, Cambridge, MA.

King RE (1999) *Computational Intelligence in Control Engineering*. Marcel Dekker, New York, NY.

Konar A (2005) *Computational Intelligence: Principles, Techniques, and Applications*. Springer-Verlag, Berlin.

Kusiak A (2000) *Computational Intelligence in Design and Manufacturing*. Wiley, New York, NY.

Ovaska SJ (ed.) (2004) *Computationally Intelligent Hybrid Systems: The Fusion of Soft Computing and Hard Computing*. Wiley, New York, NY.

Pedrycz W, Peters JF (1998) *Computational Intelligence in Software Engineering*. World Scientific, Singapore.

Pedrycz W (1997) *Computational Intelligence: An Introduction*. CRC Press, Boca Raton, FL.

Poole D, Mackworth A, Goebel R (1998) *Computational Intelligence: A Logical Approach*. Oxford University Press, New York, NY.

1.2 Artificial Neural Networks

Anderson JA, Rosenfeld E (eds.) (1988) *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA.

Anderson JA, Pellionisz A, Rosenfeld E (eds.) *Neurocomputing 2: Directions for Research*. MIT Press, Cambridge, MA.

Beale R, Jackson T (1990) *Neural Computing: An Introduction*. Adam Hilger, Bristol, UK.

Bigus JP (1996) *Data Mining with Neural Networks: Solving Business Problems – From Application Development to Decision Support*. McGraw Hill, New York, NY.

Bishop CM (1995) *Neural Networks for Pattern Recognition*. Oxford University Press, UK.

Fiesler E, Beale R (1997) *Handbook of Neural Computation*. Oxford University Press/Institute of Physics, New York, NY.

Haykin SY (1999) *Neural Networks: A Comprehensive Foundation (2nd ed.)*. Prentice Hall, Upper Saddle River, NJ.

Kohonen T (2001) *Self-Organization and Associative Memory (3rd ed)*. Springer-Verlag, Berlin.

Orr GB , Mueller K-R (eds.) (1998) *Neural Networks: Tricks of the Trade*. Springer-Verlag, Berlin.

Principe JC, Euliano NR, Lefebvre WC (2000) *Neural and Adaptive Systems: Fundamentals Through Simulations*. Wiley, New York, NY.

Reed RD, Marks II RJ (1999) *Neural Smithing: Supervised Learning for Feed-forward Artificial Neural Networks*. MIT Press, Cambridge, MA.

Ripley BD (1996) *Pattern Recognition and Neural Networks*. Cambridge University Press, UK.

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by backpropagating errors. In; Rumelhart DE, McClelland JL (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition I*. MIT Press, Cambridge, MA.

Sharkey AJC (1999) *Combining Artificial Neural Networks: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, Berlin.

Werbos P (1994) *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley, New York, NY.

1.3 Evolutionary Methods

Bäck T (1996) *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NY.

Bäck T, Fogel DB, Michalewicz Z (1997) *Handbook of Evolutionary Computation*. Oxford University Press, New York, NY.

Banzhaf W (1998) *Genetic Programming: An Introduction*. Morgan Kaufmann, San Francisco, CA.

Bonabeau E, Dorigo M, Theaulaz G (1999) *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, UK.

Davis L (ed.) (1991) *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY.

Dorigo M, Stützle T (2004) *Ant Colony Optimization*. MIT Press, Cambridge, MA.

Engelbrecht AP (2005) *Fundamentals of Computational Swarm Intelligence*. Wiley, New York, NY.

Fogel LJ (1999) *Intelligence Through Simulated Evolution*. Wiley, New York, NY.

Fogel DB (2005) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Wiley, New York, NY.

Goldberg DE, Deb K (1991) *Foundations of Genetic Algorithms: A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*. Morgan Kaufman, San Mateo, CA.

Grana M, Duro R, d'Anjou A, Wang PP (2004) *Information Processing in Evolutionary Algorithms: From Industrial Applications to Academic Speculations*. Springer-Verlag, Berlin.

Greenwood GW, Tyrrell AM (2006) *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-adaptive Systems*. Wiley, New York, NY.

Higuchi T, Yong L, Yao X (eds.) (1999) *Evolvable Hardware*. Springer-Verlag, Berlin.

Holland JJ (1992) *Adaptation in Natural and Artificial Systems (2nd ed)*. MIT Press, Cambridge, MA.

Kennedy J, Eberhart RC, Yuhui S, Shi Y (2001) *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA.

Koza J (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

Langdon WB (1998) *Data Structures and Genetic Programming: GP + Data Structures = Automatic Programming!* Kluwer Academic Press, Boston, MA.

Michalewicz Z (1996) *Genetic Algorithms + Data Structures = Evolutionary Programs*. Springer-Verlag, Berlin.

Mitchell M (1995) *Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.

Sekanina L, Arsian T (2005) *Evolvable Components: From Theory to Hardware Implementations*. Springer-Verlag, Berlin.

Tyrrell AM (2006) *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*. Wiley, New York, NY.

1.4 Fuzzy Systems

Cox E (1994) *The Fuzzy Systems Handbook*. AP Professional Books, Boston, MA.

Fogel LJ, Owens AJ (eds.) (1997) *Handbook of Fuzzy Computation*, Oxford University Press, New York, NY.

Jang J-S R, Sun C-T, Mizutani E (1993) *Neuro-Fuzzy and Soft Computing: a Computational Approach to Learning and Machine Intelligence*. Prentice Hall, Englewood Cliffs, NJ.

Kosko B (1997) *Fuzzy Engineering*. Prentice Hall, Upper Saddle River, NJ.

McNeill D, Thro E (1994) *Fuzzy Logic: A Practical Approach*. Academic Press, Boston, MA.

Nauck D (1997) *Foundations of Neuro-Fuzzy Systems*. Wiley, New York, NY.

Pedrycz W, Gomide F (1998) *An Introduction to Fuzzy Sets: Analysis and Design*. MIT Press, Cambridge, MA.

Ruspini E, Bonissone P, Pedrycz W (eds.) *Handbook of Fuzzy Computation*. Oxford University Press, New York, NY.

Von Altrock (1995) *Fuzzy Logic and Neurofuzzy Applications Explained*. Prentice Hall, Englewood Cliffs, NJ.

1.5 Other

Brown M, Harris C (1994) *Neurofuzzy Adaptive Modeling and Control*. Prentice Hall, Englewood Cliffs, NJ.

Giarratano JC, Riley G (2005) *Expert Systems: Principles and Programming (4th ed)*. Thomson, Boston, MA.

Ignizio J (1991) *Introduction to Expert Systems*. McGraw-Hill, New York, NY.

Jackson P (1998) *Introduction to Expert Systems (3rd ed)*. Addison Wesley, Reading, MA.

Negnevitsky M (2005) *Artificial Intelligence: A Guide to Intelligent Systems (2nd ed)*. Prentice Hall, Englewood Cliffs, NJ.

Padgham L, Winikoff M (2004) *Developing Intelligent Agent Systems: A Practical Guide*. Wiley, New York, NY.

Pedrycz W (ed.) (1997) *Fuzzy Evolutionary Computing*. Kluwer Academic Publishers, New York, NY.

Sipper M (2002) *Machine Nature: The Coming Age of Bio-Inspired Computing*. McGraw Hill, New York, NY.

Wolfram S (1997) *Cellular Automata and Complexity – Collected Papers*. Addison Wesley, Reading, MA.

Wooldridge M (2002) *An Introduction to Multiagent Systems*. Wiley, Chichester, UK.

Zomaya AY (ed.) (2006) *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*. Springer-Verlag, Berlin.

2 Key Survey/Review Articles

2.1 Artificial Neural Networks

Carpenter GA, Grossberg SA (1987) A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Understanding*, 37: 54–115.

Hinton GE (1992) How neural networks learn through experience. *Scientific American*. 267: 144–151.

Hopfield JJ (1984) Neurons with graded response have collective computational properties like those in two-state neurons. *Proc. National Academy Science*, 81: 3088–3092.

Lipmann RP (1987) An introduction to computing with neural networks. *IEEE ASSP Magazine*, 1: 4–42.

2.2 Evolutionary Methods

Hinchey MG, Sterritt R, Rouff C (2007) Swarms and swarm intelligence. *IEEE Computer*, 40(4): 111–113.

2.3 Fuzzy Systems

Zadeh L (1994) Fuzzy logic, neural networks, and soft computing. *Communications ACM*, 37(3): 77–84.

2.4 Other

Allen J (1998) AI growing up: the challenges and opportunities. *AI Magazine*, Winter: 32–45.

Bauch T, et al. (2006) How AI and multi-robot systems research will accelerate our understanding of social animal behavior. *Proc. IEEE*, July: 1445–1463.

Dasgupta D, Attoh-Okine N (1997) Immunity-based systems: a survey. In: *Proc. IEEE Intl. Conf. Systems, Man and Cybernetics*. Orlando, FL. IEEE Computer Society Press, Los Alamitos, CA: 326–331.

Hearst M, Hirsh H (2000) AIs greatest trends and controversies. *IEEE Intelligent Systems*, January/February: 8–17.

Hendler J (2006) Introducing the Future of AI. *IEEE Intelligent Systems*, 21(3): 2–4.

Lesser V (1995) Multiagent systems: an emerging subdiscipline of AI. *ACM Computing Surveys*, 27(3): 340–342.

Williams C (1986) Expert systems, knowledge engineering, and AI tools – an overview. *IEEE Expert*, 1(2): 2–6.

Wooldridge M, Jennings JR (1995) Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2): 115–152.

Zykov V, et al. (2005) Self-reproducing machines. *Nature*, 435(7038): 163–164.

3 Organizations, Societies, Special Interest Groups, Journals

3.1 Computational Intelligence

Computational Intelligence (Blackwell) {1.415}¹⁸

IEEE Computational Intelligence Magazine (IEEE CI Society)

IEEE Intelligent Systems (IEEE Computer Society) {2.413}

Intl. J. Computational Intelligence
(World Academy of Science Engineering and Technology)

Intl. J. Computational Intelligence and Applications (World Scientific)

Intl. J. Computational Intelligence and Organizations (Lawrence Erlbaum and Associates)

Intl. J. Computational Intelligence Research (Research India Publications)

Intl. J. Computational Intelligence Theory and Practice (Serials Publication)

International Journal of Intelligent Systems (Wiley)

Journal of Advanced Computational Intelligence and Intelligent Informatics
(Fuji Technology Press)

3.2 Artificial Neural Networks

IEEE Transactions on Neural Networks (IEEE Neural Network Society)
{2.620}

International Journal of Neural Systems (World Scientific)

Network – Computation in Neural Systems (MIT Press) {1.0}

Neural Computation (MIT Press) {2.229}

Neural Networks International Neural Networks Society (Elsevier) {2.0}

¹⁸ 2006 Thomson ISI Journal Citation Reports – Science {impact factor}.

Neural Processing Letters (Kluwer) {0.753}

Neurocomputing (Elsevier) {0.860}

3.3 Evolutionary Methods

Evolutionary Computation (MIT Press) {1.325}

Genetic Programming and Evolvable Machines J. (Kluwer)

IEEE Transactions on Evolutionary Computation (IEEE CI Society) {1.325}

3.4 Fuzzy Systems

Fuzzy Optimization and Decision Making (Springer)

Fuzzy Sets and Systems (Elsevier) {1.181}

IEEE Transactions on Fuzzy Systems (IEEE CI Society) {1.803}

Intl. J. Soft Computing and Intelligence (Intl. Fuzzy Systems Association)

Intl. J. Uncertainty, Fuzziness & Knowledge-Based Systems (World Scientific)

J. Intelligent and Fuzzy Systems: Applications in Engineering and Technology (IOS Press)

3.5 Other

AI Magazine (Association for the Advancement of Artificial Intelligence) {1.0}

Applied Soft Computing (Elsevier)

Artificial Intelligence (Elsevier) {2.271}

Artificial Life (MIT Press) {1.769}

Autonomous Agents and Multi-Agent Systems (Springer) {1.974}

Connection Science (Taylor and Francis) {1.297}

IEEE Trans. Knowledge and Data Engineering (IEEE Computer Society)

IEEE Transactions on Systems, Man and Cybernetics (IEEE SMC Society)

Intl. J. Intelligent Systems (Wiley)

J. Artificial Intelligence Research (AAAI Press) {1.795}

J. Intelligent Information Systems (Springer)

J. Machine Learning Research (MIT Press) {2.255}

KES Journal: Innovation in Knowledge-Based Intelligent Engineering Systems
(KES International)

Machine Learning (Springer) {2.654}

Soft Computing: A Fusion of Foundations, Methodology, and Applications
(Springer) {0.516}

4 Key International Conferences/Workshops

Congress on Evolutionary Computation – CEC (IEEE)

European Symposium ANNs (ENNS, INNS, IEEE-CIS)

Genetic and Evolutionary Computation Conf. (GECCO) (ACM SIGEVO)

Neural Information Processing Symposium – NIPS [published as *Advances in Neural Information Processing Systems*. Morgan Kaufmann, San Francisco, CA] (NIPS Foundation)

Intl. Conf. Fuzzy Systems – FUZZ-IEEE (IEEE)

Intl. Joint Conf. Neural Networks (IEEE/Intl. Neural Network Society)

Intl. Conf. Knowledge-Based Intelligent Information Engineering Systems
(KES International)

World Congress on Computational Intelligence – WCCI (IEEE)

5 (Open Source) Software

Stuttgart Neural Network Simulator

<http://www-ra.informatik.uni-tuebingen.de/SNNS>

<http://www.scilab.org/> (link to ANN, EVOL toolboxes)

http://www.mindmedia.com/links/mind_tools_software_neural_network_software.html (FAST ANN library)

<http://neuralnetworks.ai-depot.com/Software.html>
(includes Freeware, Shareware and Open Source)

<http://www.geocities.com/adotsaha/NNinExcel.html>

<http://sourceforge.net>

(search on 'Computational Intelligence' (CILib), 'Neural Network' (FANN), 'Genetic Algorithm' (GAUL), 'Fuzzy Logic' (FFLL), 'Swarm Intelligence')

<http://fann.sourceforge.net> (Fast ANN)

<http://simbrain.sourceforge.net>

<http://gaul.sourceforge.net> (Genetic Algorithm Utility Library)

<http://ffll.sourceforge.net> (FreeFuzzy Logic Library)

http://dmoz.org/Computers/Artificial_Intelligence/Fuzzy
(Open Source Fuzzy Inference Engine for Java)

http://dmoz.org/Computers/Artificial_Intelligence/Genetic_Programming/Algorithms/

<http://aaai.org/aitopics/html/soft.html>

<http://www.openchannelfoundation.org/> (AI and Expert Systems)

<http://opensource.arc.nasa.gov/> (link to JavaGenes)

<http://www.genetic-programming.org/gpftpsite.html> (GP and GA)

<http://geneticalgorithm.ai-depot.com/Libraries.html>

<http://www.jaga.org> (Java API for GAs)

<http://www.gamedia.com/neuralfuzzy.html> (ANNs, GAs, Fuzzy Logic)

<http://www.ghg.net/clips/CLIPS.html> (C-language Inference Production System – CLIPS)

http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyClips/fuzzyCLIPS/index.html
(FuzzyCLIPS)

<http://www.fizyka.umk.pl/~ duch/software.html>

6 Data Bases

UCI Knowledge Discovery in Databases Repository
<http://kdd.ics.uci.edu/>

University of California, Irvine Machine Learning Data Repository
<http://mlearn.ics.uci.edu/MLRepository.html>

**Preprocessing, Visualization,
Systems Integration**

Data Reduction for Pattern Recognition and Data Analysis

Tommy W.S. Chow and Di Huang

Department of Electronic Engineering, City University of Hong Kong,
eetchow@cityu.edu.hk, sshh007@hotmail.com

1 Introduction

Pattern recognition [5, 13, 58] involves various human activities of great practical significance, such as data-based bankruptcy prediction, speech/image recognition, machine fault detection and cancer diagnosis. Clearly, it would be immensely useful to build machines to fulfill pattern recognition tasks in a reliable and efficient way. The most general and most natural pattern recognition frameworks mainly rely on statistical characterizations of patterns with an assumption that they are generated by a probabilistic system. Research on neural pattern recognition has been widely conducted during the past few decades. In contrast to statistical methods, no assumptions (*a priori* knowledge) are required for building a neural pattern recognition framework. Despite the fact that different pattern recognition systems use different working mechanisms, the basic procedures of all these systems are basically the same. A typical pattern recognition procedure generally consists of three sequential parts – a sensing model for collecting and preprocessing raw data from real sites, a data processing model (which includes feature extraction/selection and pattern selection), and a recognition/classification model [13, 58]. When one is handling a pattern recognition process, the following basic issues must be addressed:

- *How to process the raw data for a pattern recognition task?* This issue concerns the sensing and preprocessing stage of pattern recognition;
- *How to determine appropriate data for a given pattern recognition model?* This is a very important concern in the data processing stage. Deleting noisy or redundant data (including features and patterns) invariably leads to enhanced recognition performance;
- *How to design an appropriate classifier based on a given data set?* This topic has been widely discussed in the pattern recognition community.

Various learning algorithms and models have been proposed in an attempt to enhance recognition accuracy as much as possible, and in a fashion that is as simple as possible.

Basically, through eliminating ‘noisy’ data (such as noisy samples and irrelevant features) and compressing redundant samples/features, a data processing technique is used to reduce the data volume without causing the loss of useful information. The main merits of such data processing include enhancing the scalability, recognition accuracy, computational and measurement efficiency, as well as to facilitate interpretation of the entire pattern recognition procedure [6, 24, 43]. As the size of data has significantly increased in recent applications, data preprocessing has become essential in many pattern recognition procedures. In this Chapter, data reduction/selection is specifically denoted as reduction/selection of data samples.

2 Data Reduction

While computer technology grows at an unprecedented pace, the size of data increases to an extent making pattern recognition incommodious. Data reduction therefore holds increasing appeal to researchers, although the use of *more* data samples can usually lead to more accurate pattern recognition results. With the aim of enhancing the overall computational efficiency, a huge pattern set is usually first reduced to a small representative (informative) pattern set on which pattern recognition models are built. It is generally assumed that data reduction should introduce no or minimal effect on final recognition results.

The simplest data reduction method is to sample the data in a random or stratified way [10]. In these methods, the user just needs to randomly draw the desired amount of samples from a data set. Generally speaking, these are very simple methods so that they can be easily implemented, and also have negligible computational burden. Thus, they have been used as an evaluation baseline in many studies. These random sampling schemes are clearly not sufficiently sophisticated to guarantee stable performance. They are so simple that they are likely to cause a loss of important data distribution information [8].

A number of more sophisticated data reduction techniques have also been developed. According to the working mechanism, data reduction models can broadly be categorized as ‘filter’ and ‘wrapper’ models. A filter model works prior to pattern recognition and is totally independent of the training of recognition models, whilst a wrapper data reduction process is embedded within the training process. The results of recognition training play a vital role in a wrapper data reduction process.

2.1 Wrapper Methods

Assuming that all patterns are not equally informative (useful) to a pattern recognition learning algorithm [63], wrapper models modify the original data distribution according to the behavior of a pattern recognition hypothesis. Useful patterns are selected into a learning process with higher probability than others.

In general, wrapper data reduction schemes [15, 25, 30, 52] start with a random data subset. A pattern recognition hypothesis is constructed using the data subset. Then, according to the performance of the resulting recognition model, the selected data subset is gradually modified. This data-modifying-model-building process repeats until the recognition accuracy cannot be improved further.

The wrapper method is model-dependent. Further, as the original data distribution has been distorted, the data reduction results may not be useful for unsupervised pattern recognition tasks, such as density estimation and data visualization. As mentioned earlier, wrapper models generally consider the samples likely to be incorrectly recognized as being ‘informative’. Samples with more information have a higher probability of participating in training the wrapper models. With this mechanism, wrapper models are likely to fail by confusing outliers with real informative samples because the former always have relatively high recognition uncertainty [55]. This is the main shortcoming of wrapper models.

2.2 Filter Methods

In contrast with wrapper methods, filter models are *independent* of recognition model training. Uncertain sampling – an example filter model – employs a classifier A to determine the informative patterns for building another classifier that may be very different [40]. ‘Informative’ patterns are those that can be correctly classified by A with lowest certainty. Most filter models explore the data distribution information instead of the classification results when conducting data reduction.

The basic aim of filter models is to determine a representative set – in other words, a reduced data set – which preserves the original data distribution as much as possible [2, 18, 32, 46, 68]. These methods are thus found to be versatile. Apart from classification, filter models can work for other pattern recognition tasks, such as data visualization and data clustering. Vector quantization error (VQE) is widely used for filter data reduction [2, 17, 30]. VQE [18] measures the extent of similarity between each given pattern and its nearest representative. The smaller the vector quantization error, or the closer that patterns are to their corresponding representatives, the better the data reduction result. Through minimizing the VQE, representative data is

found. Self-organizing maps (SOMs) [32] are a typical descent-based algorithm designed for minimizing the VQE. It is well known however that it is difficult to determine the learning parameters for a SOM, since they are problem-dependent. Also, VQE-based data reduction models often generate new data points in a given data space rather than select data points from an original data set. The generated data points may have no physical meaning, thus making them unsuitable for direct use in many pattern recognition procedures.

Another popular type of filter method is based upon probability density distribution [46, 62]. The crucial issue of density-based methods lies in the estimation of probability density functions (pdf) underlying a given data set. In [68], density is estimated by employing the maximum likelihood learning algorithm. This approach may not be efficient enough for tackling complex data distribution. In [2, 46], a much simpler and more efficient strategy is used to analyze density. The basic idea of this strategy is that the density of a pattern x is inversely related to the distance between x and its k th nearest neighbor. In other words, when the distance between a pattern and its neighbor is small, the probability density at that pattern must be high. Based on this idea, the density around each pattern is evaluated. The most dense pattern – say x_d – is then identified and is placed into the representative set. The patterns around x_d are rejected during the remainder of the representative selection process. This operation repeats until there is no pattern left for representative selection. Without involving the learning processes to build a density estimator, the above density analysis is more efficient than the maximum-likelihood approach used in [68]. However, as this approach [2, 46] requires the calculation of distances between *all* possible pattern pairs, they are very expensive in terms of computation and memory requirements, especially when a large amount of patterns is given.

2.3 Examples of Filter Methods

In order to provide a sound discussion on data reduction models, two typical distribution-based data reduction frameworks are detailed in this Section, these being the multi-scale and entropy-based data reduction methods.

The Multi-Scale Method

The multi-scale method is a typical density-based filter data reduction method, in which the pattern density is analyzed according to the distance of that pattern from its neighbor [51]. All patterns are then ranked in order of density. Based on the pattern order, the representatives are recursively determined. Given a datum X , this method can be briefly stated as follows:

Algorithm 1 The Multi-Scale Method

step 1. Determine the parameter k which is closely related to the size of the data region covered by a representative;

step 2. Calculate the distance between all possible pattern pairs in X ;

step 3.

repeat

for each pattern in X **do**

 (i) Determine the distance between the pattern and its k th neighbor;

 (ii) According to these distances, identify the most dense pattern – say, x_d – and mark it as representative;

 (iii) Draw a circle with center x_d and radius $2rad_d$, where rad_d is the distance between x_d and its k th neighbor;

 (iv) Delete all patterns which fall within the circle.

until no remaining pattern in X

Entropy-Based Data Reduction Method

Recently, a new density-based method has been proposed. This method relies on the representative entropy (RE) to guide the data reduction process, and thus is named ‘REDR’ (representative entropy data reduction) [27]. Assuming that R is the result of the data reduction process, the probability of x ($x \in X$) being represented by r_j ($r_j \in R$) is then $p(r_j | x)$. Further, the sum of the representative probabilities of x is 1, that is, $\sum_{j=1}^K p(r_j | x) = 1$. Ideally, each pattern in X is close to one and only one representative. In terms of probability, it is expected that $p(r_i | x)$ will be zero for all i except one. The more uneven the distribution of the representation information of R to X , the better the representative set R will be. This is the rationale behind ‘representative entropy’ (RE). Given a data set X and a representative set R , RE is defined as

$$RE(R; X) = \frac{1}{N \log(K)} \sum_{all\ x \in X} \sum_{r \in R} -p(r_j | x) \log(p(r_j | x)) \quad (1)$$

where N and K are the size of X and R , respectively. Assuming that a representative covers the L original patterns nearest to it, we have

$$p(r_j | x) \propto s(r_j, x) = \begin{cases} 1 - \frac{d(x, r_j)}{Radius(r_j)}, & d(x, r_j) \leq Radius(r_j) \\ 0, & otherwise \end{cases} \quad (2)$$

where $Radius(r_j)$ is the distance of r_j to the $(L + 1)$ th pattern nearest to it. After normalization, this becomes

$$p(r_j | x) = \frac{s(r_j, x)}{\sum_{r \in R} s(r_j, x)} \quad (3)$$

The REDR method includes two sequential stages, these being (i) a forward search stage, followed by (ii) a RE-based stepwise search stage. At the

beginning, a set of data points, say R_0 , is randomly drawn from a given data set, and the representative set R is empty. Then, the forward search is conducted on R_0 to recursively place the appropriate representatives into R . This process stops when R_0 has been completely scanned, or R has become a desirable size. Following the forward process is a RE based stepwise process, in which a pattern is firstly identified as representative from the area not yet well covered by R . When R is of the desired size, the ‘worst’ representative is deleted after a new representative is determined – the ‘worst’ representative being the one exhibiting the lowest representative ability. The representative ability of a representative – say r_j – can be measured using

$$RE(r_j; X) = \frac{1}{N \log(K)} \sum_{all\ x \in X} -p(r_j | x) \log(p(r_j | x)) \quad (4)$$

Given a data set X containing N patterns, REDR can be stated as follows:

Algorithm 2 Representative Entropy Data Reduction (REDR)

step 1. Randomly select a pattern set R_0 . Set the representative set R empty. Determine K , the desired size of R . Naturally, a representative represents L ($L = N/K$) patterns of X .

step 2.

repeat

In X , determine the top L patterns nearest to r_j ($r_j \in R_0$).

Based on the sum of the distances of r_j to these patterns, the most dense element of R_0 (say, r_d) is identified and placed into R .

The top L patterns nearest to r_d (including, of course, r_d itself) will be rejected in the subsequent forward search stage.

until R_0 is completely scanned or R contains K elements

step 3. In X , select out the patterns having $\max_{r_j \in RP_{outer}}(x | r_j) > 0$. Among these patterns, identify the one with $\min(\max_{r_j \in RP_{outer}}(x | r_j))$, and put it into R . $p_{outer}(x | r_j)$ is defined by:

$$p_{outer}(x | r_j) = \begin{cases} 1 - \frac{d(x, r_j)}{Rad_{outer}(r_j)}, & d(x, r_j) \leq Rad_{outer}(r_j) \\ 0, & otherwise \end{cases}$$

where $Rad_{outer}(r_j)$ is the distance of r_j with the $2L$ th pattern nearest to it.

step 4. When R consists of $K+1$ representatives, delete the worst representative – in other words, the one exhibiting the largest $RE(r_j, X)$.

step 5. Calculate $RE(R, X)$ for the newly constructed R .

step 6. Repeat Steps 3 through 5 until $RE(R, X)$ cannot be reduced for five consecutive iterations.

In the above process, the condition of $\max_{r_j \in RP_{outer}}(x | r_j) > 0$ in Step 3 guarantees a stable data reduction process. Apparently, patterns with $\max_{r_j \in RP_{outer}}(x | r_j) > 0$ must be the ones uncovered by R . Without *a priori* knowledge on the density distribution of these patterns, it is not

recommended to determine the representative from them. With the constraint of $\max_{r_j \in R} P_{outer}(x | r_j) > 0$, the newly determined representative must be around the boundary of the area that has already been covered by R . In this way, the proposed method can gradually and reliably explore the entire data space.

Given a data set comprising N patterns, the computational and memory requirement of the multi-scale method is $O(N^2)$. This means that the multi-scale method is computationally demanding when applied to a large data set. For REDR, the computational complexity is $O(N(n_i + k_0))$, where k_0 is the number of patterns initially selected into R , and n_i is the number of RE-based processes for iteratively adjusting R . Generally, $(n_i + k_0)$ is much less than N . REDR always requires substantially less computational overhead compared with the multi-scale method.

Data Reduction Method Comparison

In this Section, REDR, random sampling, SOM [18] and Mitra's multi-scale methods are compared on five data sets – three synthetic and two real, from the University of California at Irvine Knowledge Discovery in Databases repository (<http://kdd.ics.uci.edu/>) – as summarized in Table 1.

The comparisons are conducted from the perspectives of efficiency and effectiveness, with running time used to evaluate the former. For a data reduction method, 'effectiveness' means the extent that the original data distribution information is preserved after data reduction. This can be measured by the difference between the density functions estimated on the original data set and the reduced data set delivered by the tested method. Assume that we have the density function obtained on the original data – say $f(x)$ – and the one obtained on the reduced data – ($g(x)$). To evaluate the difference between $f(x)$ and $g(x)$, two indexes can be used. They are the absolute distance D_{ab} and the Kullback-Liebler distance (divergence) D_{KL} :

$$D_{ab}(f(x), g(x)) = \int_x |f(x) - g(x)| dx \quad (5)$$

and

$$D_{KL}(f(x), g(x)) = \int_x f(x) \log \frac{f(x)}{g(x)} dx \quad (6)$$

Also, we can have

$$D_{ab}(f(x), g(x)) \approx \sum_{tx_i \in TX} |f(tx_i) - g(tx_i)| \Delta tx_i, \quad (7)$$

and

$$D_{KL}(f(x), g(x)) \approx \sum_{tx_i \in TX} f(tx_i) \log \frac{f(tx_i)}{g(tx_i)} \Delta tx_i \quad (8)$$

Table 1. The data sets used for data reduction

Name of dataset	Number of training data patterns	Number of test data patterns	Number of features
Pen-based recognition of handwritten digits	5000	3498	16
Yeast data	800	684	8
Synthetic Data 1	2000 points from $N\left([0, 0], \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$		
Synthetic Data 2	1000 points from $N\left([0, 0], \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}\right)$		
	1000 points from $N\left([0.3, 0.3], \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$		
	1000 points from $N\left([-0.3, -0.3], \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$		
Synthetic Data 3	1000 points from $N\left([0, 0], \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$		
	1000 points from $N\left([0.3, 0.3], \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}\right)$		
	1000 points from $N\left([-0.3, -0.3], \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$		
	1000 points from $N\left([0.1, 0.1], \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix}\right)$		

when the data set TX is large enough and can cover most of the area with $f(x) > 0$ or $g(x) > 0$. A smaller D_{ab} or D_{KL} means a better data reduction result.

The comparative results are listed in Table 2 (data reduction effectiveness) and Table 3 (computational efficiency). In each cell of Table 2, the upper and lower values are the means and standard deviations of the results of 20 trials, respectively. RR or reduction ratio is the ratio of the reduced data set size to the original data set size. The results on different examples lead to similar conclusions. In terms of effectiveness, density-based methods – such as REDR and the multi-scale based method – significantly outperform SOM and random sampling. From the viewpoint of computational efficiency, SOM performs better than density-based methods – namely REDR and the multi-scale

Table 2. Results of data reduction methods in terms of density difference

Data set	Random sampling		SOM		Multiscale-based method			REDR-DR	
	D_{ab}	D_{KL}	D_{ab}	D_{KL}	D_{ab}	D_{KL}	RR	D_{ab}	D_{KL}
<i>RR = 1/20</i>									
Synthetic	0.052	0.073	0.050	0.057	0.042	0.056	0.05	0.041	0.054
Data 1	0.0044	0.0079	0.0007	0.0017	0.00054	0.0011	0.002	0.0004	0.0006
Synthetic	0.052	0.073	0.05	0.057	0.042	0.056	0.05	0.041	0.054
Data 2	0.0044	0.0079	0.0007	0.0017	0.00054	0.0011	0.002	0.0004	0.0006
Synthetic	0.16	0.2	0.12	0.17	0.11	0.14	0.05	0.11	0.14
Data 3	0.021	0.022	0.002	0.004	0.001	0.001	0.003	0.002	0.002
Pen-based	1.41	0.91	1.39	0.82	1.35	0.86	0.05	1.34	0.81
handwriting ($\times 10^{-9}$)	0.054	0.031	0.035	0.02	0.031	0.009	0.005	0.01	0.005
Yeast	1.39	0.97	0.58	0.46	0.54	0.42	0.05	0.55	0.53
($\times 10^{-5}$)	0.066	0.046	0.034	0.019	0.03	0.02	0.008	0.017	0.036
<i>RR = 1/10</i>									
Synthetic	0.048	0.059	0.038	0.044	0.033	0.038	0.1	0.033	0.037
Data 1	0.0033	0.0056	0.0007	0.0008	0.0002	0.0006	0.02	0.0005	0.0005
Synthetic	0.11	0.13	0.08	0.12	0.075	0.089	0.1	0.076	0.088
Data 2	0.008	0.01	0.001	0.003	0.001	0.002	0.01	0.0008	0.0009
Synthetic	0.15	0.19	0.11	0.15	0.11	0.12	0.1	0.11	0.12
Data 3	0.02	0.03	0.001	0.003	0.003	0.004	0.002	0.002	0.001
Pen-based	1.01	0.7	0.81	0.65	0.55	0.48	0.1	0.53	0.47
handwriting ($\times 10^{-9}$)	0.05	0.028	0.05	0.052	0.12	0.005	0.03	0.01	0.01
Yeast	0.86	0.72	0.54	0.43	0.53	0.43	0.11	0.48	0.4
($\times 10^{-5}$)	0.046	0.039	0.063	0.046	0.018	0.019	0.03	0.017	0.013

based one – as suggested in Table 3. Moreover, REDR is much more efficient than the multi-scale based method, which is consistent with the preceding theoretical analysis.

3 Feature Selection

An appropriate reduction of a feature set can not only improve the efficiency and scalability of a pattern recognition procedure, in some cases, it also enhances the recognition accuracy because of the finite sample size.

Table 3. Results of data reduction methods in terms of running time (in seconds)

Dataset	SOM	Multiscale-based method	REDR-DR
<i>RR = 1/20</i>			
Synthetic1	5	12	4
Synthetic2	8	44	15
Synthetic3	11	102	40
Pen-based handwriting	51	1600	349
Yeast	2	7	3
<i>RR = 1/10</i>			
Synthetic1	7	30	14
Synthetic2	9	203	37
Synthetic3	12	217	79
Pen-based handwriting	85	3500	888
Yeast	3	14	7

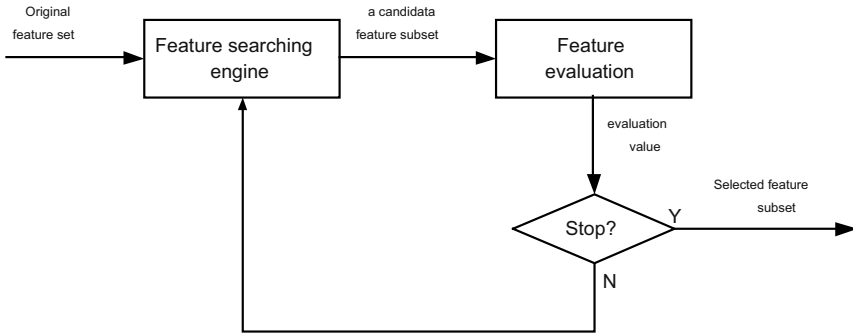


Fig. 1. Feature selection model general outline

Compared with a complex recognition model, a simple model is always preferable. These are the main reasons why feature selection has always been an essential consideration in statistics and neural computation.

Feature selection is an optimization process in the hypothesis space which includes all possible feature subsets. A general feature selection model is illustrated in Fig.1. Now both the feature evaluation criteria and the feature search engine are two crucial parts in any feature selection model. In the following Sections, the major feature selection techniques are surveyed according to these two aspects.

3.1 Feature Evaluation

Feature evaluation indices are designed to enumerate the relevancy of a feature subset to a recognition task or the redundancy among a feature subset [6, 29, 33]. According to the types of feature evaluation criteria, feature selection schemes are categorized into one of three groups – the *filter* scheme, the *embedded* scheme, and the *wrapper* scheme.

Wrapper Model

Wrapper feature selection models [29, 31] directly employ the accuracy of a certain recognition procedure to evaluate the quality of feature subsets. Due to this mechanism, it is generally argued that a wrapper scheme is able to provide better pattern recognition behavior compared with other types of feature selection models. For example, in [9, 29], comparative results show that a wrapper model performs better than a filter model for building a decision tree (DT).

On the other hand, wrapper models are usually criticized for being too computationally demanding [21, 29, 65]. This is the main shortcoming of wrapper models. Given each tested feature subset, a pattern recognition model is constructed. Then based on the performance of that model on validation data, or under a cross-validation scheme ([29] shows that the latter outperforms the former), the quality of the tested feature subset is evaluated. However, repeating this evaluation process on many feature subsets is computationally demanding. Another problem with wrapper models is overfitting. A pattern recognition procedure built in a high-dimensional domain needs to be complex. Most likely, this procedure will overfit the training data. Also, using a single data set many times over may increase the likelihood that the final selection results only perform well on the data used in the feature selection process [65].

Embedded Model

In an embedded model, the intermediate results of a pattern recognition learning algorithm rather than the final recognition results are employed to evaluate the importance of features. In practice, the rationale behind embedded feature selection methods is the idea of pruning. This idea is not novel in the neurocomputing community where it is well known that pruning unimportant components of a recognition model can avoid overfitting and bring computational and memory savings.

In [20, 44, 56, 61, 62], a pattern recognition model (such as Support Vector Machine [20, 62] or Multi-Layer Perceptron network [56, 61]) is firstly built using all the given features. Then, based on the model parameters, the

pattern recognition importance of each feature is estimated. Following elimination of unimportant features, another training iteration is performed and unimportant features are then eliminated.

Generally, an embedded method is more efficient than a wrapper method. However the basic problems of wrapper methods – namely the huge computational load and high likelihood of overfitting – cannot be instinctively addressed because constructing a recognition model in a high-dimensional domain is still inevitable in an embedded scheme.

Filter Model

In a filter model, feature selection is conducted before the learning phase of a recognition model. In other words, a ‘good’ feature subset is identified prior to construction of a recognition model. In such a way, the problems of huge computational complexity and overfitting can be circumvented.

Up to now, various statistical or distribution-based criteria [3, 12, 22, 23, 26, 28, 35, 36, 39, 42, 46, 50, 51, 60, 64, 65] have been developed for measuring feature relevance (in other words, the relationship between features and output variable(s)), and/or feature dependence (that is, the relationship between features). Some useful surveys on these criteria can be found in [42, 47]. In general, these filter criteria are categorized into several groups such as dependency, information, consistency, distance, and so forth. In this Chapter, we summarize these criteria from a different perspective. They are categorized into three main groups: *variable-similarity-based*, *probability-divergence-based*, and *pattern-distance-based*.

Variable-similarity-based criteria

Variable-similarity-based criteria employ or adopt various statistical metrics – for instance correlation coefficient and mutual information (MI) – to enumerate the similarity between feature(s) and output variable, or alternatively the association among features.

In [46], a correlation coefficient-based criterion is designed to estimate the similarity of two features. According to estimation results, similar features are identified and clustered together. Then, through selecting one representative feature from each feature cluster and discarding the others, the original feature set is reduced.

In [22], the correlation-based feature selection (CFS) index is defined as

$$CFS(S) = \frac{m\overline{r_{cf}}}{\sqrt{m + m(m-1)\overline{r_{ff}}}} \quad (9)$$

where S is the tested feature subset with cardinality m , $\overline{r_{cf}}$ is the average feature-class correlation of S , and $\overline{r_{ff}}$ is the average feature-feature correlation

of S . In $CFS(S)$, the numerator indicates the predictiveness of the feature set S , and the denominator measures the extent of redundancy of S . Obviously, in order to achieve a large value of $CFS(S)$, the pattern recognition ability of each feature in S should be high, and at the same time the redundancy in S is required to be low.

These correlation-based criteria are relatively efficient from a computational point of view. The concept of correlation, however, is not sophisticated enough to handle the complicated nature of a problem because correlation measures only the relationship between two variables, whereas feature selection schemes need to deal with *hundreds* (or even *thousands*) of variables. The feature clustering algorithm in [44] only considers the relationship between two features, and ignores the relationship of any more than two. Moreover, in $CFS(S)$, the multi-variable relationship is approximated using the linear averages of related 2-variable correlations, $\overline{r_{cf}}$ and $\overline{r_{ff}}$. This strategy is apparently not sufficiently sophisticated to measure complex relationships between variables.

Mutual information (MI) is an effective alternative for evaluating the relationship between variables. Given two variables X and Y , MI can evaluate the similarity between these two variables. Shannon's MI is defined as

$$I(X, Y) = \int_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx dy. \quad (10)$$

Compared with the correlation coefficient, it is more flexible in the sense that MI does not require X and Y to be of the same dimensionality, moreover MI can reflect the arbitrary relationship between X and Y . The main challenge of MI-based indices lies in the computation of MI as elaborated in Eqn. (10). It shows that the estimation of MI requires the probabilities underlying X and Y to be estimated. Furthermore, we need to conduct a rather computationally demanding integration when continuous variables are present. This computational process is rather difficult under a high-dimensional domain.

To estimate the probabilities $p(x)$ and $p(x,y)$, histogram and Parzen window estimators are the most common approaches. Using histograms can simplify integration of computing MI as summation. However the problem of pattern shortage paralyzes histogram estimators when working in a high-dimensional space [14, 48]. On the other hand, Parzen window is considered more effective and reliable than histograms as a probability estimator. Nevertheless Parzen window may pose certain computational difficulties because of the integral procedure involved in estimating MI.

In the MI-based feature selection method (MIFS) [3] and the MI-based feature selection method with uniform modification (MIFS-U) [35], histogram estimators are used to estimate the probability density functions. To avoid the difficulties that histograms will encounter in high-dimensional spaces,

MIFS and MIFS-U do not *directly* estimate a high-dimensional MI. Instead, high-dimensional MIs are analyzed through a linear combination of related 2-dimensional MI estimates. This approach successfully overcomes the computational problems of MI, but generates other problems, these being (1) the complex relationship between features or between input and output may not be reflected correctly using linear equations; and (2) there are no principles (guidelines) to determine the most important parameter for controlling redundancy in selected features.

In [8, 11, 36], Parzen window estimators are used in MI-based evaluation criteria. Suppose that a pattern in X belongs to one of L classes. The L classes are represented by $\omega_1, \omega_2, \dots, \omega_L$. In [36], the MI-based criterion is defined as

$$I(X; C) = H(C) + \int_x p(x) \sum_{k=1}^L p(\omega_k | x) \log(p(\omega_k | x)) dx \quad (11)$$

$$\approx H(C) + \frac{1}{|X|} \sum_{x \in X} \sum_{j=1}^L p(\omega_j | x) \log p(\omega_j | x) \quad (12)$$

The criterion developed in [54] is

$$I(X; C) = \int \sum_{k=1}^L p(x, \omega_k) \log \frac{p(x, \omega_k)}{p(x)p(\omega_k)} dx = E_{x, \omega_k} \left[\log \frac{p(x, \omega_k)}{p(x)p(\omega_k)} \right] \quad (13)$$

$$\approx \frac{1}{|X|} \sum_{x \in X} \log \frac{p(x, \omega_k)}{p(x)p(\omega_k)} = MI - raw \quad (14)$$

where $|X|$ represents the cardinality of X .

In these criteria, the integration procedure for estimating MI is approximated by a summation. This is a crude way of solving the problem because this approximation may not be reliable when X is not large enough. In [10], the MI-based criterion is modelled as:

$$QMI = I_{CS}(X; C) = \log \frac{\sum_{k=1}^L \int p(x, \omega_k)^2 dx \sum_{k=1}^L P(\omega_k)^2 \int p(x)^2 dx}{\sum_{k=1}^L \int p(\omega_k, x) P(\omega_k) p(x) dx} \quad (15)$$

Probability divergence-based criteria

Probability divergence-based criteria explore the conditional probabilities of different classes to determine the ‘goodness’ of a feature subset. The rationale behind these criteria is that a feature subset with high discriminant capability must guarantee a large distance between the conditional probabilities

of different classes [12, 26, 34]. Given two classes ω_1 and ω_2 , there are many formats to evaluate their divergence [12, 34], such as Bhattacharryya divergence (Eqn. (16))

$$J_B(\omega_1, \omega_2) = \ln \int_x (p(x | \omega_1)p(x | \omega_2))^{1/2} dx \tag{16}$$

and Kullback-Liebler divergence (Eqn. (17)).

$$J_M(\omega_1, \omega_2) = \int_x (p(x | \omega_1) - p(x | \omega_2))(\ln(p(x | \omega_1)) - \ln(p(x | \omega_2))) dx \tag{17}$$

The 2-class divergence has been generalized to multi-class cases [12, 26] in a simple way as $J = \sum_{i=1}^L \sum_{j=1}^L P(\omega_i)P(\omega_j)J(\omega_i, \omega_j)$, where the measurement $J(\omega_i, \omega_j)$ can be in any format of 2-class divergence.

There is an important aspect of the divergence-based criteria [58]. When it is known (or assumed) that all the classes are normally distributed with the same covariance, that is

$$p(x | \omega_i) = \left((2\pi)^M |\Sigma| \right)^{-1/2} \exp \left(-\frac{1}{2} (x - \mu_i) \Sigma^{-1} (x - \mu_i)^T \right) \tag{18}$$

where μ_i is the mean of all x belonging to the class ω_i , Σ is the covariance matrix, and M is the dimensionality of x . The above divergences (Eqn. (16) and Eqn. (17)) can be simplified to the Mahalanobis distance

$$J_M = 8J_B = (\mu_1 - \mu_2) \Sigma^{-1} (\mu_1 - \mu_2)^T \tag{19}$$

which is a feature selection criterion first developed within the statistics community.

Consistency [42] is designed in a discrete or discretized data space. Suppose that there are J totally distinct patterns in X , with dx_i being the i th one. Also, for dx_i , the dominant class – dc_i – is the class in which dx_i appears most in X . *Consistency* of the feature subset S is defined as

$$\text{Consistency}(S) = 1 - \frac{\sum_{i=1}^J \text{inconsistency_count_of_} dx_i}{N} \tag{20}$$

$$= 1 - \frac{\sum_{i=1}^J (n_i^{all} - n_i^{mj})}{Nx} \tag{21}$$

where N is the size of X , n_i^{all} is the number of dx_i occurrences in X , and n_i^{mj} is the number of dx_i occurring in its dominant class dc_i . In terms of

probability, the consistency (Eqn. (20)) can be explained in a clear way. It is known that n_i^{all}/N and n_i^{mj}/N are the probability $p(dx_i)$ and the joint probability $p(dx_i, dc_i)$, respectively. The consistency (Eqn. (21)) is in fact the divergence of $p(dx_i)$ to $p(dx_i, dc_i)$ over a given domain.

More recently, a Bayesian discriminant feature selection criterion has been proposed, in which the distance between conditional *posterior* probabilities of different classes is used to measure the classification capability of feature subsets [26]. Given a feature set- S , the Bayesian discriminant feature selection criterion is defined as

$$BDFS(S) = \frac{1}{|X|} \sum_{x \in X} \log \frac{P_S(\omega = c | x)}{P_S(\omega \neq c | x)} \quad (22)$$

A large $BDFS(S)$ means a low likelihood of x being incorrectly recognized.

Now, despite the fact that the divergence-based-criteria and MI-based ones are derived from different viewpoints, they are very similar in format, and encounter the same computational difficulties.

Pattern distance-based criteria

Pattern distance-based criteria exploit data distribution information without explicitly estimating the underlying probabilities [39, 47, 51, 60, 64]. In order to accurately distinguish patterns of different classes, it is natural to require that a pattern surrounded by ones from the same class, or that patterns belonging to different classes, are far from each other. This is the basis of pattern distance-based criteria, including interclass distance [47] and similarity index [39]. The interclass distance-based criterion is defined as

$$J_{inter-class} = \sum_{i=1}^L P(\omega_i) \sum_{j=i+1}^L P(\omega_j) D(\omega_i, \omega_j) \quad (23)$$

where $D(\omega_i, \omega_j)$ – the distance of two classes ω_i and ω_j – can be measured by way of

$$D(\omega_i, \omega_j) = \frac{1}{|class - \omega_i|} \frac{1}{|class - \omega_j|} \sum_{x_k \in class - \omega_i} \sum_{x_q \in class - \omega_j} d(x_q, x_k) \quad (24)$$

A large value of $J_{inter-class}$ is preferred to discriminate the patterns in different classes. In [44], the similarity between patterns is firstly measured using the distance of patterns. Then, based on the similarity estimates, a feature selection index is derived, which decreases as the interclass(intraclass) distances increase/decrease. In other words, a low value of this index indicates a clear separation of the classes.

In Q- α -FS [64], the inner products of patterns – another type of pattern distance measurement – are employed to evaluate a feature subset in terms

of cluster coherence of the first biggest k clusters. A high value of that index can indirectly indicate that the original dataset is ‘well-clustered’. Through optimizing it, the contribution of a feature to clearly cluster patterns is evaluated. Based on these evaluation results, the appropriate features are finally determined.

Fuzzy feature evaluation index (FFEI) [51] evaluates the relationship between the distance values measured in the original input space and a reduced input space. An appropriate reduction of input space should not distort the original distribution. In other words, it is expected that patterns close to (far from) each other in the original data space should be close to (far from) each other in the reduced data space. This is the basic idea of FFEI.

3.2 Search Engine

Search engine in a discrete feature domain has been an active research area. Generally, a search engine is seen from two perspectives – search direction and search strategy [43, 47]. In this Chapter, we consider search direction and search strategy as a single issue for ease of understanding. Search engines can be grouped into four categories: an optimal search engine, a heuristic search engine, a stochastic search engine, and a weighting-based search engine.

Optimal Search Engine

Optimal search engine can deliver optimal theoretical results. Optimal search engines are often referred as complete (or exhaustive) search, or best-first search [42, 50, 66]. As all possible feature subsets are tested and compared, a complete search is able to identify the best feature subset(s). However with a computation complexity of $O(2^M)$ (M being the cardinality of the original feature set), it is practically impossible to apply the complete search procedure in most real world applications.

The branch-and-bound algorithm [42, 50] is another optimal search scheme. Depending on a monotonic feature evaluation criterion, this algorithm can implicitly inspect all possible feature subsets without conducting a complete search. Through reducing the feature search domain, the branch-and-bound algorithm can offer computational savings compared with the complete search scheme. However this improvement is not enough. The branch-and-bound algorithm is not computationally feasible even when operating on a moderate size feature set. In [42], this algorithm is only applied to examples with less than 20 features.

Heuristic Search Engines

Heuristic search engines are the most popular of the four types of search engine. By avoiding expensive searching of the entire feature space, a heuristic

search scheme can greatly reduce computation burden. As a tradeoff, this type of search engine may not be able to deliver optimal selection results. However, it is alleged that floating search – a type of heuristic search scheme – is able to deliver results comparable to optimal search [54].

Typical heuristic search schemes are sequential forward search, sequential backward search and floating (compound) search [12]. As its name suggests, in the sequential forward (backward) search scheme, the important (unimportant) features are iteratively identified and added into (or eliminated from) the selected feature subset. The cost of these search methods is $O(M^k)$, where M and k are the number of given features and the number of features identified in a single step, respectively (generally, k is set to 1). As for floating search, its basic idea is simple: we apply a forward steps followed by b backward steps. The cost of this type of bidirectional search scheme is $O(M^{a+b+1})$.

Stochastic Search Engines

Stochastic (random or non-deterministic) search engines are based on a stochastic algorithm, such as a genetic algorithm (GA) or the anytime algorithm [37, 57, 68]. The basic idea of these algorithms is that the next tested object (a feature subset in the feature search domain) is generated in a relatively random way, in order not to stick to a local optimization result. Unlike heuristic search, a stochastic search engine does *not* reduce the original search space. However, the actual number of tested feature subsets can be greatly reduced by setting a reasonable search termination criterion.

Weighting-Based Search Engines

Weighting-based search engines [39, 64] firstly associate each original feature with a weight in a certain feature evaluation criterion. The value of this weight reflects the recognition contribution of the corresponding feature. Through using this weighting operation, feature selection can be fulfilled by employing an optimization algorithm in the continuous domain. In search engines of this type, an efficient and effective optimization process in the continuous domain is crucial.

3.3 Example Feature Selection Models

Several typical feature selection methods are considered in this Section to demonstrate their performance. These are Battiti's MI-based feature selection model (MIFS) [3], the MI raw-based feature selection model (MI-raw-FS) [8], the quadratic MI-based feature selection method (QMIFS) [11], the Bayesian Discriminant-based Feature Selection scheme [26], support vector machine based recursive feature elimination (SVM-RFE) [20], and Q- α -FS [64].

The first four methods are filter models in which a one-step forward search strategy is adopted. During each iteration, the one-step forward search evaluates the importance of each unselected feature and places the most important one into a selected feature subset. In MIFS, the importance of a feature (say f) is determined by

$$I(f; C | S) = I(f; C) - \beta \sum_{f \in S} I(f; f) \quad (25)$$

where S is the set of selected features, and C represents the output variable. The MI-raw-FS and QMIFS models employ MI-raw (Eqn. (14)) and QMI (Eqn. (15)) respectively to measure feature importance, while BDFS uses Eqn. (22) for this purpose.

The SVM-RFE [20] is a typical and efficient embedded model. It firstly builds an SVM model based on all features. According to the SVM model parameters, certain unimportant features are eliminated. Based on the remaining features, a new SVM model is built, thereby certain unimportant features are detected and deleted. The given feature set is reduced gradually in such a way. Q- α -FS uses the distribution of patterns to determine which features lead to improved clustering.

To evaluate the quality of selected feature subsets, four classifiers are employed. They are a neural network (NN) classifier, support vector machine model (SVM) with linear kernel, decision tree (DT), and k -Nearest Neighbour rule with $k = 1$. The above feature selection methodologies are applied to four datasets – two from the University of California, Irvine Knowledge Discovery in Databases repository (<http://kdd.ics.uci.edu/>), and two being cancer diagnosis data sets, as summarized in Table 4. The results are briefly presented in Tables 5 through 8. MIFS shows less effectiveness than the other five, which can be attributed to the fact that MIFS does not *directly* estimate the usefulness of a feature subset. This is illustrated in Eqn. (25). Moreover, in contrast to the other methods, Q- α -FS is an *unsupervised* approach. In other words, the class information of patterns is not used. This is the reason why Q- α -FS is unable to consistently deliver satisfactory results.

Table 4. The data sets used for the feature selection study

Dataset name	Number of features	Number of classes	Number of training patterns	Number of test patterns
Sonar	60	2	104	104
Ionosphere	34	2	200	151
Colon tumor [61]	2,000	2	30	32
ALL-AML [62]	7,128	2	38	34

Table 5. Results on the sonar classification data set

Number of selected features		MIFS	MI-raw-FS	QMIFS	SVM-RFE	BDFE	Q- α -FS
<i>k</i> -NN	4	0.69	0.76	0.7	0.76	0.72	0.6
	8	0.76	0.8	0.83	0.85	0.82	0.78
	15	0.76	0.82	0.86	0.88	0.92	0.87
	60	0.83					
SVM	4	0.6	0.63	0.7	0.74	0.69	0.40
	8	0.69	0.71	0.69	0.75	0.74	0.64
	15	0.69	0.69	0.74	0.73	0.77	0.73
	60	0.75					
ANN	4	0.62	0.69	0.7	0.72	0.74	0.61
	8	0.69	0.73	0.77	0.8	0.81	0.63
	15	0.74	0.79	0.77	0.77	0.84	0.84
	60	0.84					
DT	4	0.65	0.69	0.65	0.65	0.65	0.4
	8	0.69	0.71	0.67	0.67	0.71	0.4
	15	0.71	0.72	0.7	0.71	0.76	0.75
	60	0.74					

Table 6. Results on the ionosphere classification data set

Number of selected features		MIFS	MI-raw-FS	QMIFS	SVM-RFE	BDFE	Q- α -FS
<i>k</i> -NN	3	0.69	0.76	0.7	0.76	0.72	0.6
	5	0.76	0.8	0.83	0.85	0.82	0.78
	9	0.76	0.82	0.86	0.88	0.92	0.87
	33	0.83					
SVM	3	0.6	0.63	0.7	0.74	0.69	0.40
	5	0.69	0.71	0.69	0.75	0.74	0.64
	9	0.69	0.69	0.74	0.73	0.77	0.73
	33	0.75					
ANN	3	0.62	0.69	0.7	0.72	0.74	0.61
	5	0.69	0.73	0.77	0.8	0.81	0.63
	9	0.74	0.79	0.77	0.77	0.84	0.84
	33	0.84					
DT	3	0.65	0.69	0.65	0.65	0.65	0.4
	5	0.69	0.71	0.67	0.67	0.71	0.4
	9	0.71	0.72	0.7	0.71	0.76	0.75
	33	0.74					

Table 7. Results on the cancer classification data set

Number of selected features		MIFS	MI-raw-FS	QMIFS	SVM-RFE	BDFE	Q- α -FS
<i>k</i> -NN	2	0.69	0.76	0.7	0.76	0.72	0.6
	4	0.76	0.8	0.83	0.85	0.82	0.78
	8	0.76	0.82	0.86	0.88	0.92	0.87
SVM	2	0.6	0.63	0.7	0.74	0.69	0.40
	4	0.69	0.71	0.69	0.75	0.74	0.64
	8	0.69	0.69	0.74	0.73	0.77	0.73
ANN	2	0.62	0.69	0.7	0.72	0.74	0.61
	4	0.69	0.73	0.77	0.8	0.81	0.63
	8	0.74	0.79	0.77	0.77	0.84	0.84
DT	2	0.65	0.69	0.65	0.65	0.65	0.4
	4	0.69	0.71	0.67	0.67	0.71	0.4
	8	0.71	0.72	0.7	0.71	0.76	0.75

Table 8. Comparative results on ALL-AML data sets in terms of classification accuracy

Number of selected features		MIFS	MI-raw-FS	QMIFS	SVM-RFE	BDFE	Q- α -FS
<i>k</i> -NN	2	0.72	0.94	0.79	0.91	0.76	0.71
	4	0.79	0.91	0.91	0.79	0.82	0.79
	8	0.82	0.82	0.97	0.82	0.91	0.76
SVM	2	0.72	0.88	0.76	0.88	0.79	0.5
	4	0.79	0.91	0.91	0.85	0.76	0.62
	8	0.59	0.59	0.59	0.88	0.79	0.62
ANN	2	0.73	0.76	0.79	0.88	0.79	0.62
	4	0.76	0.82	0.88	0.79	0.76	0.62
	8	0.79	0.88	0.94	0.79	0.85	0.62
DT	2	0.73	0.73	0.79	0.88	0.88	0.59
	4	0.73	0.73	0.79	0.88	0.88	0.59
	8	0.79	0.91	0.79	0.88	0.94	0.62

4 Trends and Challenges of Feature Selection and Data Reduction

In the previous Sections, we discussed data reduction and feature selection. Their merits and shortcomings have been detailed. With the advent of numerous techniques and the development of computers generally, data volumes continue to increase. A major issue is to find an effective method to handle

huge volume data sets. In this Section, we briefly describe different methods that attempt to address the challenge posed by huge data volumes.

Firstly, in many physical applications, a data set may have a huge feature set, while the number of patterns is relatively small. A typical example is microarray gene expression data sets. Microarray is a relatively recent biomedical technique enabling biomedical researchers to record expression levels of up to ten thousands of genes simultaneously [19,21]. However Microarray data sets usually contain only tens or hundreds of patterns, due to the difficulties of data collection. Similar small-pattern/huge-feature problems may also occur in some image-based object recognition examples [4].

With small pattern sets, the problem of overfitting should be considered in order to deliver reliable feature selection results. When embedded or wrapper feature selection models are employed, researchers are recommended to choose a recognition model exhibiting high regularization capability. For example, support vector machine and the penalized COX model have been used in [19, 21] to select important genes (namely, features), based on microarray gene expression data.

As to filter feature selection models, few strategies have been designed to date for improving their regularization ability. In [4], a bootstrap framework is employed to alleviate overfitting. In this framework, certain data subsets are randomly selected from a given data set. Mutual information of a feature to the output variable is estimated using each data subset. Using all the obtained estimates, the relevance of a feature is finally measured. This approach can alleviate the problem of overfitting. However its large computational requirement, which is arguably the main shortcoming of the approach, substantially restricts its application.

Secondly, there are applications where the number of patterns and number of features are both large. For example, with the development of computer techniques, a large quantity of patterns can be easily collected in the context of text mining, customer management, and web page analysis [38, 49, 59]. In these cases, researchers are required to identify useful features from a large pattern set. Feature selection has to work together with data reduction in order to enhance the efficiency of feature selection.

Broadly speaking, there are two ways of combining feature selection with data reduction. First, feature selection and data reduction can be integrated in a filter way. That is, we can firstly reduce a huge pattern set. Subsequently, feature selection is conducted based on the reduced pattern set. For example, in [63], a given large data set was firstly partitioned into several parts using a KD-Tree. From each part, representative patterns were selected. Useful features were then detected based on the representative patterns. But it must be noted that this mechanism requires an efficient data reduction technique.

Second, feature selection and data reduction can be fused in an embedded way. This embedded approach generally begins with a random selection of

a set of representative patterns. Using the selected representatives, certain useful features are selected. The representative set will continually be updated according to the performance of patterns on the selected features. In such a way, feature selection and data reduction are conducted in turn and iteratively until the result cannot be further enhanced. Generally, compared with filter schemes, embedded methods are less efficient, but more effective because the reduced data sets are actively adapted in the course of feature selection.

References

1. Alon U, Barkar N, Notterman DA, Gish K, Ybarra S, Mack D, Levine AJ (1996) Broad pattern of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. National Academy Science*, 96(12): 6745–6750.
2. Astrahan MM (1970) Speech analysis by clustering, or the hyperphoneme method. *Stanford AI Project Memo*, Stanford University, CA.
3. Battiti R (1994) Using mutual information for selecting features in supervised neural net learning. *IEEE Trans. Neural Networks*, 5: 537–550.
4. Bins J, Draper B (2001) Feature selection from huge feature sets. In: *Proc. Intl. Conf. Computer Vision*, July, Vancouver, Canada: 159–165.
5. Bishop CM (1995) *Neural Networks for Pattern Recognition*. Oxford University Press, New York, NY.
6. Blum AL, Langley P (1993) Selecting concise training sets from clean data. *IEEE Trans. Neural Networks*, 4(2): 305–318.
7. Blum AL, Langley P (1997) Selection of relevant feature and examples in machine learning. *Artificial Intelligence*, 97(1–2): 245–271.
8. Bonnländer B (1996) Nonparametric selection of input variables for connectionist learning. *PhD Thesis*, Department of Computer Science, University of Colorado at Boulder, CU-CS-812-96.
9. Carunana RA, Freitag D (1994) Greedy attribute selection. In: Cohen WW, Hirsh H (eds) *Proc. 11th Intl. Conf. Machine Learning*, New Brunswick, NJ, July. Morgan Kaufmann, San Francisco, CA: 28–36.
10. Catlett J (1991) Megaindiction: machine learning on very large databases. *PhD Thesis*, Department of Computer Science, University of Sydney, Australia.
11. Chow TWS, Huang D (2005) Estimating optimal feature subsets using efficient estimation of high-dimensional mutual information. *IEEE Trans. Neural Networks*, 16(1): 213–224.
12. Devijver PA, Kittler J (1982) *Pattern Recognition: a Statistical Approach*. Prentice Hall, Englewood Cliffs, NJ.
13. Duda RO, Hart PE, Stork DG (2001) *Pattern Classification*. Wiley, New York, NY.
14. Fraser AM, Swinney HL (1986) Independent coordinates for strange attractors from mutual information. *Physics Reviews A*, 33(2): 1134–1140.
15. Freund Y, Seung H, Shamir E, Tishby N (1997) Selective sampling using the query by committee algorithm. *Machine Learning*, 28: 133–168.
16. Friedman JH (1997) Data mining and statistics: what’s the connection? In: Scott DW (ed) *Proc. 29th Symp. Interface Between Computer Science and*

- Statistics*, Houston, TX, May (available online at <http://www.stat.stanford.edu/jhf/ftp/dm-stats.ps> – last accessed March 2007).
17. Golub TR, Slonim DK, Tamayo P, Huard C, Gassenbeck M, Mesirov JP, Coller H, Loh L, Downing JR, Caligiuri MA, Bloomfield CD, Lander ES (1999) Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286: 531–537.
 18. Gray RM (1984) Vector quantization. *IEEE ASSP Magazine*, 1(2): 4–29.
 19. Gui J, Li H (2005) Penalized Cox regression analysis in the high-dimensional and low sample size settings, with application to microarray gene expression. *Bioinformatics*, 21(13): 3001–3008.
 20. Guyon I, Weston J, Barnhill S (2002) Gene selection for cancer classification using support vector machines. *Machine Learning*, 46: 389–422.
 21. Guyon I, Elisseeff (2003) An introduction to variable and feature selection. *J. Machine Learning Research*, 3: 1157–1183.
 22. Hall MA (1999) Correlation-based feature selection for machine learning. *PhD Thesis*, Department of Computer Science, University of Waikato, New Zealand.
 23. Hall MA, Holmes G (2000) Benchmarking attribute selection techniques for data mining. *Working Paper 00/10*, Department of Computer Science, University of Waikato, New Zealand (available online at <http://citeseer.ist.psu.edu/382752.html> – last accessed March 2007).
 24. Han JW, Kamber M (2001) *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA.
 25. Hart PE (1968) The condensed nearest neighbour rule. *IEEE Trans. Information Theory*, 14: 515–516.
 26. Huang D, Chow TWS (2005) Efficiently searching the important input variables using Bayesian discriminant. *IEEE Trans. Circuits and Systems – Part I*, 52(4): 785–793.
 27. Huang D, Chow TWS (2006) Enhancing density-based data reduction using entropy. *Neural Computation*, 18: 470–495.
 28. Jain AK, Zongker D (1997) Feature selection: evaluation, application, and small sample performance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(2): 153–158.
 29. John GH, Kohavi R, Pfleger K (1994) Irrelevant features and the subset selection problem. In: Cohen WW, Hirsh H (eds) *Proc. 11th Intl. Conf. Machine Learning*, New Brunswick, NJ, July. Morgan Kaufmann, San Francisco, CA: 121–129.
 30. John GH, Langley P (1996) Statistics vs. dynamics sampling for data mining. In: Simoudis E, Han J, Fayyad UM (eds) *Proc. 2nd Intl. Conf. Knowledge Discovery and Data Mining*, Portland, OR, August. AAAI Press, Menlo Park, CA: 367–370.
 31. Kohavi R, John GH (1998) The wrapper approach. In: Liu H, Motoda H (eds) *Feature Extraction, Construction and Selection*. Kluwer Academic Publishers, New York, NY: 33–50.
 32. Kohonen T (2001) *Self-Organizing Maps*. Springer-Verlag, London, UK.
 33. Kudo M, Sklansky (1997) A comparative evaluation of medium and large-scale feature selectors for pattern classifiers. In: Pudil P, Novovicova J, Grim J (eds) *Proc. 1st Intl. Workshop Statistical Techniques in Pattern Recognition*, Prague, Czech Republic, June: 91–96.
 34. Kudo M, Sklansky J (2000) Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition*, 33: 25–41.
 35. Kwak N, Choi C-H (2002) Input feature selection for classification problems. *IEEE Trans. Neural Networks*, 13: 143–159.

36. Kwak N, Choi C-H (2002) Input feature selection by mutual information based on Parzen window. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(12): 1667–1671.
37. Last M, Kandel A, Maimon O, Eberbach E (2000) Anytime algorithm for feature selection. In: Ziarko W, Yao Y (eds) *Rough Sets and Current Trends in Computing* (Proc. 2nd Intl. Conf. RSCTC), October, Banff, Canada. Springer-Verlag, London, UK: 16–19.
38. Law M, Figueiredo M, Jain A (2002) Feature saliency in unsupervised learning. *Technical Report*, Department of Computer Science, Michigan State University (available at <http://www.cse.msu.edu/~lawhiu/papers/TR02.ps.gz> – last accessed March 2007).
39. Lazzerini B, Marcelloni F (2001) Feature selection based on similarity. *Electronics Letters*, 38(3): 121–122.
40. Lewis DD, Catlett J (1994) Heterogeneous uncertainty: sampling estimation of error reduction. In: Cohen WW, Hirsh H (eds) *Proc. 11th Intl. Conf. Machine Learning*, New Brunswick, NJ, July. Morgan Kaufmann, San Francisco, CA: 148–156.
41. Liu H, Motoda H (1998) *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, London, UK.
42. Liu H, Motoda H, Dash M (1998) A monotonic measure for optimal feature selection. In: Nedellec C, Rouveiral C (eds) *Proc. European Conf. Machine Learning*, Chemnitz, Germany, April. Springer-Verlag, London, UK: 101–106.
43. Liu H, Motoda H, Yu L (2002) Feature selection with selective sampling. In: Sammut C, Hoffmann A (eds) *Proc. 9th Intl. Conf. Machine Learning*, Sydney, Australia, July. Morgan Kaufmann, San Francisco, CA: 395–402.
44. MacKay D (1992) A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4: 448–472.
45. Mitra P, Murthy CA, Pal SK (2002) Density-based multi-scale data condensation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(6): 734–747.
46. Mitra P, Murthy CA, Pal SK (2002) Unsupervised feature selection using feature similarity. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(3): 301–312.
47. Molina LC, Belanche L, Nebot A (2002) Feature selection algorithms: a survey and experimental evaluation. *Technical Report*, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.
48. Moon Y, Rajagopalan B, Lall U (1995) Estimation of mutual information using kernel density estimators. *Physics Reviews E*, 52: 2318–2321.
49. Moore J, Han E, Boley D, Gini M, Gross R, Hastings K, Karypis G, Kumar V, Mobasher B (1997) Web page categorization and feature selection using association rule and principal component clustering. *Proc. 7th Intl. Workshop Information Technologies and Systems*, Atlanta, GA, December (available online at <http://citeseer.ist.psu.edu/15436.html> – last accessed March 2007)
50. Narendra PM, Fukunaga K (1997) A branch and bound algorithm for feature subset selection. *IEEE Trans. Computers – C*, 26(9): 917–922.
51. Pal SK, De RK, Basak J (2000) Unsupervised feature evaluation: a neuro-fuzzy approach. *IEEE Trans. Neural Networks*, 11(2): 366–376.
52. Plutowski M, White H (1993) Selecting concise training sets from clean data. *IEEE Trans. Neural Networks*, 4(2): 305–318.

53. Provost F, Kolluri V (1999) A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 2: 131–169.
54. Pudil P, Novovicova J, Kittler J (1994) Floating search methods in feature selection. *Pattern Recognition Letters*, 15: 1119–1125.
55. Roy N, McCallum A (2001) Toward optimal active learning through sampling estimation of error reduction. In: Lapalme KG (eds) *Proc. 18th Intl. Conf. Machine Learning*, Williamstown, MA, June. Morgan Kaufman, San Francisco, CA: 441–448.
56. Setiono R, Liu H (1997) Neural network feature selector. *IEEE Trans. Neural Networks*, 8(3): 654–661.
57. Siedlecki W, Sklansky J (1989) A note on genetic algorithms for large scale on feature selection. *Pattern Recognition Letters*, 10: 335–347.
58. Theodoridis S, Koutroumbas K (1998) *Pattern Recognition*. Academic Press, London, UK.
59. Tong S, Koller D (2000) Support vector machine active learning with applications to text classification. In: Langley P (ed) *Proc. 17th Intl. Conf. Machine Learning*, Stanford, CA, June. Morgan Kaufmann, San Francisco, CA: 999–1006.
60. Wang H, Bell D, Murtagh F (1999) Axiomatic approach to feature subset selection based on relevance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(3): 271–277.
61. Wang W, Jones P, Patridge D (2001) A comparative study of feature-salience ranking techniques. *Neural Computation*, 13: 1603–1623.
62. Weston J, Mukherjee S, Chapelle O, Pontil M, Poggio T, Vapnik V (2001) Feature selection for SVMs. In: Solla SA, Leen TK, Muller K-R (eds) *Advances in Neural Information Processing Systems 13*. MIT Press, Cambridge, MA: 688–674.
63. Wilson AL, Martinez TR (2000) Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38: 257–286.
64. Wolf L, Shashua A (2003) Feature selection for unsupervised and supervised inference: the emergence of sparsity in a weighted-based approach. *Technical Report 2003-58*, June, Hebrew University, Israel.
65. Xing EP, Jordan MI, Karp RM (2001) Feature selection for high-dimensional genomic microarray data. In: Brodley CE, Danyluk AP (eds) *Proc. 18th Intl. Conf. Machine Learning*, Boston, MA, June. Morgan Kaufman, San Francisco, CA.
66. Xu L, Yan P, Chang T (1998) Best first strategy for feature selection. *Proc. 9th Intl. Conf. Pattern Recognition*, Rome, Italy, November. IEEE Computer Society Press, Piscataway, NJ: 706–708.
67. Yang J, Honavar VG (1998) Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems*, 13(2): 44–49.
68. Yang ZP, Zwolinski (2001) Mutual information theory for adaptive mixture models. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(4): 396–403.

Resources

1 Key Books

Devijver PA, Kittler J (1982) *Pattern Recognition: a Statistical Approach*. Prentice Hall, Englewood Cliffs, NJ.

Liu H, Motoda H (1998) *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, London, UK.

2 Key Survey/Review Articles

Blum AL, Langley P (1997) Selection of relevant feature and examples in machine learning. *Artificial Intelligence*, 97(1–2): 245–271.

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. *J. Machine Learning Research*, 3: 1157–1183.

Jain AK, Zongker D (1997) Feature selection: evaluation, application, and small sample performance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(2): 153–158.

Kudo M, Sklansky J (1997) A comparative evaluation of medium and large-scale feature selectors for pattern classifiers. In: Pudil P, Novovicova J, Grim J (eds) *Proc. 1st Intl. Workshop Statistical Techniques in Pattern Recognition*, Prague, Czech Republic, June: 91–96.

Kudo M, Sklansky J (2000) Comparison of algorithms that select feature for pattern classifiers. *Pattern Recognition*, 33: 25–41.

Molina LC, Belanche L, Nebot A (2002) Feature selection algorithms: a survey and experimental evaluation. *Technical Report*, Departament de Llenguatges i Sistemes Informtics, Universitat Politécnic de Catalunya (available at: <http://www.lsi.upc.es/dept/techreps/html/R02-62.html> – last accessed March 2007)

Provost F, Kolluri V (1999) A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 2: 131–169.

3 Organizations, Societies, Special Interest Groups

ACM Special Interest Group on Knowledge Discovery and Data Mining
<http://www.acm.org/sigs/sigkdd/>

China Data Mining Research
<http://www.dmresearch.net/>

4 Research Groups

Data Mining Research Group of Jiawei Han
<http://dm1.cs.wiuc.edu/>

Research Group of Huan Liu
<http://www.public.asu.edu/~huanliu/index.html>

WEKA Machine Learning Project
<http://www.cs.waikato.ac.nz/~ml/>

5 Discussion Groups, Forums

Data Miner
<http://blogger.org.cn/blog/blog.asp?name=idmer>

KDnuggets
<http://www.kdnuggets.com/>

6 Key International Conferences/Workshops

IEEE International Conference on Data Mining (ICDM)
<http://www.comp.hkbu.edu.hk/~wii06/icdm/>

ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining
<http://www.kdd2006.com/>

7 (Open Source) Software

WEKA (Machine Learning algorithms, including feature selection)

<http://www.cs.waikato.ac.nz/~ml/>

8 Data Bases

University of California, Irvine Machine Learning Repository

<http://www.ics.uci.edu/~mlearn/MLRepository.html>

Kent Ridge Biomedical Repository

<http://research.i2r.a-star.edu.sg/rp/>

Broad institute Bioinformatic Data

<http://www.broad.mit.edu/tools/data.html>

Topographic Maps for Clustering and Data Visualization

Colin Fyfe

Applied Computational Intelligent Research Unit, The University of Paisley,
Scotland, UK, colin.fyfe@paisley.ac.uk

1 Introduction

Topographic maps (also known as topology-preserving mappings) are projections of a data set which attempt to capture some underlying structure therein. These are essentially unsupervised mappings (though supervised versions do exist), and so the algorithms must be structured in some way so that the final projection reveals some underlying structure in the data.

The self-organizing map (SOM) of Kohonen [21] is the oldest of such mappings and remains one of the most popular exploratory data analysis tools: a biennial conference is one of the most interesting and respected conferences on the academic circuit; a recent one – in Paris 2005 – attracted over 100 participants including most of the eminent scholars in this field (including Kohonen himself). The SOM is based on an artificial neural network methodology and attempts to mimic aspects of self-organization seen *in vivo*. There are many flavours of SOM but we cannot discuss them all here since it is a wide and still very active field of research, so will content ourselves with identifying some of the major trends in research in this area. We apologise in advance for any omissions we make to this review.

A more recent development is the Generative Topographic Mapping (GTM) developed by [1] in the late 1990s. This was a very active research area for a few years but the field seems to have lost some of its vitality recently. Some of this is no doubt due to the fact that the GTM is much more complex than the SOM and so researchers more interested in viewing their data sets rather than innovating in the field of topographic mappings have tended to use the SOM rather than the GTM. Also, the emphasis of GTM publications tended to be on the fact that it was a ‘principled alternative’ to the SOM. If researchers feel bound to stick to principled approaches, their research processes are limited in ways that do not happen in more application-oriented research. Also the quasi-religious Bayesian approach does not appeal to all researchers.

The rest of this Chapter is structured as follows: in Sect. 3, we discuss the basic competitive learning paradigm and the extension which leads to the SOM. In Sect. 4, we review the GTM and illustrate its use. Finally we review some of our recent work in this area.

2 Clustering and Visualization

These two topics may at first sight seem somewhat disparate yet they are closely linked. Clustering is used with high-dimensional data sets when we have no prior information about groups into which individual data items may be placed. This is in contrast to classification in which we have a set of training examples for which we already have prior knowledge of the classes to which the data belong. Learning to cluster data is an unsupervised learning problem, whereas classification is a supervised learning problem. Perhaps the most widely used clustering algorithm is the K -Means algorithm [4, 12, 23] which places K prototypes throughout a data space in such a way that each lies in the centre of a group of data; each data point may then be allocated a label which identifies it as lying in that cluster which is represented by the prototype. If the data is high dimensional, there is no quick and easy way for human intuition to see these clusters, but we can, through this process, understand that all points which are given the same label have some features which are similar to one another; by quantizing the data to the prototypes we have augmented our knowledge about the data.

However clustering does not give us any information about the *relationship* between clusters. One way to get this information is to view the labels above as lying in a feature space and then investigate the relationships between the labels in this feature space. Indeed, many techniques go beyond this: they first of all specify the relationships between the labels in the feature space and then perform the clustering while maintaining these relationships. The movements of the prototype has to be constrained in ways which accommodates these relationships. This, indeed, is the technique used by the Self-Organizing Map which is discussed in the next Section. The resulting clustering can then be used for visualization if the relationship between the labels is sufficiently low dimensional, that is, 1, 2 or 3 dimensional. Visualization is another technique for augmenting our knowledge about a data set: we are taking a high dimensional data set and labeling the members of the data set in such a way that we can visually see the relationship between different data points.

Of course the above discussion assumes that the labels can be forced to lie in a sufficiently low dimensional space, while simultaneously the prototypes associated with the labels can sufficiently accurately model the data in data space. Sometimes these two criteria conflict and so we achieve a visualization which does not accurately reflect all the features of the data. We may still hope that we have increased our knowledge of the data set though accepting

that there is more structure to be found in the data set which is beyond human visualization capabilities.

The era of computerization has enabled us to extend visualization in ways which have never previously been possible. Most visualization algorithms have, as their core, the basic idea that they are using the computer to grind out difficult (for a human) computation, while presenting the results in a way that is easy (for a human) to spot patterns. It is notoriously difficult to create generic pattern matching software yet humans (and presumably animals) find it very easy to find structure in visual data. Therefore we are using the computer to do what it is best doing (number crunching), while leaving the human to do what he/she is best doing (pattern matching). The remainder of this Chapter will consider a variety of ways of performing clustering and visualization.

3 The Self-Organizing Map

With Minsky and Papert's [24] famous book *Perceptrons*, research into artificial neural networks almost stopped for nearly two decades. Among the few exceptions to this was Kohonen, who developed a topographic mapping based on competitive learning. In the next Section we discuss competitive learning before reviewing the SOM itself [21].

3.1 Competitive Learning

One of the non-biological aspects of the basic Hebbian learning rule is that there is no limit to the amount of resources which may be given to a synapse. This is at odds with real neural growth in that it is believed that there is a limit on the number and efficiency of synapses per neuron. In other words, there comes a point during learning in which if one synapse is to be strengthened, another must be weakened. This is usually modelled as a competition for resources.

In competitive learning, there is a competition between the output neurons to fire. Such output neurons are often called 'winner-take-all' units. The aim of competitive learning is to cluster the data. However, as with the Hebbian learning networks, we provide no correct answer (that is, no labelling information) to the network. It must self-organise on the basis of the structure of the input data.

The basic mechanism of simple competitive learning is to find a winning unit and update its weights to make it more likely to win in future should a similar input be given to the network. We first have a competition between the output neurons and then

$$\Delta w_{ij} = \eta(x_j - w_{ij}), \quad \text{for the winning neuron } i \quad (1)$$

Note that the change in weights is a function of the *difference* between the weights and the input. This rule will move the weights of the winning neuron directly towards the input. If used over a distribution, the weights will tend to the mean value of the distribution since $\Delta w_{ij} \rightarrow 0 \iff w_{ij} \rightarrow E(x_j)$, where $E(\cdot)$ indicates the ensemble average.

Probably the three most important variations of competitive learning are

1. Learning Vector Quantisation [19]
2. The ART models [2, 3]
3. The Kohonen feature map [21]

The last of these is one of the subjects of this Chapter.

The Kohonen Feature Map

The interest in feature maps stems directly from their biological importance. A feature map uses the ‘physical layout’ of the output neurons to model some feature of the input space. In particular, if two inputs \mathbf{x}_1 and \mathbf{x}_2 are close together with respect to some distance measure in the input space, then if they cause output neurons y_a and y_b to fire respectively, y_a and y_b must be close together in some layout of the output neurons. Further, we can state that the opposite should hold: if y_a and y_b are close together in the output layer, then those inputs which cause y_a and y_b to fire should be close together in the input space. When these two conditions hold, we have a feature map. Such maps are also called *topology preserving maps*.

Examples of such maps in biology include:

- *the retinotopic map*, which takes input from the retina (at the eye) and maps it onto the visual cortex (back of the brain) in a two dimensional map,
- *somatosensory map*, which maps our touch centres on the skin to the somatosensory cortex,
- *the tonotopic map*, which maps the responses of our ears to the auditory cortex.

Each of these maps is believed to be determined genetically but refined by usage. For example, the retinotopic map is very different if one eye is excluded from seeing during particular periods of development.

Kohonen’s algorithm [21] is exceedingly simple – the network is a simple 2-layer network and competition takes place between the output neurons; however now not only are the weights into the winning neuron updated but also the weights into its neighbours. Kohonen defined a neighbourhood function $f(i, i^*)$ of the winning neuron i^* . The neighbourhood function is a function of the distance between i and i^* . A typical function is the Difference of Gaussians function; thus if unit i is at point \mathbf{r}_i in the output layer then

$$f(i, i^*) = a \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma^2}\right) - b \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma_1^2}\right) \quad (2)$$

where r_k is the position in neuron space of the k th centre: if the neuron space is 1-dimensional, $r_k = k$ is a typical choice; if the neuron space is 2-dimensional, $r_k = (x_k, y_k)$, its two dimensional Cartesian coordinates.

Results from an example experiment are shown in Fig. 1. The experiment consists of a neural network with two inputs and twenty five outputs. The two inputs at each iteration are drawn from a uniform distribution over the square from -1 to 1 in two directions. The algorithm is

Algorithm 1 Kohonen’s Self-Organizing Map Algorithm

repeat

1. Select at random an input point.
2. There is a competition among the output neurons; that neuron whose centres are closest to the input data point wins the competition:

$$\text{winning neuron, } i^* = \arg \min(\| \mathbf{x} - \mathbf{w}_i \|) \quad (3)$$

3. Now update all neurons’ centres using

$$\Delta w_{ij} = \alpha(x_j - w_{ij}) * f(i, i^*) \quad (4)$$

where

$$f(i, i^*) = a \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma^2}\right) - b \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma_1^2}\right) \quad (5)$$

until some termination criterion has been met

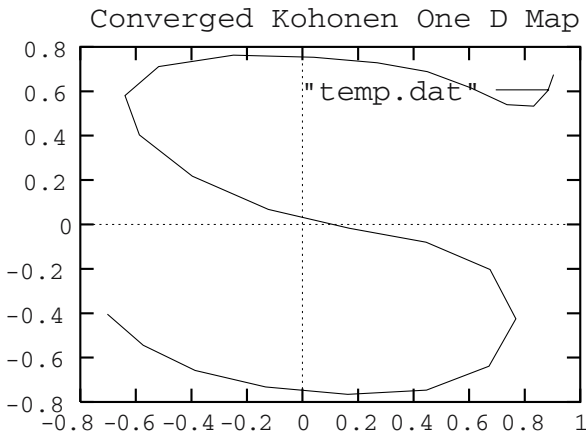


Fig. 1. A one-dimensional mapping of the two-dimensional input space

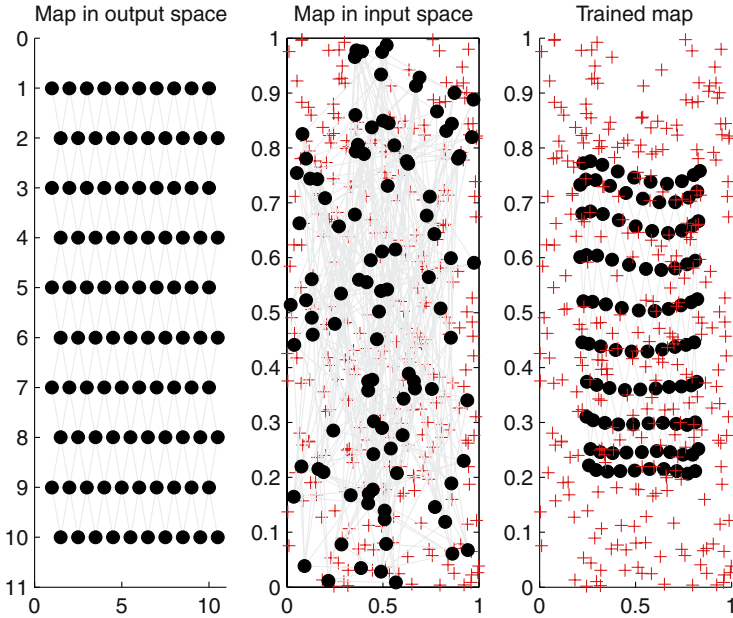


Fig. 2. *Left:* the neurons lie in a triangular grid in feature space; *centre:* the centres are initialised to lie randomly in data space; *right:* the centres in data space move into regular positions in the data space

Kohonen typically keeps the learning rate constant for the first 1000 iterations or so and then slowly decreases it to zero over the remainder of the experiment (we can be using 100,000 or more iterations for self-organising maps).

From the SOM Toolbox (<http://www.cis.hut.fi/projects/somtoolbox/>), we have Fig. 2, which illustrates the two positions which each centre may be thought to define: firstly each point determines a centre in data space but each point is also given a position in feature space which determines how close it is to the winning neuron (or any other neuron) in the feature space. The left diagram of Fig. 2 shows the positions of points in feature space. The middle diagram shows the initial positions of centres in data space while the right shows their positions in data space after training.

Note that, for visualization purposes, we ideally would like

1. points which are far distant from one another to be distant in the feature space,
2. points which are close to one another to be close in feature space,
3. points which are distant in feature space to be representing points distant in data space,
4. points which are close in feature space to be representing points close in data space.

Most existing topology preserving maps satisfy 1. and 4. but, as seen in Fig. 1, few satisfy 2. and 3. In that figure, this is caused by the fact that the data is 2-dimensional but the mapping used by this SOM network was 1-dimensional, and so the mapping has to coil round on itself to cover all of the data. This is not a problem for this data set since we can merely increase the dimensionality of the map to 2, however in general we are trying to visualise a high-dimensional data set with a two-dimensional map, and so we are making an implicit assumption that there exists a two-dimensional manifold which can adequately capture the main features of the data.

In [21], a whole chapter is given over to applications of the SOM; examples include text mining (specifically of web pages), a phonetic typewriter, clustering of financial variables, robot control and forecasting.

We have said that the most common topographic mappings are Kohonen's self-organizing map (SOM) [21] and varieties of multi-dimensional scaling [12]. The SOM was introduced as a data quantization method but has found at least as much use as a visualization tool. It does have the disadvantage that it retains the quantization element so that while its centres may lie on a manifold, the user must interpolate between the centres to infer the shape of the manifold. The Generative Topographic Mapping removes this necessity.

3.2 Illustrative Example

In this Chapter, we will use as a standard data set – a set of 118 samples from a scientific study of various forms of algae, some of which have been manually identified (<http://www.ics.uci.edu/~mlern/MLSummary.html>). Each sample is recorded as an 18-dimensional vector representing the magnitudes of various pigments. 72 samples have been identified as belonging to a specific class of algae which are labeled from '1' to '9'; 46 samples have yet to be classified and these are labeled '0'.

We show in Fig. 3 (top) a visualization of this data set from a two dimensional 10×10 SOM. Each data point is visualized as residing at the node on the map which would win the competition for that data point. However we can do rather better by defining the responsibility that the j^{th} node has for the i^{th} data point as

$$r_{ji} = \frac{\exp(-\gamma \|\mathbf{x}_i - \mathbf{w}_j\|^2)}{\sum_k \exp(-\gamma \|\mathbf{x}_i - \mathbf{w}_k\|^2)} \quad (6)$$

We then project points taking into account these responsibilities: let y_{ij} be the projection of the i^{th} data point onto the j^{th} dimension of the feature space, then

$$y_{ij} = \sum_k w_{kj} r_{ki} \quad (7)$$

We show this projection in Fig. 3 (bottom) for the same simulation which produced the top diagram.

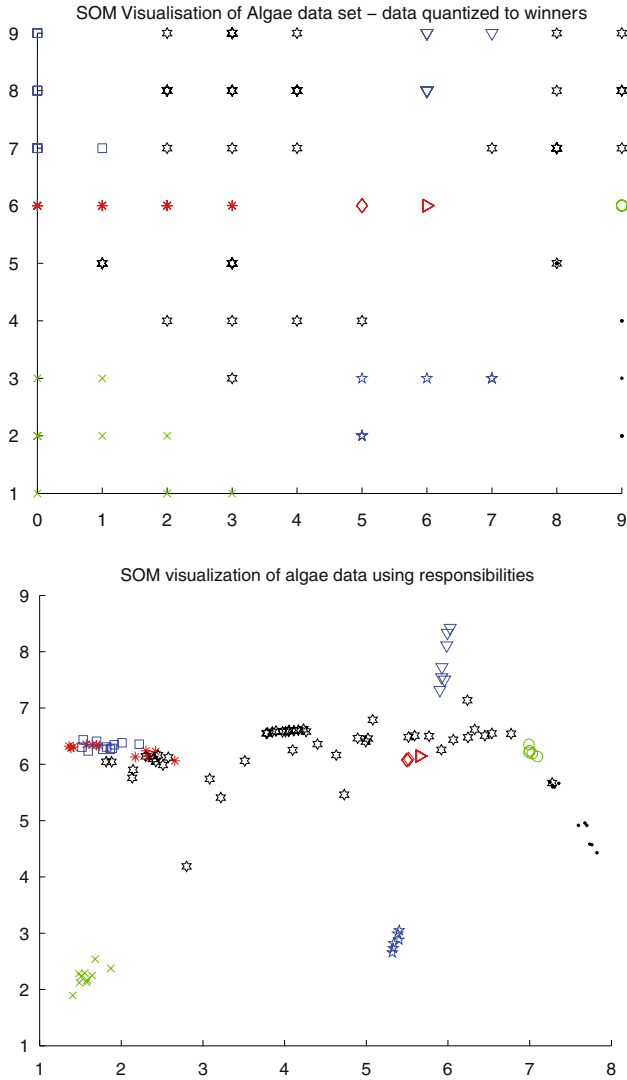


Fig. 3. *Top:* visualization of the algae data set when data points quantized to SOM nodes; *bottom:* visualization from the same simulation when we use responsibilities (see text)

3.3 Alternative Traditional Topology Preserving Mappings

Although the SOM is the most popular mapping which preserves some topology in the data set, it is by no means the only traditional mapping of this type. We mention only two alternatives in this Section. The first is multi-dimensional scaling which is designed for data sets about which we

only have distance information – in other words, we know the distances or differences between pairs of data points but have no actual positions with which to ground our mapping. Let d_{ij} be the distance between the i th and j th data point in data space and let δ_{ij} be the distance between the projections \mathbf{z}_i and \mathbf{z}_j of the data points in feature space. Then classical multi-dimensional scaling seeks to minimise the stress function

$$S = \sum_i \sum_{j \neq i} (d_{ij} - \delta_{ij})^2 \quad (8)$$

by altering the positions of \mathbf{z}_i and \mathbf{z}_j . Perhaps the most popular variant of multidimensional scaling is the Sammon mapping [12] which places more emphasis on preserving smaller distances with

$$S = \sum_i \sum_{j \neq i} \frac{(d_{ij} - \delta_{ij})^2}{d_{ij}} \quad (9)$$

An example of the Sammon mapping (which also used the implementation from the SOM Toolbox) on the algae dataset is shown in Fig. 4.

One of the disadvantages which a clustering which attempts to preserve the topography of a data sets can have is that the resultant map may have nodes which are in regions of the data space where there is no data. These are essentially dead neurons and have been pulled into their positions by the positioning of the neurons which act as its neighbours in feature space. An interesting variation on self-organising maps creates equi-probabilistic mappings, a mapping in which each node is equally likely to respond given the data set on which it is trained [27].

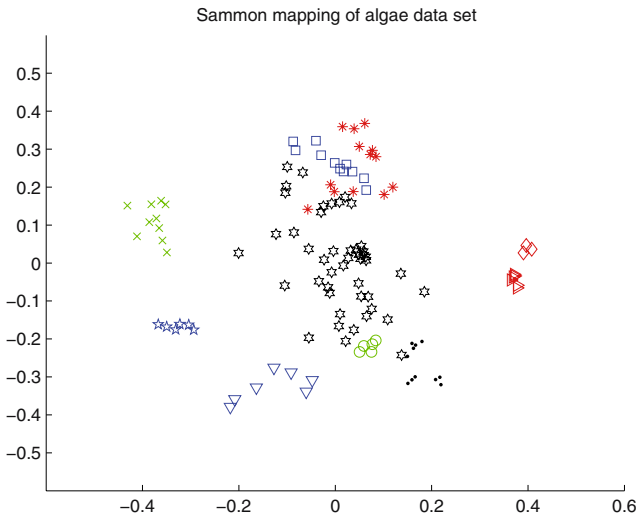


Fig. 4. A Sammon mapping of the algae data set

Kohonen himself [20] has created a mapping which visualizes an episode of data, a set of data which takes into account changes in the data over time. We may also consider combining the SOM with other visualization techniques, some of which are perhaps complementary, thereby providing us with methods which are more powerful than the individual techniques which have been combined [10].

Prior to the Generative Topographic Mapping (Sect. 4), several researchers investigated probabilistic mappings which have topology-preserving properties. Notable among these is Luttrell, who for a decade carried forward such investigations (see, for example, [22] for an early investigation).

Many researchers have investigated growing and pruning such maps. Perhaps the work of Fritzke [5–7] who investigated and compared such maps should be mentioned here.

3.4 A Last Word

We have mentioned the biennial conference on self-organizing maps; a recent one – in Paris 2005 – attracted over 100 participants and had a wide range of papers covering a variety of issues pertaining to topology preserving mappings (indeed the final sections of this Chapter are based on work presented at that conference). The next one (as I write) takes place in 2007 in Germany. It is anticipated that subsequent conferences will take place every two years after that.

Perhaps the most respected journal in the artificial neural network field is *Neural Computation* (MIT Press). Recently, a book containing some of the strongest articles on self-organizing map formation which have appeared in *Neural Computation* over the decade 1989–1999 has appeared [26]; it is essential reading for the serious researcher. There are also a number of other books on the topic.

A special edition of the journal *Neural Networks* has been devoted to current developments in self-organizing maps. Many of the leading researchers in this field have contributed articles to this issue and the resulting journal should be extremely interesting.

Other recent (as I write) developments include

- In August 2006, a research workshop was organized by Prof. A. Gorban, University of Leicester, UK, on ‘Principal manifolds for data cartography and dimension reduction’, which brought together researchers with interests in a number of related techniques.
- Another gathering of researchers in this field took place in March 2007 at the famous Schloss Dagstuhl, devoted to ‘similarity-based clustering’.

Finally, we should point interested readers to the Helsinki University of Technology website, which provides Matlab source code known as the SOM Toolbox (<http://www.cis.hut.fi/projects/somtoolbox>).

4 The Generative Topographic Mapping

In the next Section, we introduce two new topology preserving mappings the first of which we call the Topographic Products of Experts (ToPoE) and the second we call the Harmonic Topographic Map (HaToM). Based on a generative model of the experts, we show how a topology preserving mapping can be created from a product of experts in a manner very similar to that used by [1] to convert a mixture of experts to the Generative Topographic Mapping (GTM). In contrast to the SOM, neither of these mappings quantizes but rather spread the points across the manifold.

We begin with a set of experts who reside in some latent space and take responsibility for generating the data set. In a mixture of experts [17, 18], the experts divide up the data space between them, each taking responsibility for a part of the data space. This division of labour enables each expert to concentrate on a specific part of the data set and ignore those regions of the space for which it has no responsibility. The probability associated with any data point is the sum of the probabilities awarded to it by the experts. There are efficient algorithms, notably the Expectation-Maximization algorithm, for finding the parameters associated with mixtures of experts [1] constrained the experts' positions in latent space and showed that the resulting mapping also had topology preserving properties.

The Generative Topographic Mapping (GTM) [1] is a mixture of experts model which treats the data as having been generated by a set of latent points. These K latent points are also mapped through a set of M basis functions and a set of adjustable weights to the data space. The parameters of the combined mapping are adjusted to make the data as likely as possible under this mapping. The GTM is a probabilistic formulation so that if we define $\mathbf{y} = \Phi\mathbf{W} = \Phi(\mathbf{t})\mathbf{W}$, where \mathbf{t} is the vector of latent points, the probability of the data is determined by the position of the projections of the latent points in data space and so we must adjust this position to increase the likelihood of the data. More formally, let

$$\mathbf{m}_i = \Phi(\mathbf{t}_i)\mathbf{W} \quad (10)$$

be the projections of the latent points into the feature space. Then, if we assume that each of the latent points has equal probability

$$p(\mathbf{x}) = \sum_{i=1}^K P(i)p(\mathbf{x}|i) = \sum_{i=1}^K \frac{1}{K} \left(\frac{\beta}{2\pi}\right)^{\frac{D}{2}} \exp\left(-\frac{\beta}{2}\|\mathbf{m}_i - \mathbf{x}\|^2\right) \quad (11)$$

where D is the dimensionality of the data space – in other words, all the data is assumed to be noisy versions of the mapping of the latent points.

In the GTM, the parameters W and β are updated using the EM algorithm to maximise the likelihood of the data *under this model*. Thus we must be very precise about the format of this model. More specifically, the underlying structure of the experts can be represented by K latent points, t_1, t_2, \dots, t_K . To allow local and non-linear modeling, we map those latent points through a set of M basis functions, $f_1(), f_2(), \dots, f_M()$. This gives us a matrix Φ where $\phi_{kj} = f_j(t_k)$. Thus each row of Φ is the response of the basis functions to one latent point, or alternatively we may state that each column of Φ is the response of one of the basis functions to the set of latent points. One of the functions, $f_j()$, acts as a bias term and is set to one for every input. Typically the others are Gaussians centered in the latent space. The outputs of these functions are then mapped by a set of weights, W , into data space. W is $M \times D$, where D is the dimensionality of the data space, and is the sole parameter which we change during training. We will use \mathbf{w}_i to represent the i^{th} column of W and Φ_j to represent the row vector of the mapping of the j^{th} latent point. Thus each basis point is mapped to a point in data space, $\mathbf{m}_j = (\Phi_j W)^T$.

To change W , we consider a specific data point, say \mathbf{x}_i . We calculate the current responsibility of the j^{th} latent point for this data point

$$r_{ij} = \frac{\exp(-\gamma d_{ij}^2)}{\sum_k \exp(-\gamma d_{ik}^2)} \quad (12)$$

where $d_{pq} = \|\mathbf{x}_p - \mathbf{m}_q\|$, the Euclidean distance between the p^{th} data point and the projection of the q^{th} latent point (through the basis functions and then multiplied by W). If no centres are close to the data point (the denominator of Eqn.(12) is zero), we set $r_{ij} = \frac{1}{K}, \forall j$.

We use these responsibilities to change W using

$$W_{new}^T = (\Phi^T G \Phi)^{-1} \Phi^T R X \quad (13)$$

where R is the matrix of responsibilities, G is a diagonal matrix with $G_{ii} = \sum_j r_{ij}$ and X is the $N \times D$ data matrix. β is similarly adjusted using

$$\frac{1}{\beta_{new}} = \frac{1}{ND} \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|W_{new} \Phi(t_k) - \mathbf{x}_n\|^2 \quad (14)$$

4.1 Illustrative Examples

We begin with an example from the GTM_DEMO from Netlab [25] which is highly recommended (<http://www.ncrg.aston.ac.uk/netlab/index.php>). In Fig. 5 we show the convergence of the GTM on an artificial two-dimensional

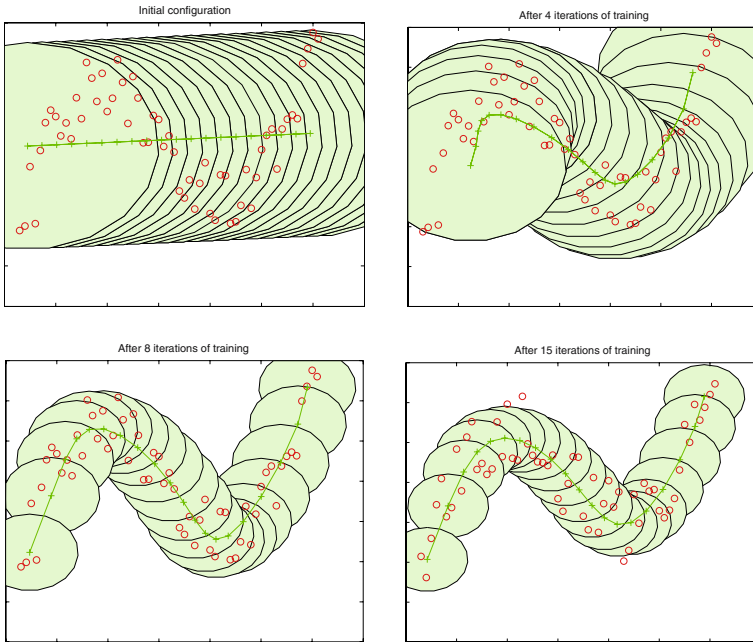


Fig. 5. The initialised GTM centres are strung out along the first principal component and all mixture components have equal variance; we then see the changes in the positions of the centres and the variances of the components after 4, 8 and 15 iterations

data set in which $y = x + 1.25 \sin(2x) + \mu$, where x is drawn uniformly from $[0.15, 3.05]$ and μ is noise. The centres are initialized to lie uniformly along the first principal component of the data and we can see that, during training, they move to the centre of the data manifold very quickly (the top right diagram shows the positions after only four iterations of the EM algorithm). Eventually the system stabilizes with the variance of the Gaussians very much smaller than the initial estimates.

It is of interest to compare the GTM on the algae data: we use a two-dimensional latent space with a 10×10 grid for comparison. The results are shown in Fig. 6. The GTM makes a very confident classification: we see that the responsibilities for data points are very confidently assigned, in that individual classes tend to be allocated to a single latent point. This, however works against the GTM in that, even with zooming in to the map, one cannot sometimes disambiguate the two different classes such as at the points $(1, -1)$ and $(1, 1)$. This was not alleviated by using regularization in the GTM though we should point out that we have a very powerful model for a rather small data set.

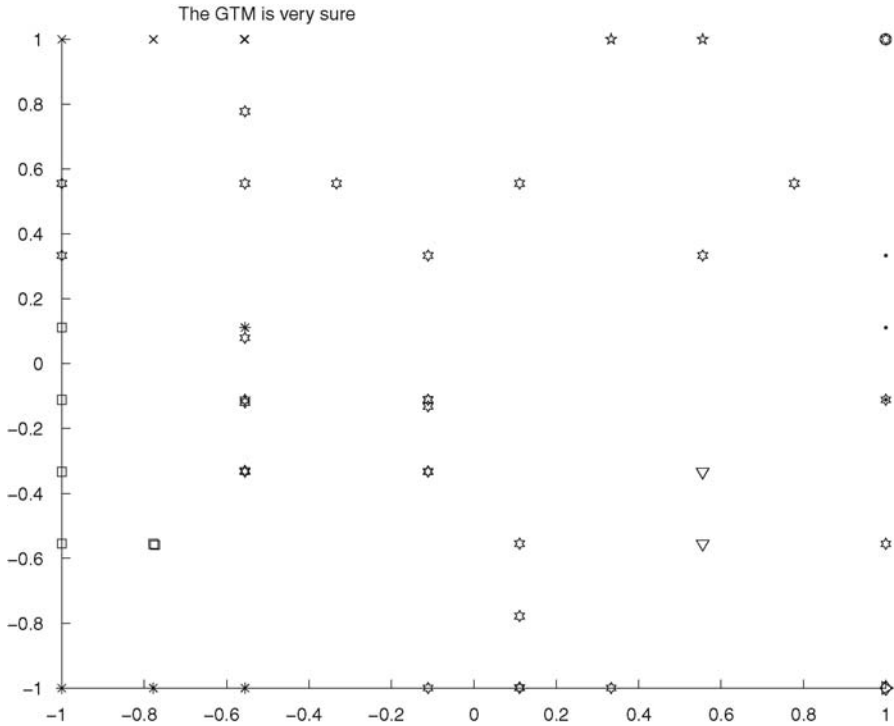


Fig. 6. The projection of the algae data given by the GTM

4.2 Adjusting the Latent Space

The GTM suffers from a common problem with topographic mappings – the latent points determine the topography *a priori* and the model is then made to fit the data as well as possible. So what happens in cases in which the model does *not* fit the data so well? An example is shown in Fig. 7: we see that the data is composed of four clusters of points which lie approximately on a line. We have used a 1-dimensional set of latent points and so they should be well-matched to the data. However, some of the latent points are mapped to regions of the data space in which there is *no* data. As noted above, this type of problem has been tackled by SOM researchers but does not feature in GTM research. Yet there are many models which combine top-down generative models with bottom-up data driven modeling, for instance [15, 29].

These models alternate optimising the parameters assuming the latent model to be correct with optimising the parameters assuming the data has priority. We now add the latter feature to the GTM. Note that

$$P(t|\mathbf{x}) = \frac{P(t, \mathbf{x})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|t)P(t)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|t)}{KP(\mathbf{x})} \tag{15}$$

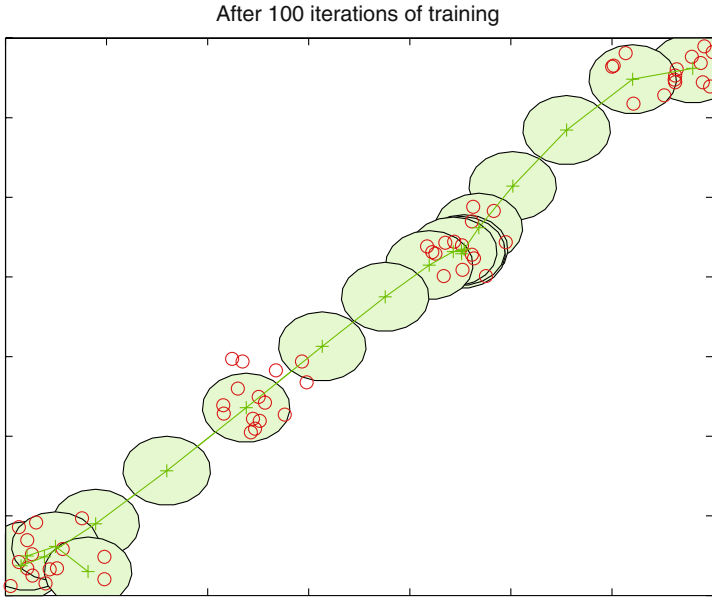


Fig. 7. The standard GTM positions the projection of some latent points well away from the data

if $P(t) = \frac{1}{K}$. Then we wish to maximize this probability – that is, we are assuming now that the data has priority and retain the same model structure as before, but now change the position of the latent points in latent space. With some abuse of the notation, we will now use t both for the latent point and its position. We note that the denominator is independent of t and so

$$\frac{\partial P(t|\mathbf{x})}{\partial t} \propto \exp(-\beta/2 * (\Phi(t)W - \mathbf{x})^T(\Phi(t)W - \mathbf{x}))\beta(\Phi(t)W - \mathbf{x})^T(\Phi(t)W) \tag{16}$$

Thus we enable the latent points to change positions in latent space using gradient ascent. We have found it useful to either

1. use the constraint that they must retain their ordering so that $t_i < t_{i+1} + \epsilon, \forall i$ at all times; ϵ is a small real number used to ensure there remains some distance between the latent points.
2. or if the change to the k^{th} latent point takes it closer to the $(k + 1)^{th}$, then move all the points from k up to K by the same amount. Similarly if the change to the k^{th} latent point takes it closer to the $(k - 1)^{th}$, move all the points from k down to 1 by the same amount.

Both of these methods are designed to ensure that the resulting movement retains the topographic ordering of the latent points.

Using gradient descent with the latter method, we get positions such as shown in Fig. 8.

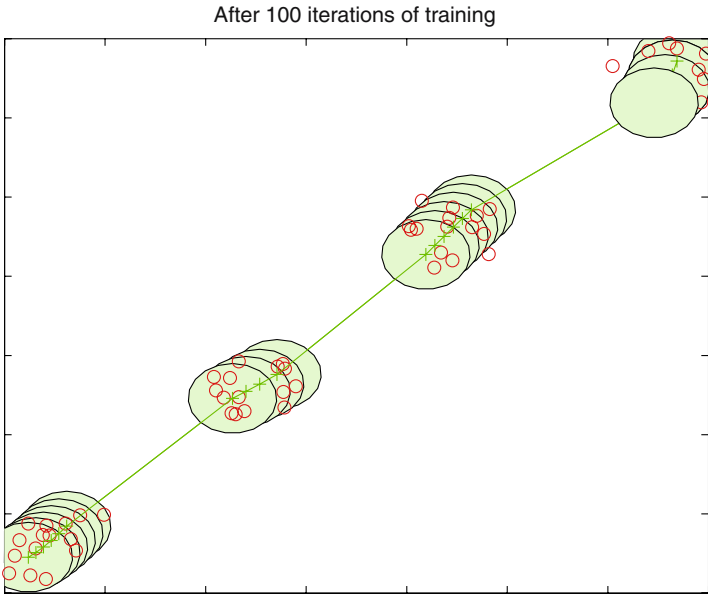


Fig. 8. Allowing the positions of the latent points to change (in latent space) enables their projections to better fit the data

Examining the positions of the latent points in latent space we see that they have formed four clusters which enables the nonlinear mapping to data space to easily identify the four clusters of data points. The final positions of the latent points are shown in Fig. 9.

4.3 Deleting Latent Points

An alternative is to delete latent points which have been misplaced. To determine a latent point which has been misplaced, we need only ascertain which latent points have not been given greatest responsibility for any data points. An equivalent criterion would be to determine which latent points have projections in data space which are not closest to any data point. Whichever criterion is used, such points can be deleted. The positions of the remaining latent points in latent space are not changed but the Φ matrix must be recalculated. Training now continues with the reduced set. An example of this method is shown in Fig. 10.

5 Topographic Product of Experts (ToPoE)

In a product-of-experts, all the experts take responsibility for all the data: the probability associated with any data point is the (normalized) product of the probabilities given to it by the experts. As pointed out in for example [16],

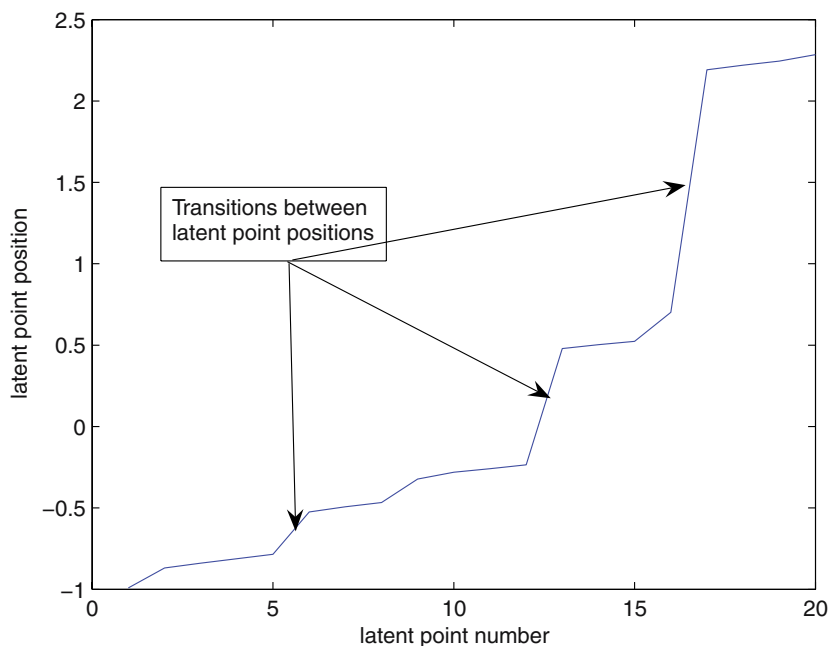


Fig. 9. The final positions of the latent points

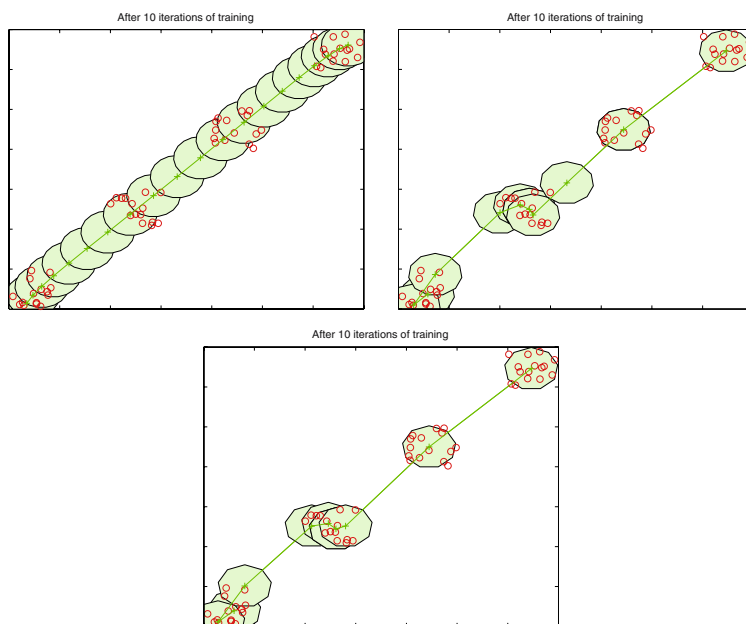


Fig. 10. The projections of the GTM latent points after 10, 20 and 30 iterations, at which time the map had stabilized. After every 10 iterations, those points not currently closest to any data point are deleted

this enables each expert to waste probability mass in regions of the data space where there is no data, provided each expert wastes his/her mass in a different region. The most common situation is to have each expert take responsibility for having information about the data's position in one dimension while having no knowledge at all about the other dimensions, a specific case of which is called a Gaussian pancake [28]: a probability density function which is very wide in most dimensions but is very narrow (precisely locating the data) in one dimension. It is very elegantly associated with Minor Components Analysis [28].

[14] investigated a product of K experts with

$$p(\mathbf{x}_n|\Theta) \propto \prod_{k=1}^K p(\mathbf{x}_n|k) \quad (17)$$

where Θ is the set of current parameters in the model. Hinton notes that using Gaussians alone does not allow us to model say multi-modal distributions, however the Gaussian is ideal for our purposes. Thus our base model is

$$p(\mathbf{x}_n|\Theta) \propto \prod_{k=1}^K \left(\frac{\beta}{2\pi}\right)^{\frac{D}{2}} \exp\left(-\frac{\beta}{2}\|\mathbf{m}_k - \mathbf{x}_n\|^2\right) \quad (18)$$

We will, as with the GTM, allow latent points to have different responsibilities depending on the data point presented:

$$p(\mathbf{x}_n|\Theta) \propto \prod_{k=1}^K \left(\frac{\beta}{2\pi}\right)^{\frac{D}{2}} \exp\left(-\frac{\beta}{2}\|\mathbf{m}_k - \mathbf{x}_n\|^2 r_{kn}\right) \quad (19)$$

where r_{kn} is the responsibility of the k^{th} expert for the data point \mathbf{x}_n . Thus all the experts are acting in concert to create the data points but some will take more responsibility than others. Note how crucial the responsibilities are in this model: if an expert has no responsibility for a particular data point, it is in essence saying that the data point could have a high probability as far as it is concerned. We do not allow a situation to develop where no expert accepts responsibility for a data point; if no expert accepts responsibility for a data point, they all are given equal responsibility for that data point (see below). For comparison, the probability of a data point under the GTM is

$$p(\mathbf{x}) = \sum_{i=1}^K P(i)p(\mathbf{x}|i) = \sum_{i=1}^K \frac{1}{K} \left(\frac{\beta}{2\pi}\right)^{\frac{D}{2}} \exp\left(-\frac{\beta}{2}\|\mathbf{m}_i - \mathbf{x}\|^2\right) \quad (20)$$

We wish to maximize the likelihood of the data set $X = \{\mathbf{x}_n : n = 1, \dots, N\}$ under this model. The ToPoE learning rule Eqn. (22) is derived from the minimization of $-\log(p(\mathbf{x}_n|\Theta))$ with respect to a set of parameters which generate the \mathbf{m}_k .

The underlying model is identical to the GTM: we have K experts which generate the K centres, \mathbf{m}_k . The experts can be represented by K latent points, t_1, t_2, \dots, t_K which are mapped through a set of M basis functions, $f_1(), f_2(), \dots, f_M()$. This gives us a matrix Φ where $\phi_{kj} = f_j(t_k)$. The output of these functions are then mapped by a set of weights, W , into data space. W is $M \times D$, where D is the dimensionality of the data space, and is the sole parameter which we change during training. Each basis point is mapped to a point in data space, $\mathbf{m}_j = (\Phi_j W)^T$.

We may update W either in batch mode or with online learning. To change W in online learning, we randomly select a data point, say \mathbf{x}_i . We calculate the current responsibility of the j^{th} latent point for this data point,

$$r_{ij} = \frac{\exp(-\gamma d_{ij}^2)}{\sum_k \exp(-\gamma d_{ik}^2)} \quad (21)$$

where $d_{pq} = \|\mathbf{x}_p - \mathbf{m}_q\|$, the Euclidean distance between the p^{th} data point and the projection of the q^{th} latent point (through the basis functions and then multiplied by W). If no centres are close to the data point (the denominator of Eqn.(21) is zero), we set $r_{ij} = \frac{1}{K}, \forall j$.

Now we wish to maximize Eqn.(20) so that the data is most likely under this model. We do this by minimizing the $-\log()$ of that probability: define $m_d^{(k)} = \sum_{m=1}^M w_{md} \phi_{km}$ – in other words, $m_d^{(k)}$ is the projection of the k^{th} latent point on the d^{th} dimension in data space. Similarly let $x_d^{(n)}$ be the d^{th} coordinate of \mathbf{x}_n . These are used in the update rule

$$\Delta_n w_{md} = \sum_{k=1}^K \eta \phi_{km} (x_d^{(n)} - m_d^{(k)}) r_{kn} \quad (22)$$

where we have used Δ_n to signify the change due to the presentation of the n^{th} data point, \mathbf{x}_n , so that we are summing the changes due to each latent point's response to the data points. Note that, for the basic model, we do not change the Φ matrix during training at all.

5.1 Comparison with the GTM

The Generative Topographic Mapping (GTM) [1] is a mixture-of-experts model which treats the data as having been generated by a set of latent points. The GTM is a probabilistic formulation so that if we define $\mathbf{y} = \Phi \mathbf{W} = \Phi(\mathbf{t})\mathbf{W}$, where \mathbf{t} is the vector of latent points, the probability of the data is determined by the position of the projections of the latent points in data space and so we must adjust this position to increase the likelihood of the data. Then, if we assume that each of the latent points has equal probability

$$p(\mathbf{x}) = \sum_{i=1}^K P(i) p(\mathbf{x}|i) = \sum_{i=1}^K \frac{1}{K} \left(\frac{\beta}{2\pi} \right)^{\frac{D}{2}} \exp \left(-\frac{\beta}{2} \|\mathbf{m}_i - \mathbf{x}\|^2 \right) \quad (23)$$

where D is the dimensionality of the data space – that is, all the data is assumed to be noisy versions of the mapping of the latent points. This equation should be compared with Eqns. (19) and (20).

In the GTM, the parameters W and β are updated using the EM algorithm though the authors do state that they could use gradient ascent. Indeed, in the ToPoE, the calculation of the responsibilities may be thought of as being a partial E-step while the weight update rule is a partial M-step.

The GTM, however, does have the advantage that it can optimise with respect to β as well as W . However note that, in Eqns. (19) and (20), the variance of each expert is dependent on its distance from the current data point via the hyper-parameter, γ . Thus we may define

$$(\beta_k)_{|\mathbf{x}=\mathbf{x}_n} = \beta r_{kn} = \beta \frac{\exp(-\gamma d_{nk}^2)}{\sum_t \exp(-\gamma d_{nt}^2)} \quad (24)$$

Therefore the responsibilities are adapting the width of each expert locally dependent on both the expert’s current projection into data space and the data point for which responsibility must be taken. Initially, $r_{kn} = \frac{1}{K}, \forall k, n$ and so we have the standard product-of-experts. However during training, the responsibilities are refined so that individual latent points take more responsibility for specific data points. We may view this as the model softening from a true product of experts to something between that and a mixture of experts.

A model based on products of experts has some advantages and disadvantages. The major disadvantage is that no efficient EM algorithm exists for optimizing parameters. [14] suggests using Gibbs sampling but even with the very creative method discussed in that paper, the simulation times were excessive. Thus we have opted for gradient descent as the parameter optimization method.

The major advantage which a product-of-experts method has is that it is possible to get very much sharper probability density functions with a product rather than a sum of experts.

5.2 Illustrative Example

Figure 11 shows the result of a simulation in which we have 20 latent points deemed to be equally spaced in a one-dimensional latent space, passed through five Gaussian basis functions and then mapped to the data space by the linear mapping W which is the only parameter we adjust. We generated 60 two-dimensional data points, (x_1, x_2) , from the function $x_2 = x_1 + 1.25 \sin(x_1) + \mu$ where μ is noise from a uniform distribution in $[0, 1]$. We use 10000 iterations of the learning rule (randomly sampling with replacement from the data set) with $\beta = 2, \gamma = 20, \eta = 0.1$. The final placement of the projections of the latent points is shown by the asterisks in the Figure and we clearly see that

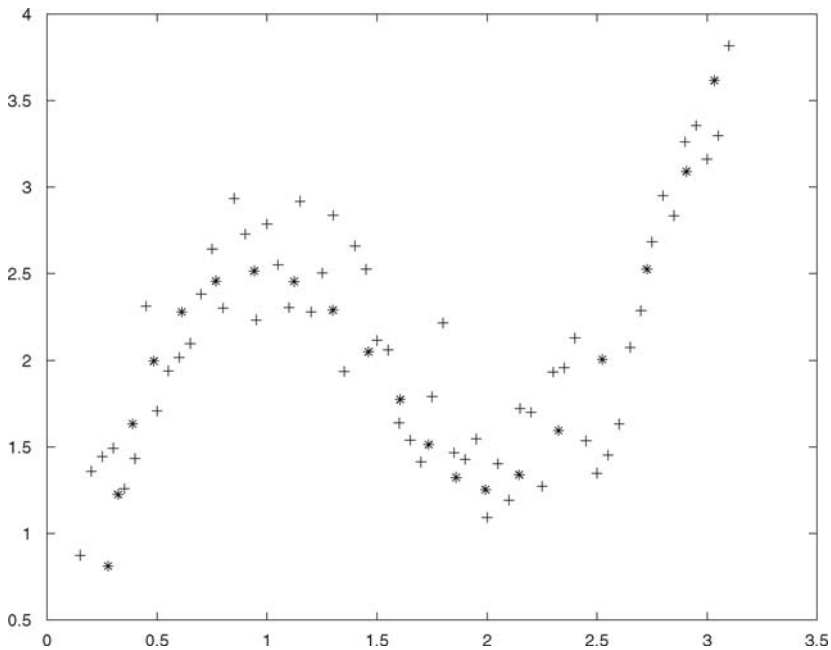


Fig. 11. The projections of 20 latent points into data space is shown by the asterisks; the training data are the other points

the 1-dimensional nature of the data has been identified. Also, the centres are placed along this manifold in the order in which they appear in the latent space showing that a topographic projection has been created.

We have similar results when we use a batch method, presenting all the data and not updating the weights till we have accumulated all the changes. Also we have similar experiments with higher dimensional data and grids, for instance with 400 latent points arranged in a two dimensional grid (20×20) and 5×5 basis functions.

We may show the growth of the responsibilities from either the perspective of an individual latent point (Fig. 13) or from the perspective of a single data point (Fig. 12). Initially we see the latent points assuming a broad responsibility which is refined in time till each latent point has only a responsibility for a few data points and conversely each data point is being generated (under the model) by only a few latent points: we have moved some way from the product-of-experts towards a mixture-of-experts.

5.3 Projections

As a visualization technique the ToPoE has one advantage over the standard SOM: the projections of the data onto the grid need not be solely to

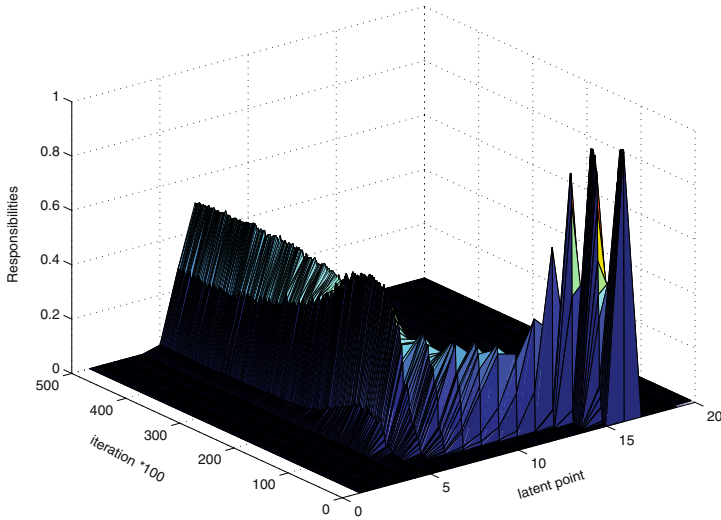


Fig. 12. There is an initial competition to take responsibility for a specific data point but quickly converge so that just a few latent points do so

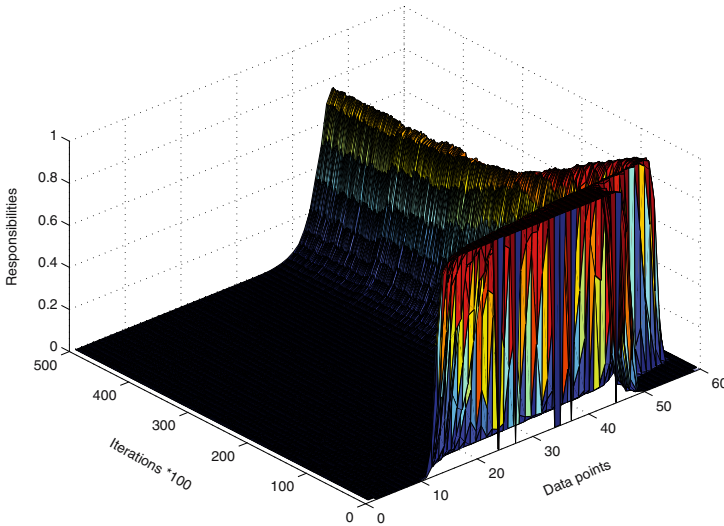


Fig. 13. The latent point initially has broad responsibilities but learns to take responsibility for only a few data points

the grid nodes. If we project each data point to that node which has highest responsibility for the data point, we get a similar quantization to that of the SOM. However if we project each data point, \mathbf{x}_n onto $\sum_k \mathbf{m}_k * r_{kn}$, we get a mapping onto the manifold at intermediate points. Figure 14 (left) shows

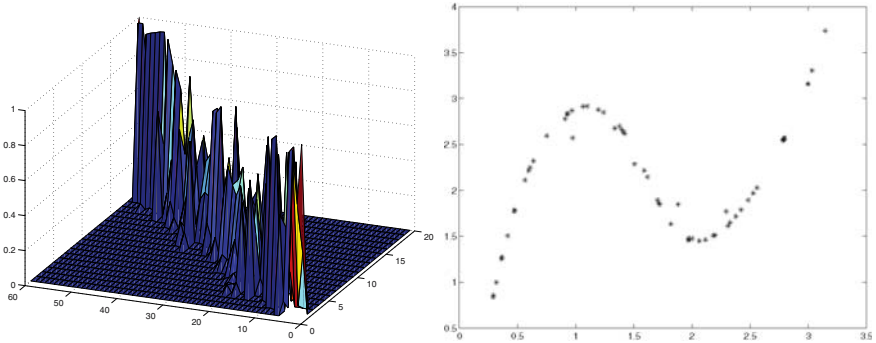


Fig. 14. *Left:* the responsibilities of the 20 latent points for 60 data points which are arranged in approximately increasing distance along the manifold; *Right:* the re-projection of the 60 data points onto the manifold

the responsibilities which 20 latent points have for 60 data points (arranged in ascending order of their position along the manifold), and (right) the subsequent re-projection of the latent points to the data space when taking these responsibilities into account.

5.4 Growing and Pruning ToPoEs

One advantage of this method is that we can easily grow a net: we train a net with a small number of latent points and then increase the number of latent points. Thus we have to recalculate the Φ matrix but need not change the W matrix of weights which can simply continue to learn from its current values. An example is shown in Fig. 15 in which we use five basis functions (together with a bias term) on the same data as before and increase the number of latent points from 7 to 20. The mapping becomes increasingly smooth.

Equally we may question the completed map to investigate whether any latent point is being mapped to a part of the data space which has no data nearby. If a latent point does not have the greatest responsibility for any data point, it can be deleted from the map. This technique is illustrated in Fig. 16. In each diagram the ‘+’s show the positions of the data points: the data consists of 4 distinct clusters. The trained map is shown on the left: the projections of the 20 latent points map cover the data set but some are placed in positions in which there is no data for which they need take responsibility. Such points are excluded and the map continues to learn to get the situation in the right diagram: only 10 latent points remain. It must be emphasized that we do not alter the positions of either the latent points (in latent space) or the basis functions when we continue training. These remain at their original locations.

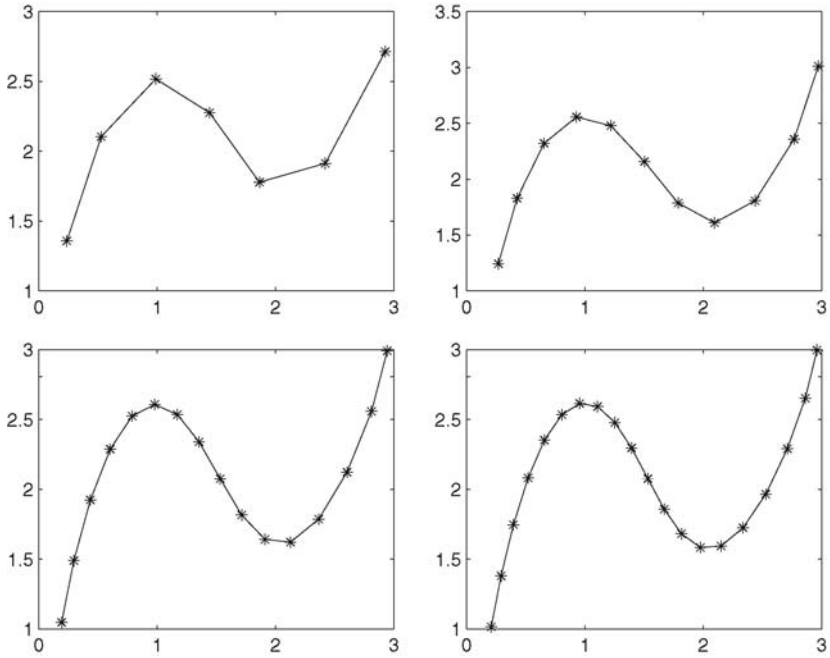


Fig. 15. The growing map. *Top left:* 7 latent points; *top right:* 11 latent points; *bottom left:* 16 latent points; *bottom right:* 20 latent points

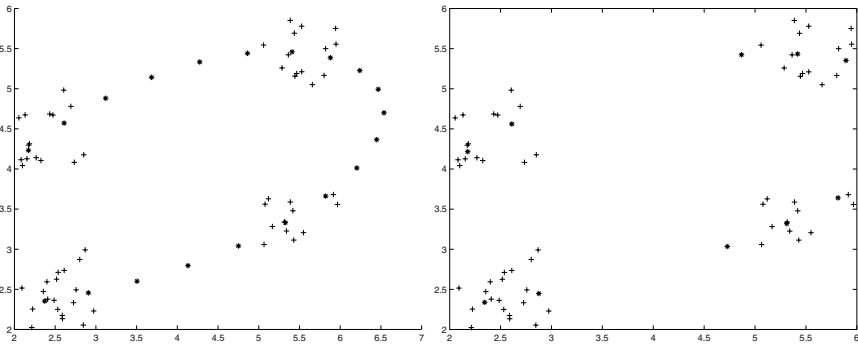


Fig. 16. In both diagrams the data set is shown by the '+'s. *Left:* the projections of the 20 latent points are shown with '*'s.; *right:* after pruning, the 10 remaining latent points may continue training

5.5 Different Noise Models

We may change the underlying noise model and reflect this in a different learning rule. For example, an alternative model based on a Laplacian distribution is

$$p(\mathbf{x}_n) = \frac{1}{Z} \exp \left(-\frac{\beta}{2} \sum_{k=1}^K (\|\mathbf{m}_k - \mathbf{x}_n\|_1 r_{kn}) \right) \tag{25}$$

where $\|\cdot\|_1$ signifies the 1-norm [8]. In this case, we derive the learning rule

$$\Delta_n w_{md} = \eta \sum_{k=1}^K \phi_{km} \text{sign}(x_d^{(n)} - m_d^{(k)}) r_{kn} \tag{26}$$

where $\text{sign}(t) = 1$, if $t > 0$ and $\text{sign}(t) = -1$, otherwise.

While this rule may be more appropriate for rather more kurtotic noise than in the above simulations, it can be used with data which is corrupted by Gaussian noise or even uniform (and hence far from kurtotic) noise. Simulations on exactly the same data as used for Fig.1 have shown similar convergence to that achieved in that figure.

5.6 Twinned ToPoEs

[11] have previously investigated twinning principal curves and self-organizing maps with a view to forecasting one data set from another with which it has some (nonlinear) correlation. We may do the same with the ToPoE. Consider first having a single underlying cause which we can map into two data spaces simultaneously – namely, we have $\mathbf{m}_k = \Phi_k W_1$ in the first data space and $\mathbf{l}_k = \Phi_k W_2$ in the second data space. Figure 17 illustrates the results of this twinning when we calculate a single responsibility of each latent point for both data points together. Let \mathbf{x}_i in one space be twinned with \mathbf{y}_i in the second data space. Then

$$r_{ij} = \frac{\exp(-\gamma d_{ij}^2)}{\sum_k \exp(-\gamma d_{ik}^2)} \tag{27}$$

where $d_{pq} = \|\mathbf{x}_p - \mathbf{m}_q\| + \|\mathbf{y}_p - \mathbf{l}_q\|$. The ‘*’s in Fig. 17 illustrate that the latent points have indeed been mapped to appropriate positions in data space.

However unless one changes either the nature of the nonlinearity which maps from latent space or the noise model, one can argue that this method is equivalent to a single ToPoE mapping into a data space of dimension equal to the sum of the dimensions of the two spaces in this Section. Therefore we really require to use different nonlinear basis functions or different noise models to benefit from this method. A different noise model was discussed above when we used Laplacian noise; an alternative is to use other radial basis functions or as, we shall see, even non-radial functions.

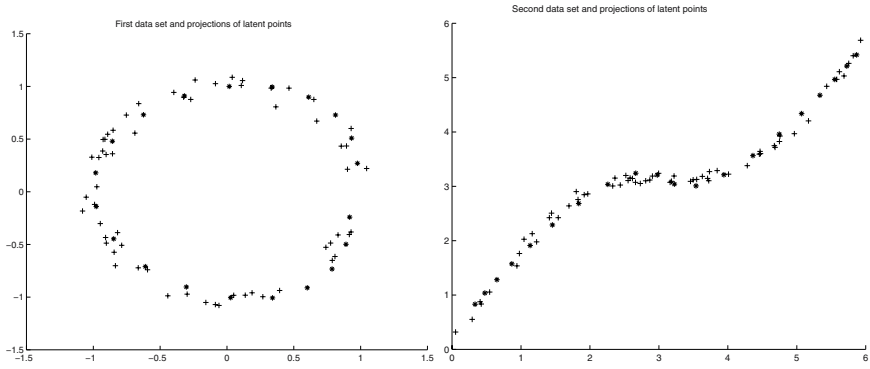


Fig. 17. The two diagrams show the two data sets ('+'s) and the projections of the latent points ('*'s)

[13] lists the most common radial basis functions as

1. multiquadrics: $f_j(t_i) = \sqrt{(j^2 + t_i^2)}$
2. inverse multiquadrics: $f_j(t_i) = \frac{1}{\sqrt{(j^2 + t_i^2)}}$
3. and of course the Gaussian used above.

Note that the first of these is non-local in character but [13] notes that such functions can approximate a smooth mapping with greater accuracy than say the Gaussian.

Of course, we need not restrict ourselves to these functions. For example, we may keep the model entirely as it was above but use a matrix Φ in which $\phi_{kj} = f_j(t_k) = \tanh(jt_k)$. On the same data set as previously the $\tanh()$ model gives very similar results. It might be thought that the $\tanh()$ non-linearity, being global, might present a difficulty for the learning of the local responsibilities. Figure 18 shows that this is not so.

5.7 Visualizing and Clustering Real Data Sets

In this Section, we use a 2-dimensional grid of latent points: we use a 10×10 grid of latent points being passed through a 5×5 set of Gaussian basis vectors. We begin by illustrating the method on the well-known wine data set from the UCI Repository of machine learning databases (<http://www.ics.uci.edu/~mllearn/MLSummary.html>). It has 178 samples, 13 features and 3 classes. The resulting projection is shown in Fig. 19. Because some of the features are scaled up to 1500 and others lie between 0 and 1, we preprocessed the data by normalizing all features between -0.1 and 0.1 . The clustering is obvious.

Figure 20 shows the projection of the 9 labeled classes (72 samples). Most of these are easily identified. When we zoom into the central part of this mapping (Fig. 21(left)), we find that we can disambiguate the 8th and 9th classes. However, the right half of that figure suggests that the remaining two

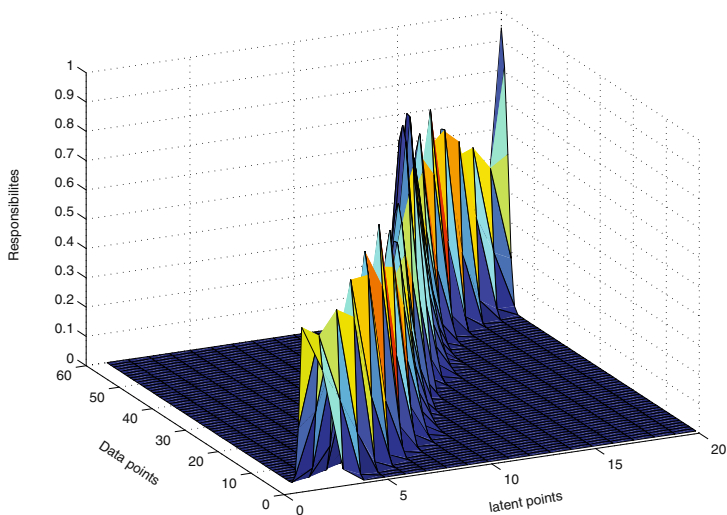


Fig. 18. Even though the $\tanh()$ nonlinearity is not local, the responsibilities learned are very local

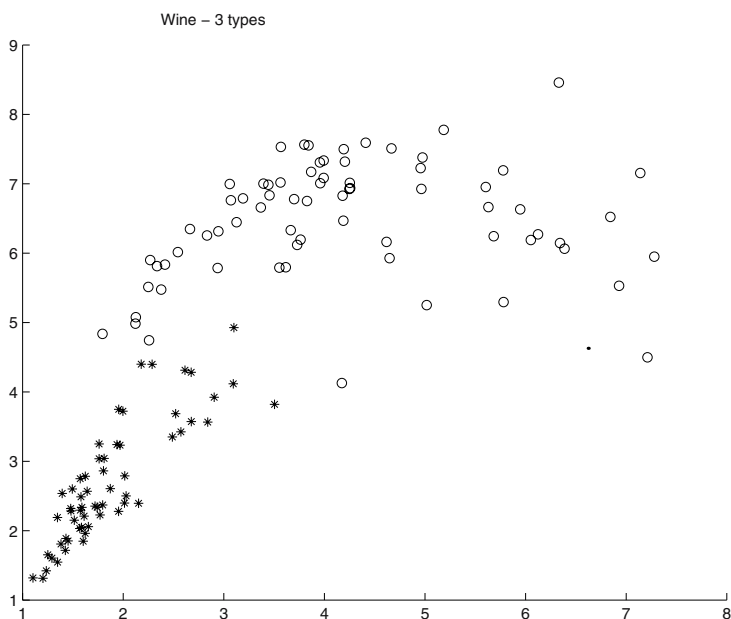


Fig. 19. The projection of the wine data set

classes are not completely distinguished. Figure 22 shows the projection of the whole data set including the unlabeled samples. From this, we conjecture that

- there are other classes in the data set which have not yet been identified,

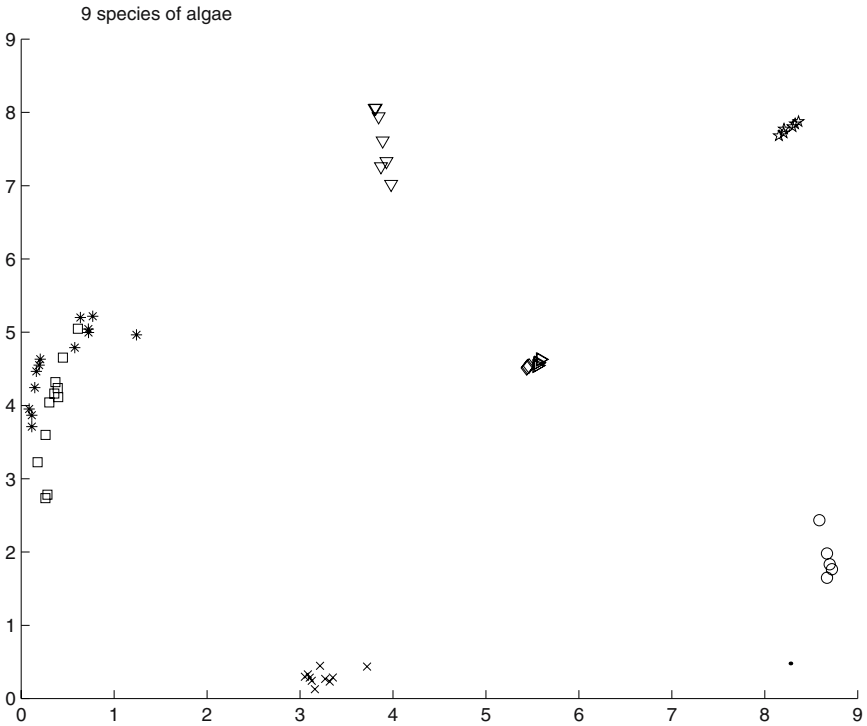


Fig. 20. Projection of the 9 classes by the ToPoE

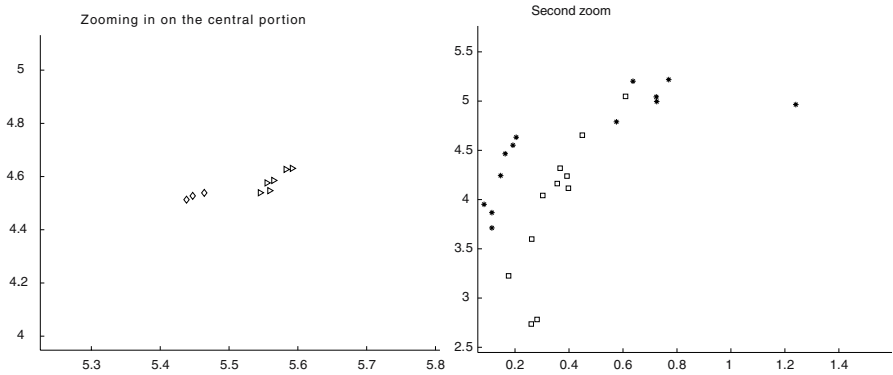


Fig. 21. *Left*: zooming in on the central portion; *Right*: zooming in on the left side

- some of the unclassified samples belong to classes already identified,
- some may be simply outliers.

These are, however, speculations on our part and must be validated by a scientist with biological expertise.

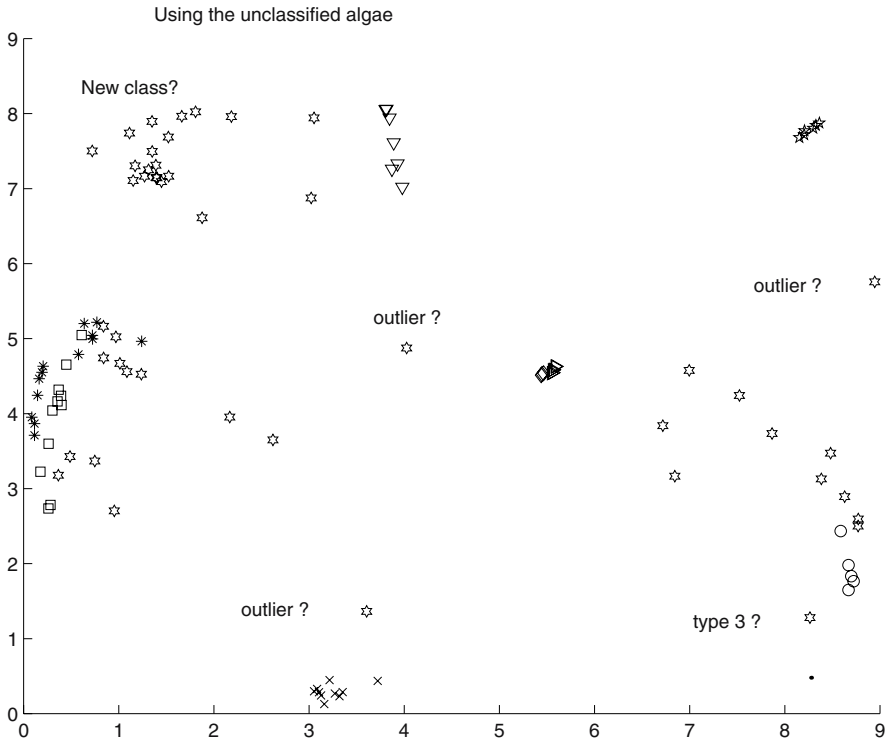


Fig. 22. The projection of the whole data set by the ToPoE

In fact, we can control the level of quantization by changing the γ parameter in Eqn. (21). For example by lowering γ , we share the responsibilities more equally and so the map contracts to the centre of the latent space to get results such as shown in Fig. 23; the different clusters can still be identified but rather less easily. Alternately, by increasing γ , one tends to get the data clusters confined to a single node, that which has sole responsibility for that cluster.

5.8 Discussion

We have shown above that we can grow these maps incrementally and prune them if necessary also. Intuitively, since this growth requires us only to change the number K of latent points, and we are only adapting W which is $M \times D$ and so not directly concerned with the latent points, we can train W with a certain number of latent points and then increase this number but simply continue training W without resetting W : W is approximately correct and training can be continued from its current values. This is an optional feature in

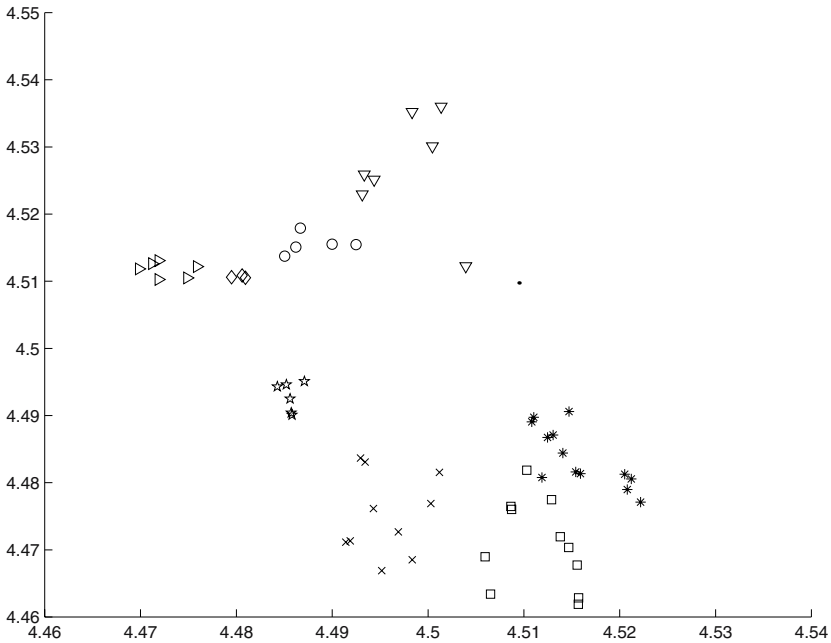


Fig. 23. By lowering the γ parameter, the ToPoE map is contracted

ToPoE but becomes essential in the Harmonic Topographic Mapping, HaToM, which we discuss in the next Section.

One feature of ToPoE which is less than satisfactory is that when no latent point accepts responsibility for a data point, all latent points are given equal responsibility for that data point. This ensures that every data point is covered by the projections of the latent points but, while this is a sensible thing to do at the start of training, it seems unconvincing when it is performed in the middle of training. We therefore seek a mapping which does not have this feature.

As we leave this Section, we note that minimization of the logarithm of Eqn. (20) is equivalent to minimization of the mean squared error between the data and the projections of the latent points. This has led us to investigate alternative criteria such as used in the HaToM.

6 Harmonic Averages

Harmonic Means or Harmonic Averages are defined for spaces of derivatives. For example, if you travel half of a journey at 10 km/hour and the other half at 20 km/hour, your total time taken is $\frac{d}{10} + \frac{d}{20}$ and so the average speed is

$\frac{2d}{\frac{d}{10} + \frac{d}{20}} = \frac{2}{\frac{1}{10} + \frac{1}{20}}$. In general, the Harmonic Average is defined as

$$HA(\{a_i, i = 1, \dots, K\}) = \frac{K}{\sum_{k=1}^K \frac{1}{a_k}} \quad (28)$$

6.1 Harmonic k-means

This has recently [30, 31] been used to make the K -means algorithm more robust. The k-means algorithm [12] is a well-known clustering algorithm in which N data points are allocated to K means which are positioned in data space. The algorithm is known to be dependent on its initialization: a poor set of initial positions for the means will cause convergence to a poor final clustering. [30, 31] have developed an algorithm based on the Harmonic Average which converges to a better solution than the standard algorithm.

The algorithm calculates the Euclidean distance between the i^{th} data point and the k^{th} centre as $d(\mathbf{x}_i, \mathbf{m}_k)$. Then the performance function using Harmonic averages seeks to minimize

$$Perf_{HA} = \sum_{i=1}^N \frac{K}{\sum_{k=1}^K \frac{1}{d(\mathbf{x}_i, \mathbf{m}_k)^2}} \quad (29)$$

Then we wish to move the centres using gradient descent on this performance function

$$\frac{\partial Perf_{HA}}{\partial \mathbf{m}_k} = -K \sum_{i=1}^N \frac{4(\mathbf{x}_i - \mathbf{m}_k)}{d(\mathbf{x}_i, \mathbf{m}_k)^3 \left\{ \sum_{l=1}^K \frac{1}{d(\mathbf{x}_i, \mathbf{m}_l)^2} \right\}^2} \quad (30)$$

Setting this equal to 0 and ‘solving’ for the \mathbf{m}_k ’s, we get a recursive formula

$$\mathbf{m}_k = \frac{\sum_{i=1}^N \frac{1}{d_{i,k}^3 \left(\sum_{l=1}^K \frac{1}{d_{i,l}^2} \right)^2} \mathbf{x}_i}{\sum_{i=1}^N \frac{1}{d_{i,k}^3 \left(\sum_{l=1}^K \frac{1}{d_{i,l}^2} \right)^2}} \quad (31)$$

where we have used $d_{i,k}$ for $d(\mathbf{x}_i, \mathbf{m}_k)$ to simplify the notation. There are some practical issues to deal with in the implementation details of which are given in [30, 31].

[31] have extensive simulations showing that this algorithm converges to a better solution (less prone to finding a local minimum because of poor initialization) than both standard K -means or a mixture of experts trained using the EM algorithm.

6.2 The Harmonic Topographic Map

The above can now be used with the latent variable model. Since

$$\frac{\partial \text{Perf}_{HA}}{\partial W} = \frac{\partial \text{Perf}_{HA}}{\partial \mathbf{m}_k} \frac{\partial \mathbf{m}_k}{\partial \mathbf{W}} = \frac{\partial \text{Perf}_{HA}}{\partial \mathbf{m}_k} \Phi_k \quad (32)$$

we could use the algorithm directly in a learning rule as with the ToPoE. However an alternative method is suggested in this Chapter.

With this learning rule on the same model as above, we get a mapping which has elements of topology preservation but which often exhibits twists, such as are well-known in the SOM [21]. We therefore opt to begin with a small value of K (for 1-dimensional latent spaces, we tend to use $K = 2$, for 2-dimensional latent spaces and a square grid, we use $K = 2 \times 2$) and grow the mapping. As noted earlier, we do not randomize W each time we augment K . The current value of W is approximately correct and so we need only continue training from this current value. Also for this Chapter we have implemented a pseudo-inverse method for the calculation of W from the positions of the centres, rather than Eqn. (32). Then the algorithm is

Algorithm 2 Harmonic Topographic Map (HaToM) Algorithm

1. Initialise K to 2; initialise the W weights randomly and spread the centres of the M basis functions uniformly in latent space.
 2. Initialise the K latent points in latent space.
 3. Calculate the projection of the latent points to data space. This gives the K centres, \mathbf{m}_k .
 - (a) count = 0
 - (b) For every data point, \mathbf{x}_i , calculate $d_{i,k} = \|\mathbf{x}_i - \mathbf{m}_k\|$.
 - (c) Recalculate means using Eqn.(31).
 - (d) If count < MAXCOUNT, count = count + 1 and return to 3(b).
 4. Recalculate W using $(\Phi^T \Phi + \delta I)^{-1} \Phi^T \mathbf{m}$ where \mathbf{m} is the matrix containing the centres, I is identity matrix and δ is a small constant, necessary because initially $K < M$ and so the matrix $\Phi^T \Phi$ is singular.
 5. If $K < K_{max}$, $K = K + 1$ and return to 2.
-

In the simulations below, MAXCOUNT was set at 20.

6.3 Simulations

Artificial Data

With similar data as before, we get results such as in Fig. 24. We see that for a small number of latent points the mapping from latent space to data space preserves the 1-dimensional nature of the data. However the last diagram in that figure shows the mapping of 20 latent points to data space. We

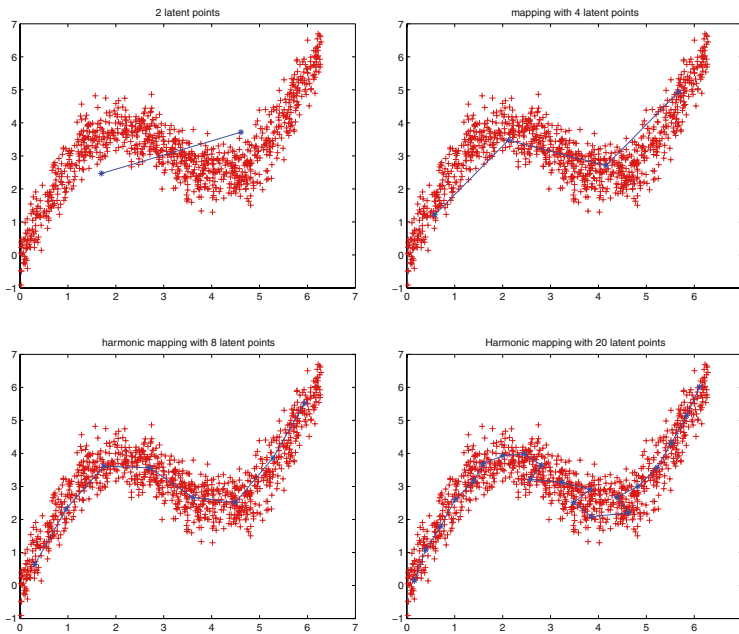


Fig. 24. The harmonic topology preserving mappings with 2, 4, 8 and 20 latent points

see that the algorithm is so eager to spread these projections about in data space that the mapping moves across the data set rather than just along the manifold. This begins to happen with about 16 latent points and becomes more pronounced as more latent points are added.

With the standard (for illustrative purposes) data set of data drawn from a uniform distribution in $[-1, 1] \times [-1, 1]$, we get the results shown in Fig. 25. We see that the mapping loses its shape fairly quickly. We consider this as evidence of an over-responsiveness to the data so that the structure of the latent space is very strongly deformed in its projection in data space.

The Algae data set

We show in Fig. 26 the projections of the above algae data set onto a two-dimensional manifold from a 10×10 harmonic topographic map. If we compare this map with the equivalent map from ToPoE (or the GTM), we see that it is far more spread out than before; the data points' projections into this space are more diffuse than before and so more of the space is being used for discriminating the data. Note that only at a single boundary between classes ('.' and '+') is the separation not linear. Also the unclassified points' projections show an interesting structure composed of a central cluster and two extruding arms, the meaning of which would have to be the subject of a biologist's investigation.

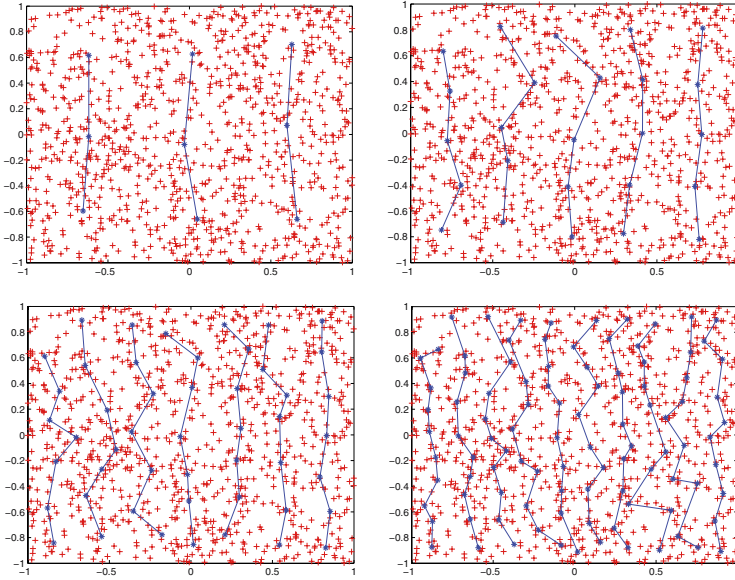


Fig. 25. The harmonic topology preserving mappings with grids of size 3×3 , 5×5 , 7×7 , and 10×10

Note that the types of algae represented by triangles and diamonds have this time been easily separated though there is one algae represented by a ‘x’ which is badly positioned. This merely emphasizes that different projections are helpful in exploratory data investigations.

6.4 Generalized Harmony Learning

[30] generalizes the above using

$$Perf_{HA} = \sum_{i=1}^N \frac{K}{\sum_{k=1}^K \frac{1}{d(\mathbf{x}_i, \mathbf{m}_k)^p}} \tag{33}$$

Then we wish to move the centres using gradient descent on this performance function

$$\frac{\partial Perf_{HA}}{\partial \mathbf{m}_k} = -K \sum_{i=1}^N \frac{2p(\mathbf{x}_i - \mathbf{m}_k)}{d(\mathbf{x}_i, \mathbf{m}_k)^{p+2} \left\{ \sum_{l=1}^K \frac{1}{d(\mathbf{x}_i, \mathbf{m}_l)^p} \right\}^2} \tag{34}$$

Solving now for \mathbf{m}_k , we get

$$\mathbf{m}_k = \frac{\sum_{i=1}^N \frac{1}{d_{i,k}^{p+2} \left(\sum_{l=1}^K \frac{1}{d_{i,l}^p} \right)^2} \mathbf{x}_i}{\sum_{i=1}^N \frac{1}{d_{i,k}^{p+2} \left(\sum_{l=1}^K \frac{1}{d_{i,l}^p} \right)^2}} \tag{35}$$

with which we can readily replace Eqn. (31) in the above algorithm.

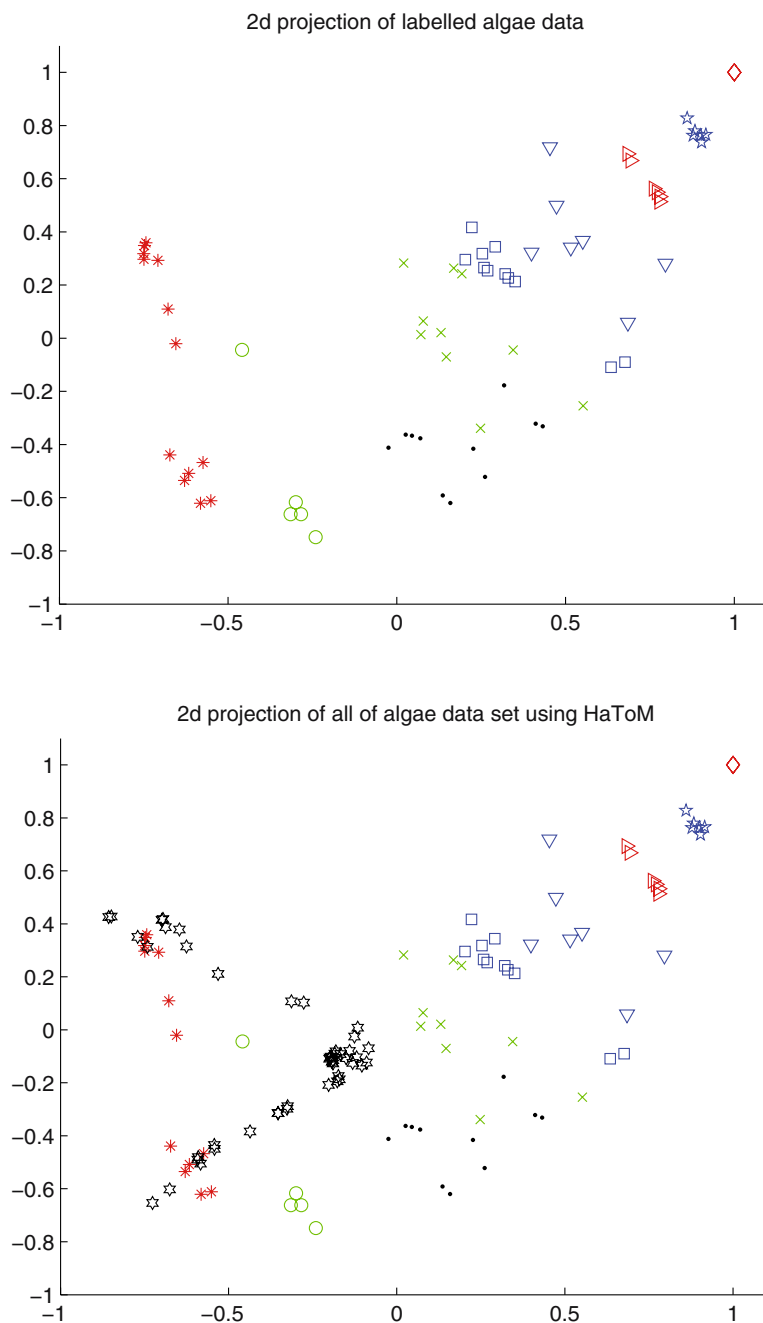


Fig. 26. *Top:* the projection of the 9 labelled classes on a harmonic mapping with a 2-dimensional set of 10×10 latent points; *bottom:* the projection of the whole data set

[30] shows how this algorithm with $p > 2$ acts like boosting for supervised learning: data points which are not well represented by the K -Harmonic Means are given greater priority in the recalculation of the positions of the means. Since the current data sets are already well covered by the HaToM, we are currently seeking especially difficult data sets to investigate this effect.

6.5 Conclusion

We have discussed a model which uses latent points which have some structure in an underlying latent space. We have investigated projecting these latent points into data space by mapping them through a nonlinear basis and then taking linear combinations of this to map a data set. We have trained the weights of this mapping by two methods:

1. The first was based on a *product-of-experts*. We trained the weights in order to maximize the probability of the data under this product-of-experts. The crucial difference between this model and other models involving products of experts is that we incorporate a responsibility term which causes the whole model to move somewhat from a pure product-of-experts to something approaching a mixture of experts. It remains however defined as a product-of-local experts.
2. The second mapping is based on the *harmonic average*. The Harmonic K -Means algorithm is extensively shown in [31] to converge to better solutions than K -means or the mixture-of-experts. For our purposes, we have shown that HaToM is more data driven than ToPoE.

The fact of being more data driven is not necessarily a good thing. If we wish to emphasize the low dimensionality of a data set, then allowing the mapping to spread may reduce insight into a low-dimensional manifold. On the other hand, we have shown with the algae data set that more insight into a data set can be achieved through diverse mappings.

Thus this Chapter has introduced two additional topographic mappings which will not replace existing mappings but will be used in addition to existing mappings to give data analysts more insight into high-dimensional data sets.

7 Conclusion

We have discussed a number of different topology preserving mappings, particularly in the context of visualization of high dimensional data. In particular, we have discussed

The Self-Organizing Map An established technique which is robust, reliable and well-used. There are many different varieties of SOM and we have not been able to provide pointers to more than a few.

The Generative Topographic Mapping A more modern probabilistic mapping which received a great deal of publicity in the late 1990s but which does not seem to be gaining many adherents in terms of researchers whose interest is in investigating specific data sets. It is possible that its more complex structure is to blame for this. However, it is also true that this mapping was sold as a principled alternative to the SOM and so researchers may feel obliged to have to make the effort to understand the probabilistic principles underlying the GTM whereas they can merely pick up the SOM and use it in an *ad hoc* manner.

The Topographic Product of Experts This mapping is relatively new and is based on probabilistic underpinnings of a product of experts rather than a mixture. It utilizes the same underlying model as the GTM but self-organizes in a different manner. Indeed, it is worth noting that the final map lies somewhat between a product and a mixture of experts, being a mixture of local products of experts.

The Harmonic Topographic Mapping This mapping uses the same underlying map as the GTM and ToPoE but recognizes that the resulting ToPoE algorithm needs no such underpinnings and so dispenses with the gradient descent method to use the underlying K -Harmonic means method. ToPoE and HaToM are our own contributions which we hope will take their place with the other mappings as alternative visualization techniques.

We have illustrated these methods on a single data set but recognize that this is far from satisfactory from the perspective of providing a truly objective comparison of these methods. One would have to analyze a large number of such data sets and then find a way of comparing the resulting projections which was based on objectivity rather than a researcher's, perhaps coloured, beliefs. One possible method, suggested in [9] is based on clustering indices but that would form the basis for a further article.

References

1. Bishop CM, Svensen M, Williams CKI (1997) GTM: the generative topographic mapping. *Neural Computation*, 10(1): 215–234.
2. Carpenter GA, Grossberg S (1987) Art 2: self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26: 4919–4930.
3. Carpenter GA, Grossberg S (1990) Art 3: hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, 3: 129–152.
4. Duda RO, Hart PE, Stork DG (2001) *Pattern Classification (2nd ed)*. Wiley-Interscience, New York, NY.
5. Fritzke B (1991) Let it grow- self-organising feature maps with problem dependent structure. In: Kohonen T, Mkisara K, Simula O, Kangas J (eds.) *Proc. Intl. Conf. Artificial Neural Networks (ICANN-91)*, Helsinki, Finland, Elsevier Science, Amsterdam, The Netherlands: 403–408.

6. Fritzke B (1993) Kohonen feature maps and growing cell structures – a performance comparison. In: Hanson SJ, Cowan JD, Giles CL (eds.) *Advances in Neural Information Processing Systems 5 (Proc. NIPS92, 30 November–3 December, Denver, CO.* Morgan Kaufmann, San Francisco, CA: 123–130.
7. Fritzke B (1993) Vector quantization with a growing and splitting elastic net. In: Gielen S, Kappen B (eds.) *Proc. Intl. Conf. Artificial Neural Networks*, 13–16 September, Amsterdam. Springer-Verlag, London, UK: 580–585.
8. Fyfe C, MacDonald D (2002) Epsilon-insensitive hebbian learning. *Neurocomputing*, 47: 35–57.
9. Garcia-Osorio C (2005) *Data mining and visualization*. PhD thesis, School of Computing, University of Paisley, Scotland, UK.
10. Garcia-Osorio C, Fyfe C (2005) The combined use of self-organising maps and Andrews’ curves. *Intl. J. Neural Systems*, 15(3): 197–206.
11. Han Y, Corchado E, Fyfe C (2004) Forecasting using twinned principal curves and twinned self organising maps. *Neurocomputing*, (57): 37–47.
12. Hastie T, Tibshirani R, Friedman J (2001) *The Elements of Statistical Learning*. Springer-Verlag, Berlin.
13. Haykin S (1994) *Neural Networks- A Comprehensive Foundation*. Macmillan, New York, NY.
14. Hinton GE (2000) Training products of experts by minimizing contrastive divergence. *Technical Report GCNU TR 2000-004*, Gatsby Computational Neuroscience Unit, University College, London (available online at <http://www.gatsby.ucl.ac.uk/> – last accessed: April 2007).
15. Hinton GE, Dayan P, Frey BJ, Neal RM (1995) The ‘wake-sleep’ algorithm for unsupervised neural networks. *Science*, 268: 1158–1161.
16. Hinton GE, Teh Y-W (2001) Discovering multiple constraints that are frequently approximately satisfied. In: Breese JS, Koller D (eds.) *Proc. 17th Conf. Uncertainty in Artificial Intelligence*, 2–5 August, Seattle, WA. Morgan Kaufmann, San Francisco, CA: 227–234.
17. Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE (1991) Adaptive mixtures of local experts. *Neural Computation*, 3: 79–87.
18. Jordan MI, Jacobs RA (1994) Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6: 181–214.
19. Kohonen T (1984) *Self-Organization and Associative Memory*. Springer-Verlag, Berlin.
20. Kohonen T (1974) An adaptive associative memory principle. *IEEE Trans. Computers*, C-23: 444–445.
21. Kohonen T (2001) *Self-Organising Maps (3rd ed)*. Springer-Verlag, Berlin.
22. Luttrell SP (1991) Code vector density in topographic mappings: Scalar case. *IEEE Trans. Neural Networks*, 2(4): 427–436.
23. MacKay DJ (2003) *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, UK.
24. Minsky M, Papert S (1969) *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, MA.
25. Nabney IT (2001) *Netlab, Algorithms for Pattern Recognition*. Springer-Verlag, Berlin.
26. Obermayer C, Sejnowski TJ (eds.) (2001) *Self-Organizing Map Formation, Foundations of Neural Computation*. MIT Press, Cambridge, MA.

27. Van Hulle M (2000) *Faithful Representations and Topographic Maps: from Distortion to Information-based Self-organization*. Wiley-Interscience, New York, NY.
28. Williams C, Agakov FV (2001) Products of gaussians and probabilistic minor components analysis. *Technical Report EDI-INF-RR-0043*, University of Edinburgh, Scotland, UK.
29. Xu L By harmony learning, structural rpcl, and topological self-organizing on mixture models. *Neural Networks*, 15: 1125–1151.
30. Zhang B (2000) Generalized k-harmonic means – boosting in unsupervised learning. *Technical Report*, HP Laboratories, Palo Alto, CA, October.
31. Zhang B, Hsu M, Dayal U (1999) k-harmonic means – a data clustering algorithm. *Technical Report*, HP Laboratories, Palo Alto, CA, October.

Resources

1 Key Books

Allinson NM, Yin H (2002) Self-organizing Maps for Pattern Recognition. In: Kaski S, Oja E (eds.) *Kohonen Maps*. Elsevier, New York, NY: 111–120.

Kohonen T (2001) *Self-Organizing Maps (3rd ed)*. Springer-Verlag, Berlin.

Kohonen T (1984) *Self-Organization and Associative Memory*. Springer-Verlag, Berlin.

Obermayer C, Sejnowski TJ (eds.) (2001) *Self-Organizing Map Formation, Foundations of Neural Computation*. MIT Press, Cambridge, MA.

Ritter H, Martinez T, Schulten K (1992) *Neural Computation and Self-organizing Maps: An Introduction*. Addison Wesley, Reading, MA.

Seiffert U, Jain LC (eds.) (2002) *Self-Organizing Neural Networks*. Springer-Verlag, Berlin.

2 Key Survey/Review Articles

Allinson NM, Opermeyer K, Yin H (eds.) (2002) *Neural Networks* (special issue on New Developments in SOMs), 15.

Bishop CM, Svensen M, Williams CKI (1997) GTM: the generative topographic mapping. *Neural Computation*, 10(1): 215–234.

Cottrell M, Verleysen M (eds.) (2006) *Neural Networks*, (Special Issue on Advances in SOMs – WSOM'05), 19(6–7): 721 – 976.

Ishikawa M, Miikkulainen R, Ritter H (eds.) (2004) *Neural Networks*, (Special Issue on New Developments in SOMs), 17(8–9): 1037–1389.

MacDonald D, Fyfe C (2000) The kernel self organizing map. In: Howlett RJ, Jain LC (eds.) *Proc. 4th Intl. Conf. Knowledge-based Intelligent Engineering Systems and Applied Technology Conf. (KES'2000)* 30 August–1 September, Brighton, UK. IEEE Press, Piscataway, NJ: 317–320.

Oja M, Kaski S, Kohonen T (2003) Bibliography of self-organizing map (SOM) papers: 1998–2001 addendum. *Neural Computing Surveys*, 3: 1–156.

3 Key Journals

Neural Computation (MIT Press).

Neural Networks (Elsevier).

IEEE Transactions on Neural Networks (IEEE).

International Journal on Neural Systems (World Scientific).

Neurocomputing (Elsevier).

Neural Processing Letters (Kluwer).

Machine Learning (Springer).

4 Key International Conferences/Workshops

ICANN – *International Conference on Artificial Neural Networks*.

ESANN – *European Symposium on Artificial Neural Networks*.

ICONIP – *International Conference on Neural Information Processing*.

IJCNN – *International Joint Conference on Neural Networks*.

ECML – *European Conference on Machine Learning*.

International Workshops on SOM:

WSOM'97: Helsinki University of Technology, Finland

WSOM'99: Helsinki University of Technology, Finland

WSOM'01: University of Lincolnshire and Humberside, UK
WSOM'03: Kyoshu Institute of Technology, Japan
WSOM'05: Université Paris I Panthéon Sorbonne, France
WSOM'07: University of Bielefeld, Germany

5 Software

We have mentioned two sets of downloadable resources in the main text. They are repeated here for convenience:

The SOM Toolbox

<http://www.cis.hut.fi/projects/somtoolbox/>

Netlab

<http://www.ncrg.aston.ac.uk/netlab/index.php>

6 Data Bases

University of California, Irvine Machine Learning Data Repository

<http://www.ics.uci.edu/~mlearn/MLSummary.html>

Complex Systems Paradigms for Integrating Intelligent Systems: A Game Theoretic Approach

Yoshiteru Ishida

Department of Knowledge-Based Information Engineering, Intelligent Sensing System Research Center, Research Center for Future Vehicle, Toyohashi University of Technology, Tempaku, Toyohashi 441-8580, Japan, ishida@tutkie.tut.ac.jp

1 Introduction

Complex systems have provided not only an analytic view that computational intelligence could be attained at a critical point (edge of chaos) where a phase transition takes place, but also a synthetic view that computational intelligence could be embedded in the field where an open and evolutionary environment for selfish agents will lead to collective phenomena. In the synthetic view, using complex systems themselves for intelligent systems, such as DNA computing (we focus on immunity-based computing in another Chapter of this volume), grid computing, and parasitic computing, is another important paradigm.

The Internet is undoubtedly the most complex and largest artifact that humankind has ever invented. Observing how the Internet has been built and evolved suggests that systems of this complexity may be built not by a usual design but by its own logic that not even the designer conceived of before its maturation. After the Internet itself became an area that allows many selfish activities, several utilities and protocols converged on what may be called the ‘Nash equilibrium’ from which no players want to deviate [19]. The game theoretic approach sheds new light on computational intelligence. That is, rather than implementing an intelligent program, one could design a field in the Internet that allows intelligent systems to emerge as the Nash equilibrium of the Internet. Further, game theoretic and computational approaches to the Internet (see, for example, [6, 7, 15, 17, 23, 25]) reveal that it is computationally difficult to obtain a Nash equilibrium. Conversely, this fact suggests that a computationally difficult task could be solved by selfish agents [8, 20]. Resource allocation, for example, which is computationally tough, is solved by a market mechanism in which many selfish agents participate [29].

This Chapter investigates the first step towards embedding computational intelligence in the Internet field by selfish agents, namely, whether selfish agents can ever cooperate and converge on some tasks. Selfish routing and task allocation have been studied extensively in the computational game community, but can intelligent tasks be done or can agents ever take care of themselves in the first place? We first pose the problem of self maintenance in an agent population, and then use a game theoretic approach to test whether cooperation would occur or under what conditions cooperation will occur.

The above-mentioned research focused on algorithms and computational complexity for obtaining equilibrium when selfish agents compete for resources, or the Nash equilibrium as a convenient substitute. The cost for the Nash equilibrium relative to the optimized solution has also been discussed to measure the cost of ‘anarchy’ [15]. Instead, this Chapter focuses on the self maintenance task, self-repairs by mutual copying in particular, and discusses when selfish agents begin to cooperate. We extend the discussion to when these selfish agents (called ‘selfishware’) organize themselves to mutually supporting collectives (called ‘Internet being’).

The present research is significant in two respects: one is engineering and the other is theoretical. For engineering significance, a computing paradigm such as grid computing [9] and parasitic computing becomes the background. When grid computing becomes dominant for large-scale computing, what we call *agents* (autonomous programs that can passively move from nodes to nodes) will become like *processes* in the Unix OS. One important difference is that agents (or what we call later *selfishware*) are selfish, and will not be organized with central authority as is done in a conventional OS. Therefore, an organization of selfish agents will become an organization with weakest central authority, or even with distributed authority, as seen in the free market economy. Naturally, information processing with selfish agents will be imperative, thus making the game theoretic approach and economic approach – such as selfish task allocation and routing – important.

Also of significance is that it will provide an organizational view for artificial life or what we call an ‘Internet being’ (a life-like form which has some identity and hence boundary). Self-organization of selfish agents will be more than a mere collection of independent agents, but rather an organization of cooperative agents. This would reveal an intrinsic logic and process that selfish agents form multi-agent organisms, similarly to multi-cellular organisms. The game theoretic approach would provide conditions and mechanisms for defective selfish agents to develop into cooperative selfish agents when payoffs are recast in a broader context of time and space.

After a brief introduction in Sect. 2, a microscopic analysis focusing on interactions between two agents will be presented in Sect. 3. Section 4 deals with a macroscopic model with many networked agents. In both the micro and macro models, the importance of involving neighbor agents in each agent’s

payoff is stressed. Section 5 briefly discusses the significance of the game theoretic approach to large-scale complex systems such as the Internet. The game theoretic approach is imperative, since autonomous and distributed control and management is inevitable for such complex and large-scale systems.

2 Economic Theory for the Internet Being with Selfish Agents

The game theoretic approach has demonstrated its power in the field of economics and biology. The Internet has already reached a level of complexity comparable to that of economic and biological systems. Moreover, the agent approach permits a structural similarity where selfish individuals (in the free market of the economic system) and selfish genes (in biological systems) cooperate or defect in an open network where many options have been left undetermined before the convergence.

The economic approach has been actively studied in the distributed artificial intelligence community (for instance, [27, 29]), and its application to auctions is a successful example (such as [24]). The economic approach, and the game theoretic approach in particular, has been extensively studied in the algorithm and computation community and has had an impact on network applications. Rigorous arguments with equilibrium concepts – the Nash equilibrium, among others – are providing a framework ground theory for the economic aspects of the Internet. The cost of selfish routing has been estimated by examining how bad is the equilibrium to which selfish routing might converge (that is, the Nash equilibrium from where no one wants to deviate) relative to the optimal solution. In the seminal paper by [23], the TCP/IP protocol is recast as the Nash equilibrium and an economic model that allows TCP/IP as an equilibrium point has been called for as an open problem. That is, design an Internet model where TCP/IP is the Nash equilibrium in a space of available protocols. Protocols such as TCP [1], Aloha, CDMA and CSMA/CA have been studied. Packet forwarding strategies in wireless ad hoc networks can also be recast in the framework. Network intrusion detection has also been investigated [16] within the framework of a two-player game: ‘intruder’ and ‘defender’.

What has been computed by a market mechanism or more generally by a collection of selfish agents turned out to be difficult when attempted by computation (a typical example is prices of commodities as an index for resource allocation). This fact indicates that the market economy – or more generally free and hence selfish agents properly networked – has a potential for computing something that could be difficult when approached otherwise. Also, some cases in which a planned economy perturbed by a market economy resulted in eradication of the planned economy by the market economy indicates that the market economy may be ‘evolutionarily stable’ within these economic systems.

This fact further indicates that a problem solving framework by properly networked selfish agents may have some advantage over other usual problem solving frameworks, such as the one organized with central authority. Also, solutions can be obtained almost free or as a byproduct of the problem solving mechanism, or solutions that are almost inseparably embedded in the solving mechanism. The above two observations encourage us to recast problems which have been known to be computationally difficult or problems which are difficult to even properly define and approach, such as attaining intelligent systems.

Mechanism Design, a subfield of Economics, has been studied [20] and has recently been extended to Algorithmic Mechanism Design [6] and to Distributed Algorithmic Mechanism Design [8].

Studies on computational intelligence by agents usually assume that agents can be autonomous, hence allowing different rules of interactions – in other words, heterogeneous agents. We further assume that agents are selfish in the sense that they will try to maximize the payoff for themselves. Thus, agents are broader than a program (or software), and they involve users that are committed to the agents. We use the word ‘selfishware’ to include not only programs but also the humans (end-point users and providers running autonomous systems for the Internet) behind the programs. The organization that would appear to be a collective interplay with *selfishware* will be called an ‘Internet being’. It can be an entity that can perform some tasks that require intelligence. Spam email, computer viruses and worms may not be called Internet beings, because they are obviously not mutually supporting collectives (with exceptions, such as Distributed Denial-of-Service – DDOS attacks), although they are guided by selfishware. They are rather parasitic lone wolves.

The idea developed here can apply not only to the Internet but also to other information networks, such as sensor networks, as long as they are put in the model. The models dealt with in this Chapter have the following components:

- M1. *States*: agents have two states (0 for normal; 1 for faulty). The state will be determined by the action and state of interacting agents.
- M2. *Actions*: agents have two actions (*C* for cooperation; *D* for defection).
- M3. *Network*: agents are connected by a network and agents can act only on the connected agents (neighbor agents).

Actions may be controlled uniformly (Sect. 4.1) or may be determined by the acting agent itself in the selfish agent framework (Sect. 4.4), such that the payoff assigned to each agent will be maximized. A network may be defined explicitly with a graph or implicitly by specifying the neighbor agents (for example, lattice structure as in cellular automata and dynamical network as in scale-free networks).

Since we focus on the self-maintenance task by mutual repair, cooperation and defection correspond to repairing and not repairing, respectively. As we have remarked, we do not assume recognition of the states of target agents before actions, since repairing could harm the target agents, particularly if the acting agents themselves are faulty: this is what we call a ‘double-edged sword’.

In the agent-based approach of this Chapter, we place the following restrictions on our view which we also placed in perusing immunity-based systems, since an autonomous and distributed character is similar to them.

- *Local information:* for each immune cell mounting a receptor or a receptor itself (antibody), only matching or not (some quantitative information on degree of matching is allowed) can be provided as information.
- *No a priori labeling:* for an immune cell or antibody, an antigen is labeled neither as ‘antigen’ nor as ‘nonself’.

Because of these two restrictions and because we do not assume recognition (in contrast to recognition centered in immunity-based systems in Chap. 25 of this Handbook) of the states of target agents before actions, we face the double-edged sword problem in this Chapter, since the effector part (repairing by copying) could harm rather than cure, based only on local information. This problem may be more significant than that of immunity-based systems because we do not assume recognition capability (that could avoid adverse effects) here as in immunity-based systems; actions of agents are motivated by selfishness (payoff) rather than the state of the target.

In Sect. 3, we use a Markov model used for reliability theory as a microscopic model, and probabilistic cellular automata used for percolation theory as a macroscopic model. Both models incorporate M1, M2 and M3, as above. While the microscopic model in Sect. 4 focuses on the incentive for cooperation retaining a simple network with only two agents, the macroscopic model deals with situations where faulty agents are eradicated by comparing agents controlled with central authority and selfish agents.

3 A Microscopic Model: Negotiation Between Agents

3.1 The Prisoner’s Dilemma

In solving the problem of cleaning a contaminated network by mutual copying, another problem (other than the double-edged sword) is that each autonomous (and hence selfish) node may not repair others and fall into a deadlock waiting for other nodes to be repaired. The situation is similar to that of the Prisoner’s Dilemma (PD) that has been well studied in game theory and has been applied to many fields.

Table 1. The payoff matrix of the Prisoner's Dilemma: R, S, T, P are payoffs to agent-1

	C (Agent 2)	D (Agent 2)
C (Agent 1)	R (reward)	S (sucker)
D (Agent 1)	T (temptation)	P (punishment)

The Prisoner's Dilemma (PD) [5] is a game played just once by two agents with two actions (cooperation C , or defect D). Each agent receives a payoff (R, T, S, P) (Table 1) where $T > R > P > S$ and $2R > T + S$.

In the Iterated Prisoner's Dilemma (IPD) [2], each iterated action is evaluated. In the Spatial Prisoner's Dilemma (SPD) [21], each site in a two-dimensional lattice corresponding to an agent plays PD with the neighbors, and changes its action according to the total score it received.

3.2 Repairing from Outside the System: A Conventional Model [12]

Consider a model with only two agents i ($i = 1, 2$) that become faulty and thence repaired. Using conventional notations in reliability theory, ν and μ indicate the rate for becoming faulty and the rate for repair, respectively. When the repair is done from outside the system (by repairmen, for example), the state-transition diagram as a Markov model is as shown in Fig. 1 (for example, [26]). The corresponding Kolmogorov equation is:

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{M}\mathbf{P}(t) \quad (1)$$

where the time dependent vector variable $\mathbf{P}(t) = (p_{00}(t), p_{01}(t), p_{10}(t), p_{11}(t))^T$ comprises a component $p_{s_1, s_2}(t)$ denoting the probability of agent-1 being s_1 and agent-2 being s_2 at time t , where $s_1, s_2 \in \{0, 1\}$ (0: normal; 1: abnormal). The matrix \mathbf{M} is a transition matrix corresponding to the state-transition diagram shown in Fig. 1.

$$\mathbf{M} = \begin{pmatrix} -2\lambda & \mu & \mu & 0 \\ \lambda & -\lambda & 0 & \mu \\ \lambda & 0 & -\lambda & \mu \\ 0 & \lambda & \lambda & -2\mu \end{pmatrix} \quad (2)$$

3.3 Mutual Repair within Systems

When the repair is done by these two agents mutually, some complications occur. The double-edged sword framework allows agents that are capable of

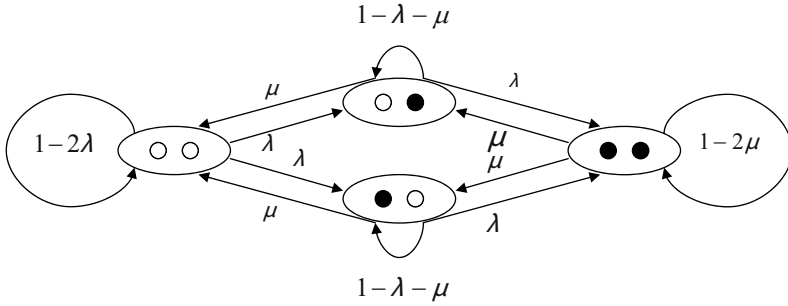


Fig. 1. State-transition diagram for the conventional Markov state diagram for availability for ordinary repairing from outside the system; white circles indicate normal nodes and black ones abnormal nodes

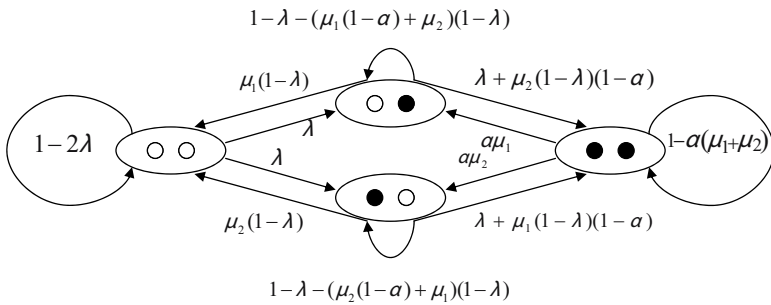


Fig. 2. State-transition diagram for the mutually-repairing two-agent system; white circles indicate normal nodes and black ones abnormal nodes [12]

repairing other agents, but when the repairing agents are themselves faulty they will cause the target agents to become faulty (infect contamination) rather than repair them. Thus the state-transition diagram as a Markov model is as shown in Fig. 2 [12]. Let μ_i denote the repair done by the agent i , and α (<1) indicate the success rate when repair is done by a faulty agent. In this model, a transition matrix \mathbf{M} corresponding to the state-transition diagram is as follows:

$$\mathbf{M} = \begin{pmatrix} -2\lambda & \mu_1(1-\lambda) & \mu_2(1-\lambda) & 0 \\ \lambda & -\lambda - (\mu_2(1-\alpha) + \mu_1)(1-\lambda) & 0 & \alpha\mu_2 \\ \lambda & 0 & -\lambda - (\mu_1(1-\alpha) + \mu_2)(1-\lambda) & \alpha\mu_1 \\ 0 & \lambda + \mu_2(1-\lambda)(1-\alpha) & \lambda + \mu_1(1-\lambda)(1-\alpha) & -\alpha(\mu_1 + \mu_2) \end{pmatrix} \quad (3)$$

3.4 Mutual Repair with Selfish Agents

For a game theoretic argument, it is further assumed that an agent must decide whether it will repair others or not, corresponding to cooperation and defection in the Prisoner’s Dilemma. For agent i , $C_i = 1$ if it repairs other agents, and 0 otherwise. Let $P_i(C_1, C_2)$ denote the probability of agent i being

Table 2. Steady-state reliability of each agent when mutual repair is involved

	$C_2 = 1$	$C_2 = 0$
$C_1 = 1$	$P_1(1, 1) = \frac{\beta}{\lambda(\lambda + \frac{\beta}{\alpha\mu}) + \beta}$	$P_1(1, 0) = 0$
	$P_2(1, 1) = P_1(1, 1)$	$P_2(1, 0) = P_1(0, 1)$
$C_1 = 0$	$P_1(0, 1) = \frac{1}{\lambda + \frac{\beta}{\alpha\mu}}$	$P_1(0, 0) = P_2(0, 0) = 0$
	$P_2(0, 1) = P_1(1, 0)$	

alive when agent i 's action is C_i . Simple calculation yields the steady-state probability $P_i(C_1, C_2)$ in Table 2 below, arranged as per Table 1 [12].

Table 2 can be regarded as a payoff matrix of the two-player game where each ij entry indicates the payoff that player i gains. If we simply regard $P_i(C_1, C_2)$ as agent i 's payoff when actions C_1, C_2 are taken, mutual repairing would happen due to the inequalities: $P_1(1, 1) > P_1(0, 1) > P_1(1, 0) = P_1(0, 0)$; $P_2(1, 1) > P_2(1, 0) > P_2(0, 1) = P_2(0, 0)$.

While the self action does not make any difference (for instance, for agent 1, $P_1(1, 0) = P_1(0, 0)$) when the other agent does not cooperate, the agent should certainly cooperate when the other agent cooperates (for example, for agent 1, $P_1(1, 1) = P_1(0, 1)$). This raises the reliability of others, making the repairing of self by others more effective and having a cyclic effect.

Let us take the cost of repairing into consideration. Then agent 1, for example, will choose its action C_1 to maximize $P_1(C_1, C_2) - c \cdot C_1$, where c is the cost of repairing relative to the benefit measured by the reliability of itself. Involving the cost for cooperation would naturally bias the situation towards more defect-benefiting. When the opponent defects, the agent simply loses the cost of cooperation if it cooperates. However, there is still a chance for mutual cooperation when the opponent cooperates, so $P_1(1, 1) - c > P_1(0, 1)$ holds when the cost relative to benefit satisfies:

$$\frac{(\beta - \lambda)(\lambda + \frac{\beta}{\alpha\mu}) - \beta}{(\lambda(\lambda + \frac{\beta}{\alpha\mu}) + \beta)(\lambda + \frac{\beta}{\alpha\mu})} > c \tag{4}$$

Selfishness of an agent is reflected on the objective function that the agent will maximize, but this reflection is not a trivial task. The above agents are a short-sighted implementation of selfishness; a more foresighted agent would consider the event of other agent failure as losing the chance of being repaired by the agent, and the extinction of all agents as a fatal event that should be avoided by paying a high cost. In the above model where repairing by faulty agents does not happen, extinction of alive agents is an absorbing state from which no other state arises.

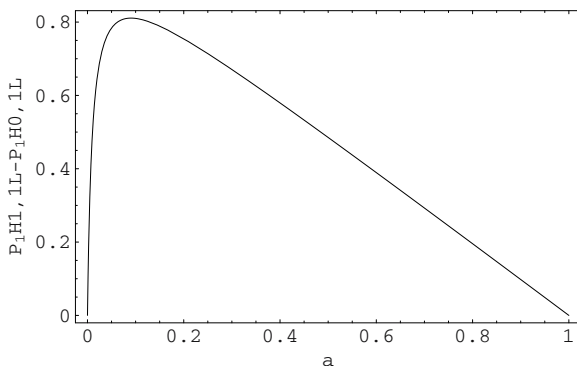


Fig. 3. Plot of the difference $P_1(1,1) - P_1(0,1)$ when the repair success rate by abnormal agents α changes from 0 to 1, and $\lambda = 10^{-4}$, $\mu = 10^2\lambda$ are fixed [12]

Table 3. Steady-state availability AV

	$C_2 = 1$	$C_2 = 0$
$C_1 = 1$	$AV(1,1) = \frac{\beta + \lambda}{\lambda(\lambda + \frac{\beta}{\alpha\mu}) + \beta}$	$AV(1,0) = \frac{1}{\lambda + \frac{\beta}{\alpha\mu}}$
$C_1 = 0$	$AV(0,1) = \frac{1}{\lambda + \frac{\beta}{\alpha\mu}}$	$AV(0,0) = 0$

Figure 3 plots the difference $P_1(1,1) - P_1(0,1)$ when the repair success rate by abnormal agent α changes from 0 to 1 and $\lambda = 10^{-4}$, $\mu = 10^2\lambda$ are fixed [12]. There is a strong incentive for agent 1 to cooperate when the success rate is about 0.1. The incentive decreases linearly when the rate exceeds 0.2, which indicates that reliable repairs by abnormal agents promote cooperation.

If the availability (the probability that at least one agent remains normal) is used as a payoff for each agent, then there will be stronger incentive to cooperate when the other agents cooperate, since the difference $AV(1,1) - AV(0,1)$ is larger than the difference $P_1(1,1) - P_1(0,1)$, as shown in Table 3.

This indicates that even for selfish agents, they will be more likely to cooperate if they take a systemic payoff that evaluates the cost and benefit in a more system wide and longer term; this is the beginning of self-organization to mutual supporting collectives.

4 A Macroscopic Model: Boundary Formation among Agents

4.1 A Model with Uniform Control

We consider the possibility of cleaning up the network by mutual copying. Repair by copying in information systems is also a double-edged sword and it should be identified under what condition the network can eradicate abnormal

elements from the system. We consider a probabilistic cellular automata (PCA) to model the situation where computers in a local area network (LAN) mutually repair by copying their content. Since the problem involves the double-edged sword leading to a critical phenomenon, repairs have to be decided giving consideration to the resources used and remaining in the system and the network environment.

As a first macro model, we use a PCA [11]; we assume the actions are done in a synchronous fashion. As in the micro model, the repairing may be done by copying its content to the other agents. Further, the network for the model considered in this Section is restricted to the one-dimensional array shown in Fig. 4 (which could be an n -dimensional array, a complete graph, a random graph, or even a scale-free network) that could have S neighbors for each agent with a boundary condition – in other words, the structure of the array is a ring with agent 1 adjacent to agent N .

Also, a probabilistic cellular automaton requires probabilistic rules for interactions. The model of the current Section controls the repairing of all the agents uniformly. That is, each agent tries to repair its neighbor agents in a synchronous fashion with a probability μ (repair rate). Repair will be successful with probability α_0 when it is done by a normal agent, but with probability α when done by an abnormal agent ($\alpha < \alpha_0$). The repaired agents will be normal when all repairs are successful. Thus, when repairing is performed by the two neighbor agents, both of these two repairs must be successful in order for the repaired agent to be normal.

As a probabilistic cellular automaton, the transition rules are shown in Table 4. The self-state is the center in parentheses, and the two neighbor states to the left and right; the self-state will be changed to the state indicated to the right of the arrow.



Fig. 4. One-dimensional array with two states: normal (0) and abnormal (1)

Table 4. State change rules in the probabilistic cellular automaton

State change	Probability
(000) → 1	$\mu(1 - \alpha_0)(2 - \mu(1 - \alpha_0))$
(001) → 1	$\mu^2(1 - \alpha\alpha_0) + \mu(1 - \mu)((1 - \alpha) + (1 - \alpha_0))$
(101) → 1	$\mu(1 - \alpha)(2 - \mu(1 - \alpha))$
(010) → 1	$1 - \mu\alpha_0(2(1 - \mu) + \mu\alpha_0)$
(011) → 1	$1 - \mu((\alpha + \alpha_0)(1 - \mu) + \mu\alpha\alpha_0)$
(111) → 1	$1 - \mu\alpha(\mu\alpha + 2(1 - \mu))$

Table 5. State change rules in the PCA when $\alpha_0 = 1$

State change	Probability
(000) \rightarrow 1	0
(001) \rightarrow 1	$\mu(1 - \alpha)$
(101) \rightarrow 1	$\mu(1 - \alpha)(2 - \mu(1 - \alpha))$
(010) \rightarrow 1	$(1 - \mu)^2$
(011) \rightarrow 1	$(1 - \mu)^2 + \mu(1 - \alpha)$
(111) \rightarrow 1	$\mu(1 - \alpha)(2 - \mu(1 - \alpha)) + (1 - \mu)^2$

Table 6. Rules for the DK model, where p_1 and p_2 are two parameters for the DK model, and the symbol * is a wildcard

State change	Probability
(0*0) \rightarrow 0	1
(0*1) \rightarrow 1	p_1
(1*1) \rightarrow 1	p_2

When the repair rate by normal agents $\alpha_0 = 1$, the probability in the change rule can be reduced greatly (Table 5). The relation among these change probabilities is obvious – for instance, the probability for the state change (111) \rightarrow 1 can be obtained by adding those for (101) \rightarrow 1 and (001) \rightarrow 1; also the probability for the state change (011) \rightarrow 1 can be obtained by adding those for (010) \rightarrow 1 and (001) \rightarrow 1.

The Domany-Kinzel (DK) model [4] is a one-dimensional, two-state and totalistic probabilistic cellular automaton (PCA) in which the interaction timing is specific. The interaction is done in an alternated synchronous fashion: the origin cell with state 1 is numbered as 0. The numbering proceeds $\{1, 2, \dots\}$ to the right, and $\{-1, -2, \dots\}$ to the left. At the N -th step the even numbered cells will act on the odd numbered cells, and the odd numbered cells will act at the next step. The neighbor is two cells adjacent to oneself without self-interaction. The interaction rule is as shown in Table 6.

Our PCA model can be equated with the DK model [11] when $\mu = 1$ (namely, agents always repair), with parameters $p_1 = (1 - \alpha)$, $p_2 = (1 - \alpha^2)$; that is, the case of the directed bond percolation.

Under the approximation that the probability that the state of agent 0 is a constant p_0 (mean field approximation and steady state), the following steady state probability of p_0 is obtained:

$$\frac{dy}{dt} = ay^2 + by + c \quad (5)$$

where $a = -\mu^2(\alpha_0 - \alpha)^2$, $b = -2\mu(1 - \alpha_0)(-\mu(\alpha_0 - \alpha) + 1) + \mu(\mu - 2\alpha)$, and $c = \mu(1 - \alpha_0)(2 - \mu(1 - \alpha_0))$.

When $\alpha_0 > \alpha$ (hence $\alpha < 0$), the steady state can be obtained as follows:

$$p_0 = 1 + \frac{1}{2a}(b + \sqrt{b^2 - 4ac}) \tag{6}$$

When $\alpha_0 = 1$, the above form reduces to $a = -\mu^2(1 - \alpha)^2, b = \mu(\mu - 2\alpha)$, and $c = 0$, and hence

$$p_0 = 1 + \frac{\alpha(2(1 - \mu) + \alpha\mu)}{\mu(1 - \alpha)^2} \tag{7}$$

In order for abnormal nodes to be eradicated, c must be 0 (that is, $\alpha_0 = 1$), otherwise normal nodes could have spread abnormal states. When $c = 0$, the following threshold condition must be satisfied for eradication of abnormal agents, since the time derivative $\frac{dy}{dt}$ must be negative in the equation above.

$$2\alpha \geq \mu \tag{8}$$

This steady state probability also matches qualitatively with the above simulation results (Fig. 5), however, simulations are needed when the result by mean field analysis does not match well with the simulation result when $\alpha_0 = 1$ (Figs. 5 and 6). Here the size of lattice is 20×20 , hence the number of agents is 400. Simulation results qualitatively match the steady state fraction of normal agents obtained by the mean field analysis above. When the repair success rate by normal agents $\alpha_0 = 1$, the result by mean field analysis does not closely match the simulation result.

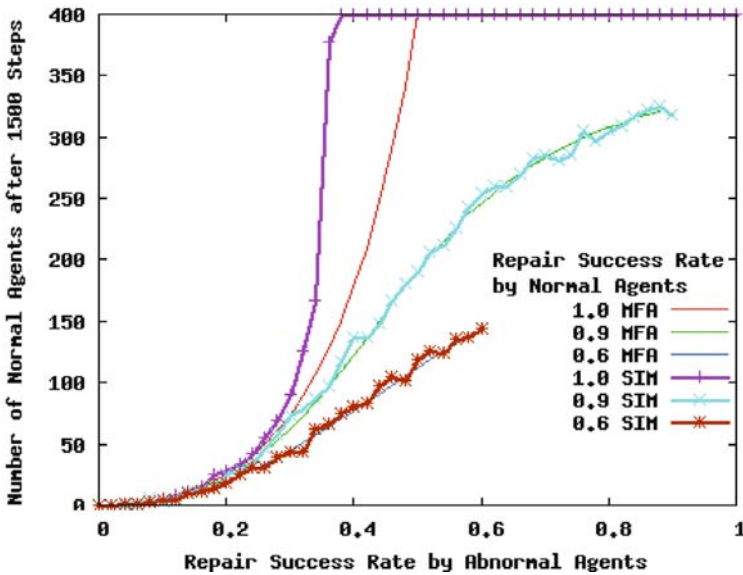


Fig. 5. The number of normal agents after 1500 steps, when the repair success rate by normal agents α varies

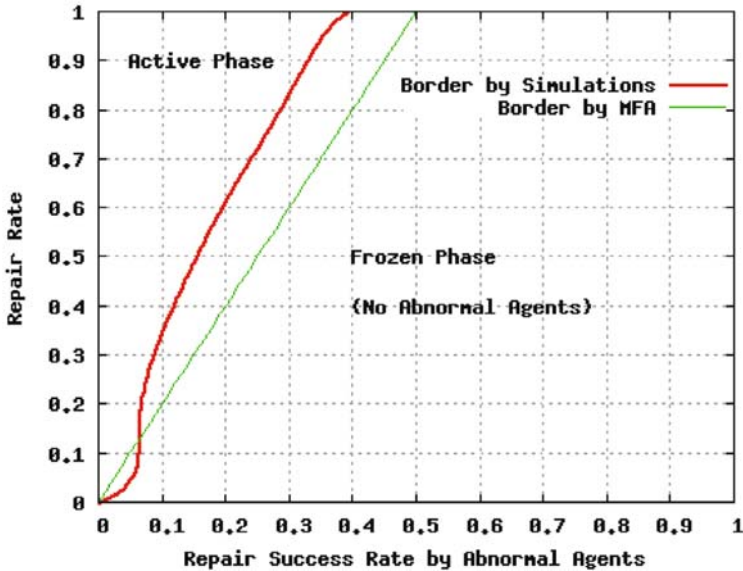


Fig. 6. Frozen (the right region where all the units are normal), and active phases (the left region where some units remain abnormal) when $\alpha_2 = 1$ [11]

4.2 The Spatial Prisoner's Dilemma

The Spatial Prisoner's Dilemma (SPD) has been studied to investigate when, how, and why cooperation emerges among selfish agents when they are spatially arranged, hence interactions are limited only to their neighbors. In SPD pioneered by [21], each player was placed at each lattice of the two-dimensional lattice. Each player has an action and a strategy, and receives a score. Each player plays PD with the neighbors, and changes its strategy to the strategy that earns the highest total score among the neighbors. We will use this deterministic SPD. In the stochastic version, the agent will decide its action based on a probability proportional to the difference between its own payoff and the highest payoff in the neighbors' agents (similarly to replicator dynamics [10, 28]).

The SPD is generalized by introducing a spatial strategy [13], which determines the next action dependent upon the spatial pattern of actions in the neighbors. A score is calculated by summing up all the scores received from PD with 8 neighbor players. After r (strategy update cycle) steps of interactions with neighbors, the strategy will be chosen from a strategy with the highest score among the neighbors.

To specify a spatial strategy, the actions of all the neighbors and the player itself must be specified. For simplicity, we restrict ourselves to a 'totalistic

spatial strategy' that depends on the number of D (defect) actions of the neighbor, not on their positions.

4.3 A Model with Selfish Agents

Although the actions of agents in the above models are controlled uniformly by the parameter μ , selfish agents in the current model will determine their actions by accounting their payoffs. To implement this selfish framework, we use spatial strategies in the Spatial Prisoner's Dilemma.

The self-repairing network consists of agents capable of repairing other agents connected to them. In the probabilistic cellular automaton model of Sect. 4.1, agents do not have a failure rate and do not become abnormal by themselves, however, the agents in the present model [14] incorporate a failure rate (λ). Repairing is controlled by repair rate (μ). When a repair is carried out, it will be successful with a repair success rate (α), and the repaired agents are rendered normal.

The adverse impact caused by the abnormal agents is implemented by raising the failure rate (by an amount according to the damage rate δ) of the repaired agents (when repaired by abnormal agents). Further, the agents are assumed to use some resources (R_λ) for repair. This amounts to a cost for cooperation, and hence motivates selfish agents to engage in free-riding. The agents have to perform the tasks assigned to them, but *without* performing repairs. Abnormal agents increase and the performance of the system decreases – hence we are faced with a dilemma. An agent is able to repair more than one other agent, provided that the quantity of maximum resource R_{max} is not exceeded. We consider the available resource (the resource not used for repair) as the agent's 'score'. Throughout this Chapter, simulations are conducted using the parameters listed in Table 7.

Table 7. Parameter list

	Description	Value
$L \times L$	size of the space	50×50
N	number of agents	2500
$N_f(0)$	initial number of abnormal agents	100
λ	failure rate	0.01
μ	repair rate	0.01
α	repair success rate	0.1
δ	damage rate	0.1
r	strategy update cycle	100
R_{max}	maximum number of resources	25
R_λ	number of resources used for repairing	1

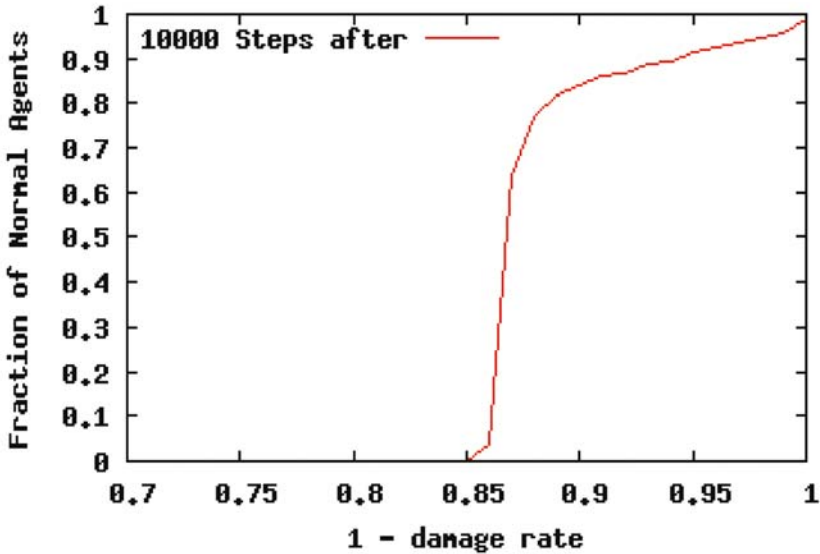


Fig. 7. The fraction of normal agents when the damage rate δ varies (parameters are as per Table 7, except failure rate 0.001, repair rate 1.0, repair success rate 0.01, and the damage rate varies; a random selection of 100 agents is initially made abnormal [22])

Simulations are conducted in a 2-dimensional lattice. To contrast the results with the selfish repair rate control in Sect. 3, simulations are conducted for the above self-repair network with a uniform repair rate. This model has a threshold for the damage rate δ (Fig. 7) [22], as expected from the probabilistic cellular automaton in Sect. 4.1. Above the damage rate threshold, all the agents become abnormal.

To contrast with the one with systemic payoff, we use only two trivial strategies: All-*C* and All-*D*; we have reported elsewhere [22] on the use of nontrivial spatial strategies such as *k-C*. In the simulation shown in Fig. 8, All-*D* will eradicate All-*C* strategies; hence all agents will remain silent without repairing any other agents. Thus, eventually all the agents will be abnormal with a positive failure rate. In Sect. 4.4, we will modify the payoff, incorporating not only its own remaining resources but also all neighbor resources.

4.4 Strategic Repair with Systemic Payoff

As in the simulation (Fig. 8) [22], a repair control by allowing agents to take only All-*C* (repair) or All-*D* (not repair) resulted in all silent agents, and hence ended with all abnormal agents. Here, the payoff is modified to include all the neighbor resources. This modified payoff has the impact of making agents

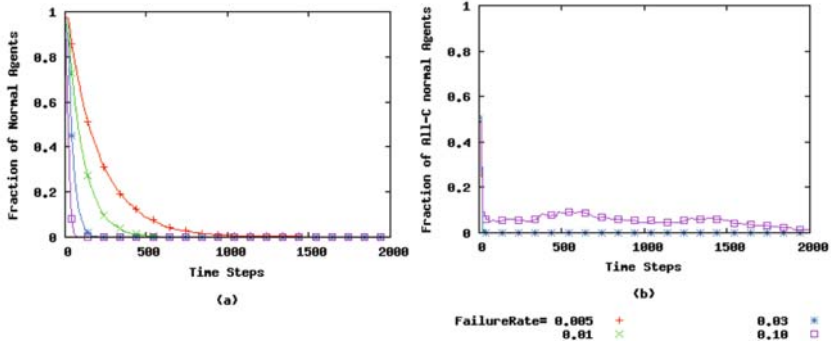


Fig. 8. SPD with simple payoff measured by available agent resources (parameters are: failure rate 0.005–0.10, repair success rate 0.1, damage rate 0.1, strategy update cycle 20, max resources 9, cost for repair 1. 100 agents are made randomly abnormal initially, and half chosen at random to take all-*D* [22])

more attentive by caring for neighbor agents that might possibly repair them in the future.

Simulations were conducted for strategic repair with modified payoff: not only the remaining agent resources but also resources of the neighboring agents were added to the payoff. Figure 9 plots (a) the time evolution of the fraction of normal agents, (b) the available resources left in the system, and (c) the fraction of agents with All-*C* [22].

It can be observed that this strategic repair with modified payoff can adapt to the failure rate: when the failure rate is low, the fraction of All-*C* agents is kept small (Fig. 9(c)) limiting unnecessary repair, whereas when the failure rate is high, the fraction of All-*C* agents is also made high. As a result of this flexible change of repair rate, the fraction of normal agents (Fig. 9(a)) as well as available resources (Fig. 9(b)) are made stable and the difference in failure rate is absorbed.

4.5 Comparison Between Uniform Repair and Strategic Repair

An advantage of strategic control with SPD is that agents can switch between repair and not-repair adapting to the spatial environment around the agents. Strategic control with modified payoff (where the available resources of neighboring agents are added to the payoff) has been compared with control by a uniform rate in a spatially heterogeneous environment, where the lattice space is divided into two regions: the right region with high failure rate ($\lambda = 0.1$), and the left region with low failure rate ($\lambda = 0.001$) (Fig. 10). Black indicates a normal cooperator, white an abnormal cooperator, light gray a normal defector, and dark gray an abnormal defector. It can be observed that cooperators

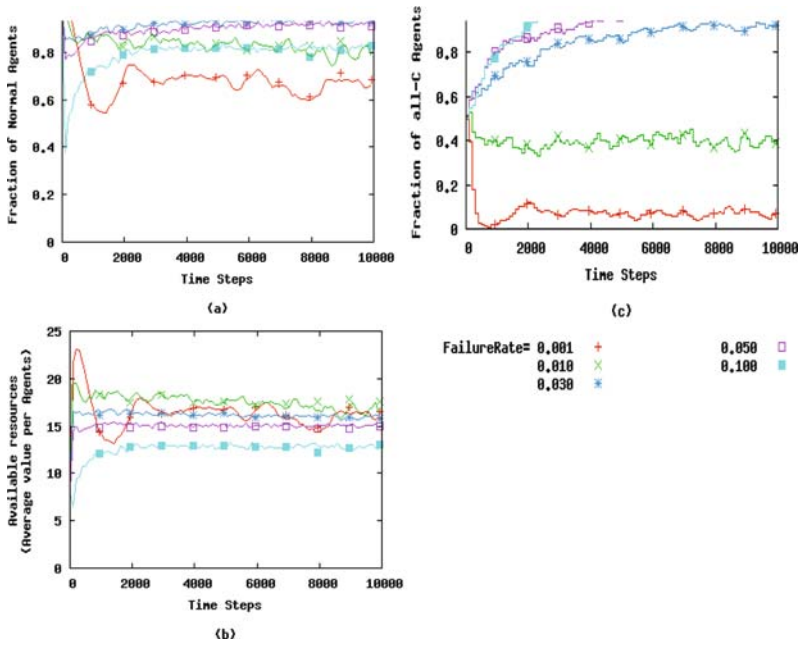


Fig. 9. SPD with strategic control with modified payoff (available resources of the neighbor agents being added to the payoff): (a) fraction of normal agents, (b) available resources, (c) fraction of All-C agents (parameters are as per Table 7, and the initial configuration is with half of All-D agents and 100 failure agents chosen at random [22])



Fig. 10. Snapshot of agent configurations at 1400 time steps when simulation is carried out with the parameters listed in Table 7, except for failure rate and repair rate $\mu = 0.5$, where strategic repair (*left*) and uniform repair (*right*) are compared

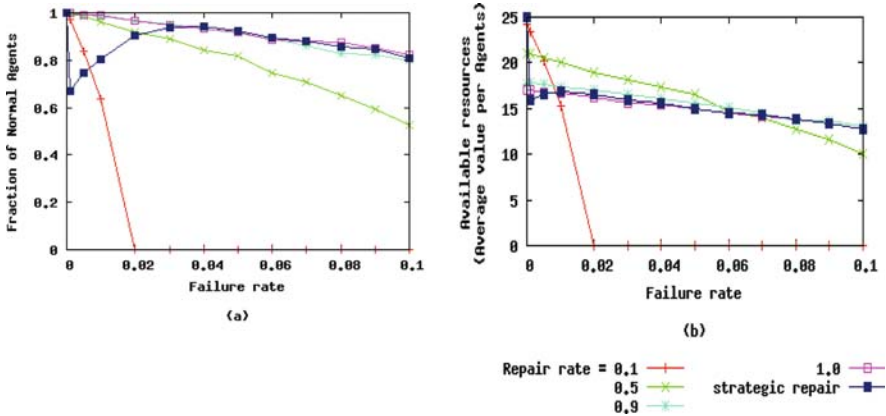


Fig. 11. Maximum number of resources is 25 – comparison between strategic control with modified payoff (available resources of neighboring agents being added to the payoff), and control with uniform rate: (a) fraction of normal agents, and (b) available resources when the failure rate λ varies [22]

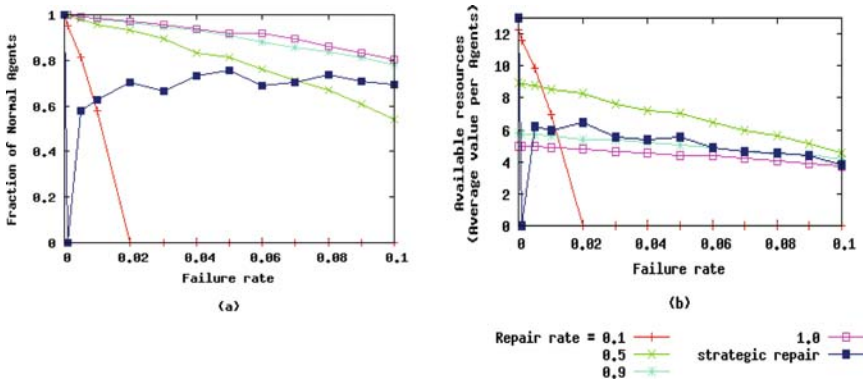


Fig. 12. Maximum number of resources is 13 – comparison between strategic control with modified payoff (available resources of neighboring agents being added to the payoff), and control with uniform rate: (a) fraction of normal agents, and (b) available resources when the failure rate λ varies [22]

(black and white agents) are found in the right region with high failure rate in the strategic repair (left), while no such adaptive behavior is observed for uniform repair.

Further, in spatially homogeneous environment, strategic control with modified payoff has been compared with control by a uniform rate. Figures 11 and 12 are simulation results for varying values of max resource: 25 and 13, respectively [22]. Changes of the max resource will change the relative cost of repair. In each figure, the fraction of normal agents (a) as well as the available

resources (b) are monitored; available resources, which are correlated with the fraction of normal agents, are a rough measure of performance.

First, it can be observed that the performance of the uniform rate control varies in these three simulations, while that of the strategic rate control shows reasonable performance. For example, the available resources by the uniform rate control with repair rate 0.5 is worst when the failure rate is 0.1 and max resource is 25 (Fig. 11(b)), however it is the best when max resource is 12 (Fig. 12(b)). Thus, the performance comparison between uniform and strategic rate control can be summarized as:

- the strategic rate control is neither best nor worst;
- strategic control is robust against parameter changes.

The simulations indicate that an appropriate uniform rate could be set when parameters were correctly identified. However, it is often the case that parameters are difficult to identify, or that they may change dynamically. In such cases, strategic rate control can be used. The above discussion holds only when the damage rate is below the threshold (as in Fig. 7).

It has been shown that strategic repair leaves the decision as to whether to repair neighbor agents to each selfish agent. This game theoretic framework is suitable for an autonomous and distributed decision-making context that is suitable for the regulation and maintenance of large-scale information systems.

A major problem of using the Spatial Prisoner's Dilemma in regulating the repair rate of agents is that agents tend to remain silent and stuck at the Nash equilibrium of mutual defection. Here, we present a new solution to this problem: involving more systemic payoff incorporating not only its own remaining resources, but all its neighbor resources. With this modified payoff, agents not only have an adaptive decision-making dependent on the environmental parameters, such as failure rate and damage rate, but also have a more favorable resource allocation compared with a uniform regulation of repair rate.

5 Selfishware and Internet Being

In the microscopic model, we have shown the possibility of cooperation emerging when more systemic payoffs such as system reliability and further availability are taken into account, rather than simply counting the cost for repair. Indeed, when the payoff is modified to a more systemic one involving neighbor resources in the macro model, a cooperative strategy that will support neighbors by repairing can exist even when a defective strategy exists. With extended payoff, even adaptability to the environment (such as fault probability and maximum resources) has emerged.

Strategic repair cannot outperform a uniform repair strategy that is optimally tuned to the environment, however, the optimal one changes when the environment changes. Although strategic repair is not optimum in *any* environment, it has shown reasonable performance. This is due to the fact that strategic repair amounts to distributing the repair rate in a spatio-temporal sense: distinct agents can have distinct repair rates and agents can have distinct repair rates at distinct times. The merit of spatio-temporal flexibility of strategic repair will be more conspicuous in spatio-temporally dynamic environments – for example, failure rate could vary from agent to agent and from time to time.

Although we focused on a self-repairing task, the tendency would hold for other tasks that require cooperation of agents. The condition for existence of both selfishware and Internet being is obviously that all the players involved in the entity will benefit from it, or at least will not suffer from it. For the existence and maintenance of Internet being, however, a further condition is needed: payoff involves not only the short-sighted one, but also a payoff on a more systemic and longer term basis.

This would explain two phenomena observed in the Internet: one is that hypertexts (web documents) are posted and linked so explosively, and another is that computer viruses, worms, spyware, and spam email cannot be wiped out. The former is considered to be example of Internet being, corresponding to an organization of cooperators, while the latter is an example of selfishware but not Internet being, corresponding to an isolated small cluster of defectors.

The linked network of web documents benefits all the players involved – that is, not only the readers but also the providers (the ones who make postings), thus forming the Nash equilibrium. Viruses and spam mail benefit only the providers and rather harm users. However, the users have to pay a high cost to eradicate them and instead choose to neglect them, forming again a Nash equilibrium.

Focusing on spam mail, the usual e-mail network is an Internet being in the sense that both senders and receivers benefit from it. Spam mail can appear as ‘intruders’, since the usual e-mail network is not an evolutionarily stable strategy (ESS) [18].

Game theoretic study of complex systems such as the Internet will reveal that the network is a ‘culture media’ for artificial life, since it would allow Internet beings to emerge when certain conditions are met. As the network is used as a repository of knowledge and data, it can be a concentrator of computational intelligence when an organization such as grid computing is in operation. Then parasitic computing [3] will also emerge as selfishware.

6 Conclusion

For complex and large-scale artificial systems such as the Internet, the systems tend to be out of control; centralized and planned control would be difficult to apply. Autonomous and distributed management will be imperative and unavoidable rather than uniform control using a central authority. Autonomous and distributed management is favorable not only for control and management purposes but also for robustness against dynamic change, for such complex systems always undergo changes. When autonomous and distributed management is chosen as a framework, then we have to deal with selfish agents in exchange for leaving the control and management to each agent. The framework must guide the selfish actions of each agent toward the welfare of the entire system. Game theory has been studied and developed for such purpose, and has been applied to theoretical biology as well as economics.

The game theoretic approach to both micro and macro models for a network cleaning problem in a self-repairing network has been discussed. The game theoretic approach revealed conditions that selfish agents can cooperate and form an organization of cooperative selfish agents: their payoff must involve not only the selfish agent itself but also its neighbors' survival. By doing so, interacting selfish agents can avoid not only being deadlocked waiting for neighbors' support (Nash equilibrium), but also being attracted to all the dead states (attractor).

Based on the results, we also discussed when selfishware will emerge and under what condition it would further develop into an Internet being.

Acknowledgements

I am grateful to Prof. Fulcher not only for giving me the opportunity to present this work, but also his great assistance in editing and proofreading. I am also grateful to the anonymous reviewers, whose comments were quite helpful in improving the Chapter. I am indebted to Mr. Toshikazu Mori, Mr. Masakazu Oohashi and Mr. Yuta Aoki who helped conduct the simulations. This work was supported in part by a Grant-in-Aid for Scientific Research (B) 16300067, 2004. This work was also partly supported by the 21st Century COE Program 'Intelligent Human Sensing' of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1. Akella A, Seshan S, Karp R, Shenker S, Papadimitriou C (2002) Selfish behavior and stability of the internet: a game theoretic analysis of TCP. In: *Proc. ACM Annual Conf. of Special Interest Group on Data Communications (SIGCOMM'02)*, August, Pittsburgh, PA. ACM Press, New York, NY: 117–130.

2. Axelrod R (1984) *The Evolution of Cooperation*. Basic Books, New York, NY.
3. Barabasi A-L, Freeh VW, Jeong H, Brockman JB (2000) Parasitic computing. *Nature*, 412: 894–897.
4. Domany E, Kinzel W (1984) *Equivalence of Cellular Automata to Ising Models and Directed Percolation*. *Phys. Rev. Lett.* 53: 311
5. Dresher M (1961) *The Mathematics of Games of Strategy: Theory and Applications*. Prentice-Hall, Englewood Cliffs, NJ.
6. Feigenbaum J, Papadimitriou C, Shenker S (2001) Sharing the cost of multicast transmissions. *J. Computer and System Sciences*, 63: 21–41.
7. Feigenbaum J, Papadimitriou C, Sami R, Shenker S (2002) A bgp-based mechanism for lowest-cost routing. In: *Proc. 21st ACM Symp. Principles of Distributed Computing (PODC'02)*, July, Monterey, CA, ACM Press, New York, NY: 173–182.
8. Feigenbaum J, Shenker S (2002) Distributed algorithmic mechanism design: recent results and future directions. In: *Proc. 6th ACM Workshop Discrete Algorithms and Methods for Communication (Dial-M'02)*, 28 September, Atlanta, GA. ACM Press, New York, NY: 1–13.
9. Foster I, Kesselman C (eds) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA.
10. Hofbauer J, Sigmund K (2003) Evolutionary game dynamics. *Bulletin American Mathematical Society*, 40: 479–519.
11. Ishida Y (2005) A critical phenomenon in a self-repair network by mutual copying. In: Khosla R, Howlett RJ, Jain LC (eds.) *Proc. 9th Knowledge-Based Intelligent Engineering Systems (KES 2005)*, Lecture Notes in Computer Science LNCS/LNAI 3682. Springer-Verlag, Berlin: 86–92.
12. Ishida Y (2006) A game theoretic analysis on incentive for cooperation in a self-repairing network. In: Elleithy K (ed.) *Advances and Innovations in Systems, Computing Sciences and Software Engineering*. Proc. Intl. Joint Conf. Computer, Information and Systems Sciences and Engineering (CIS2E 06), 4–14 December, Bridgeport, CT, Springer-Verlag, Berlin.
13. Ishida Y, Mori T (2005) Spatial strategies on a generalized spatial prisoner's dilemma. *J. Artificial Life and Robotics*, 9(3): 139–143.
14. Ishida Y, Mori T (2005) A network self-repair by spatial strategies in spatial prisoner's dilemma. In: Khosla R, Howlett RJ, Jain LC (eds.) *Proc. 9th Knowledge-Based Intelligent Engineering Systems (KES 2005)*, Lecture Notes in Computer Science (LNCS/LNAI 3682), Springer-Verlag, Berlin: 79–85.
15. Koutsoupias E, Papadimitriou C (1999) Worst-case equilibria. In: Meinel C, Tison S (eds.) *Lecture Notes in Computer Science LNCS1563*: 404–413.
16. Lakshman TV, Kodialam M (2003) Detecting network intrusions via sampling: a game theoretic approach. In: *Proc. 22nd Annual Joint Conf. IEEE Computer and Communications Societies (INFOCOM'03)*, 30 March – 3 April, San Francisco, CA. IEEE Press, Piscataway, NJ: 1880–1889.
17. Mavronikolas M, Spirakis P (2001) The price of selfish routing. In: *Proc. 33rd Symp. Theory of Computing (STOC'01)*, 6–8 July, Hersonissos, Greece. ACM Press, New York, NY: 510–519.
18. Maynard-Smith J (1982) *Evolution and the Theory of Games*. Cambridge University Press, Cambridge, UK.
19. Nash J (1950) The bargaining problem. *Econometrica*, 18: 155–162.
20. Nisan N, Ronen A (2001) Algorithmic mechanism design. *Games and Economic Behavior*, 35: 166–196.

21. Nowak MA, May RM (1992) Evolutionary games and spatial chaos. *Nature*, 359: 826–829.
22. Oohashi M, Ishida Y (2007) A game theoretic approach to regulating mutual repairing in a self-repairing network. In: Sobh T, Elleithy K, Mahmood A, Karim M (eds.) *Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications*. Springer-Verlag, Berlin: 281–286.
23. Papadimitriou C (2001) Algorithms, games, and the internet. In: *Proc. 33rd Symp. Theory of Computing (STOC'01)*, 6–8 July, Hersonissos, Greece. ACM Press, New York, NY: 749–753.
24. Parkes D (1977) Iterative combinatorial auctions: achieving economic and computational efficiency. *PhD Thesis*, Department of Computer and Information Science, University of Pennsylvania, PA.
25. Roughgarden T, Tardos E (2002) How bad is selfish routing? *J. ACM*, 49(2): 236–259.
26. Shooman ML (1968) *Probabilistic Reliability: An Engineering Approach* McGraw-Hill, New York, NY.
27. Shoham Y, Wellman M (1997) Economic principles of multi-agent systems. *Artificial Intelligence*, 94: 1–6.
28. Taylor PD, Jonker LB (1978) Evolutionarily stable strategies and game dynamics. *Mathematical Bioscience*, 40: 145–156.
29. Walsh W, Wellman M (1998) A market protocol for decentralized task allocation. In: *Proc. 3rd Intl. Conf. Multi-Agent Systems (ICMAS-98)*, July, France. IEEE Computer Society Press, Los Alamitos, CA: 325–332.

Resources

1 Key Books

Axelrod R (1984) *The Evolution of Cooperation*. Basic Books, New York, NY.

Bertsekas D, Gallager R (1992) *Data Networks (2nd ed)*. Prentice-Hall, Englewood Cliffs, NJ.

Czumaj A (2004) Selfish routing on the Internet. In: Leung J (ed.) *Handbook of Scheduling*. CRC Press, Boca Raton, FL.

McKnight LW, Bailey JP (eds.) (1997) *Internet Economics*. MIT Press, Cambridge, MA.

Maynard-Smith J (1982) *Evolution and the Theory of Games*. Cambridge University Press, UK.

Tayler M (1987) *The Possibility of Cooperation*. Cambridge University Press, UK.

Weibull J (1995) *Evolutionary Game Theory*. MIT Press, Cambridge, MA.

2 Organisations, Societies, Special Interest Groups

Grid Computing Info Centre
<http://www.gridcomputing.com/>

IEEE distributed systems online
<http://dsonline.computer.org/portal/site/dsonline/index.jsp>

Market Design Inc.
<http://www.market-design.com>

3 Research Groups

Papadimitriou CH, Computer Science Division, University of California, Berkeley

<http://www.cs.berkeley.edu/~christos/>

Czumaj A, Department of Computer Science, New Jersey Institute of Technology

<http://web.njit.edu/~czumaj/>

Kearns M, Institute for Research in Cognitive Science at University of Pennsylvania

<http://www.cis.upenn.edu/~mkearns>

4 Discussion Groups, Forums

Topology Project

<http://topology.eecs.umich.edu/>

5 Key International Conferences/Workshops

FOCS 2007: 48th Annual IEEE Symposium on the Foundations of Computer Science

<http://www.focs2007.org/>

IEEE INFOCOM: Annual Joint Conference of the IEEE Computer and Communications Societies

<http://www.comsoc.org/confs/infocom/index.html>

PODC: ACM Symposium on Principles of Distributed Computing

<http://www.acm.org/podc/>

SODA : ACM/SIAM Symposium on Discrete Algorithms

<http://www.informatik.uni-trier.de/~ley/db/conf/soda/index.html>

STACS: Symposium on Theoretical Aspects of Computer Science

<http://www.informatik.uni-trier.de/~ley/db/conf/stacs/index.html>

STOC: ACM Symposium on the Theory of Computing

<http://sigact.acm.org/stoc/>

6 (Open Source) Software

The network simulator: ns-2

<http://isi.edu/nsnam/ns/>

7 Data Bases

Consensus road map for defeating distributed denial of service attacks

<http://www.sans.org/dosstep/roadmap.php>

Traceroute and Looking Glass

<http://www.traceroute.org>

Artificial Intelligence

Emotional Intelligence: Giving Computers Effective Emotional Skills to Aid Interaction

Chris Creed and Russell Beale

School of Computer Science, University of Birmingham, UK, cpc@cs.bham.ac.uk,
r.beale@cs.bham.ac.uk

1 Introduction

Why do computers need emotional intelligence? Science fiction often portrays emotional computers as dangerous and frightening, and as a serious threat to human life. One of the most famous examples is HAL, the supercomputer onboard the spaceship *Discovery*, in the movie *2001: A Space Odyssey*. HAL could express, recognize and respond to human emotion, and generally had strong emotional skills – the consequences of which were catastrophic. However, since the movie's release almost 40 years ago, the traditional view of emotions as contributing to irrational and unpredictable behavior has changed. Recent research has suggested that emotions play an essential role in important areas such as learning, memory, motivation, attention, creativity, and decision making. These findings have prompted a large number of research groups around the world to start examining the role of emotions and emotional intelligence in human-computer interaction (HCI).

For almost half a century, computer scientists have been attempting to build machines that can interact intelligently with us, and despite initial optimism, they are still struggling to do so. For much of this time, the role of emotion in developing intelligent computers was largely overlooked, and it is only recently that interest in this area has risen dramatically. This increased interest can largely be attributed to the work of [6] and [85] who were amongst the first to bring emotion to the attention of computer scientists. The former highlighted emotion as a fundamental component required in building believable agents, while the latter further raised the awareness of emotion and its potential importance in HCI. Since these publications, the literature on emotions and computing has grown considerably with progress being made on a number of different fronts.

The concept of designing computers to have emotional intelligence may seem strange, but equipping computers with this type of intelligence may provide a number of important advantages. For example, in spite of a computer's

impressive ability to process huge volumes of data in milliseconds and to perform complex calculations in a fraction of a second, they still have neither the ability to see or hear us, nor the ability to understand how we are feeling and to adapt themselves accordingly. Users often experience feelings of frustration at computers when they display obscure error messages, behave in erratic ways, and frequently crash. We may feel like venting our anger and frustration towards the computer, but it does not have the ability to respond in a constructive way. However, a computer that has some form of emotional intelligence would be able to detect that the user is feeling frustrated and angry, and would be in a position to take productive steps to alter this.

Such computers could use a number of strategies to help alleviate feelings of anger and frustration, such as opening a dialogue with users to ascertain the source of their emotions, apologizing for any mistakes made and working with the user to resolve them, and through expressions of empathy and understanding at the user's emotional state. Recent research has suggested that people who have damaged the emotional components of their brain find it difficult to make decisions as they cannot associate any emotion to a decision (in other words, would a particular course of action result in positive or negative feelings? [32]). Such a disability can have a destructive effect on people's lives as they consistently make poor decisions. This suggests that instead of computers becoming more unpredictable and irrational through having emotional intelligence, they would instead act and behave more rationally and predictably.

So how do we go about giving computers effective emotional skills that aid our interaction with them? This Chapter will provide an overview of what is required to achieve this, as well as the numerous issues involved in doing so. This Chapter is split into three core sections. Section 2 concentrates on the theory and research related to the building of emotional intelligence into computers. It starts by providing an overview of emotion theory, with a particular focus on the concepts and ideas most relevant for building and evaluating emotionally intelligent agents. A detailed perspective of the different approaches used in developing emotional intelligence in computers is then provided, along with a discussion of the limitations of these different approaches.

Section 3 provides an overview of our own work and how it relates to the standard approaches detailed in the previous Section. This includes an overview of what affective embodied agents are, how we respond to synthetic displays of emotion in such agents, research which has suggested that we have social and emotional relationships with computers, a discussion of the importance of conducting longitudinal studies when evaluating interface agents, and a description of the affective embodied agent that we have developed for experimental purposes. The final Section contains a discussion of the application of our approach to the real world by initially discussing areas and problems that affective embodied agents could potentially be useful for. We then provide a detailed example of where such an agent can be used within

a nutritional scenario, detailing how it can simulate a human health professional to help people change problematic behavior such as smoking, eating, and exercise.

2 Overview of Affective Computing

In order to understand how we can give computers effective emotions that aid our interactions with them, we need to start with an understanding of what emotions are and what exactly constitutes emotional intelligence. This section starts by providing an overview of what we currently know about emotions, including what causes them, how we express them, and what influences they have on the way we feel and behave. A detailed overview of the standard approaches used in attempting to incorporate emotional intelligence into computers is then provided.

2.1 What Are Emotions?

Emotion theorists have debated for centuries about what emotions are and what their primary function in human life is. This debate is far from over and there is currently no universally agreed upon definition of emotions. However, many scholars would at least agree that we experience different types of emotions in our everyday lives. An overview of these is provided below.

Basic Emotions

For much of the previous century, emotion scholars generally subscribed to a cultural theory of emotion, where emotions were believed to be culturally-specific learned behaviors that could only be experienced through observing other people expressing such emotions. However, [38] discovered that some emotions are not necessarily learned as previously believed, but are in fact innate and shared across all cultures. In his study, Ekman travelled to a preliterate culture (the Fore, in New Guinea) to ensure that the people there had not been exposed to Western media and had not learned the emotional expressions of Westerners. The subjects were told a number of stories, then asked to choose from a set of photographs of Americans expressing different emotions, the one which most closely matched the story. When tested, the Fore pointed to the same expressions that Westerners linked to the story. For further clarification, some Fore people were videotaped displaying facial expressions appropriate to each of the stories. After returning home, the experiment was completed in reverse by asking Americans to link the Fore faces to the different stories. The judgements of both the Fore people and the Americans again matched. These were named ‘basic emotions’, and while researchers often disagree about *how many* basic emotions there are, many would agree that anger, disgust, fear, joy, sadness and surprise can be classed as basic emotions.

Culturally Specific Expressions of Emotions

There are also cultural variations in the way in which humans express emotion. For example, [38] investigated the different emotion display rules that Americans and the Japanese have. In this experiment, both American and Japanese men were videotaped whilst watching some video clips. The clips varied as to whether they displayed neutral or pleasant events (such as a canoe trip) or less pleasant events (for example, nasal surgery). There were two showings of the video clips: one where subjects watched the clips on their own and another where subjects watched the clips with an interviewer present. When subjects watched the clips in private, similar expressions were noted in both American and Japanese subjects. However, when the interviewer was present, Japanese subjects smiled more and showed less disgust than the American subjects. When the videotapes were watched back in slow motion the researchers noticed that when the interviewer was present, Japanese subjects actually started to make the same expressions of disgust as the Americans did, but they were able to mask these expressions very quickly afterwards. Therefore, it appeared that the American and Japanese participants *did* actually experience the same basic emotions as these were automatic responses hard-wired into their brains. It was only a few hundred milliseconds later, that the Japanese subjects could apply their learnt *cultural display rules* and override the automatic response.

It has also been suggested that some emotions are culturally specific. For example, [65] reported on an emotion that is experienced by the Gururumba people of New Guinea that is not believed to be experienced from people of other cultures. This is known as the state of ‘being a wild pig’ and people who experience this state can become aggressive and often start looting, but rarely is anyone actually hurt or anything of importance stolen. This state is considered as normal among the Gururumba, as a way of relieving stress and maintaining mental health across the community.

Higher Cognitive Emotions

[56] has argued that in addition to basic and culturally-specific emotions, there are also ‘higher cognitive emotions’. These emotions are similar to basic emotions in that they are universal, but there are also variations on the way that they are expressed and experienced by different cultures, and there is also no single facial expression associated with them. Higher cognitive emotions also take longer than basic emotions to both develop and pass away. For example, consider romantic love. This emotion usually develops gradually in people over a period of weeks and months, while surprise (a basic emotion) is typically a very quick reaction to an event. Surprise also has a single universal facial expression associated with it, while there is no single universal facial expression for love. It is suggested that emotions such as love, jealousy, pride, embarrassment and guilt should be called ‘higher cognitive emotions’, because

these emotions typically require more processing in the cortex of the brain. This essentially means that these emotions can be influenced more by cognitive thought processes, while basic emotions are more reactive in nature.

Neurological Model of Emotion

The model of Fig.1 has been developed for some emotions (in particular, fear), based on work in neuroscience where it was found that fear is controlled by two different pathways in the brain [65]. Furthermore, the following three key regions of the brain were identified as being associated with fear: the thalamus, the limbic system (in particular, the amygdala) and the cortex [65]. Sensory input is initially received by the thalamus from the environment and transmitted simultaneously across the *low* road to the limbic system and up the *high* road to the cortex. The relevance of the inputs to an individual's concerns (in other words, their needs and goals) are then continually assessed by the limbic system, and if an input is evaluated as relevant, signals are sent both to the body for physiological reaction and to the cortex for processing. The first pathway (the thalamic-limbic) is the quicker of the two, and forces us to react to potential dangers. In being quicker, it is prone to make more errors and can often be initiated by false alarms, such as hearing a door slam. The second pathway (the thalamic-cortex) is slower, but more accurate, and can override feelings of fear evoked from the first pathway.

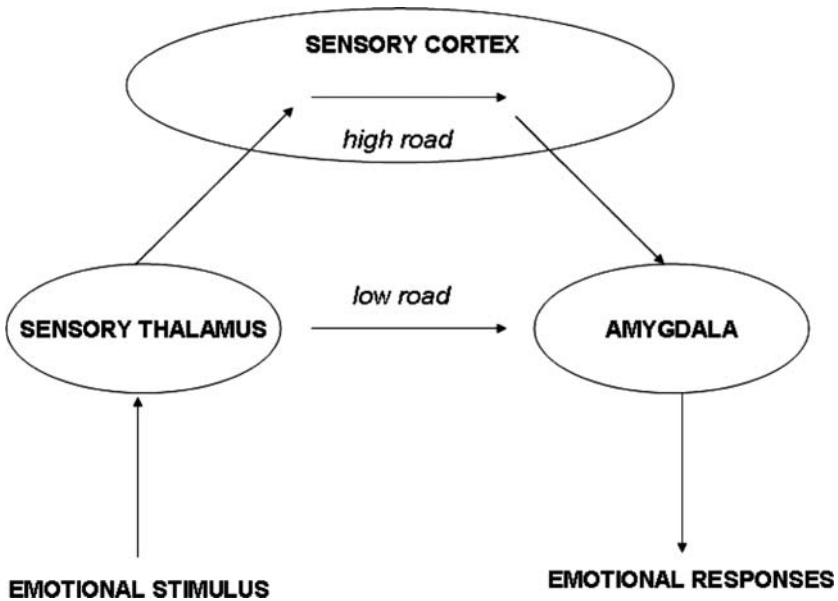


Fig. 1. LeDoux's neurological model of fear [65]

Primary, Secondary, and Tertiary Emotions

Emotions aroused from the first pathway are referred to as ‘primary emotions’, in other words, our hard-wired primitive emotions [32]. Primary emotions are typically reactions to stimuli such as outrage, being startled, or sexually stimulated. By contrast, ‘secondary emotions’ are defined as those emotions that require more cognitive thought, such as grief, frustration and pride. One patient – ‘Elliot’ (who had acquired damage to his frontal cortex as a result of a brain tumour) – was used to illustrate the difference between primary and secondary emotions. Elliot’s primary emotions still appeared to be functioning correctly as he could, for example, still be startled by a loud bang. However, if he saw a disturbing scene depicting a human head exploding, he knew cognitively that he should feel shocked, but physiologically there was no response where normally there would be. Elliot’s limbic-cortical pathway had been damaged, and as a result he knew that he *should* feel certain emotions, but did not.

It has been argued that there also exist ‘tertiary emotions’, these being emotional states that involve a partial loss of control over thought processes. When experiencing a tertiary emotion, it can be hard to concentrate on anything else, making it particularly difficult to attend to important and urgent tasks. Humiliation, infatuation, guilt and excited anticipation can be viewed as examples of tertiary emotions [97].

2.2 Emotions and Moods

A major problem in emotion research is the lack of a common language. Terms such as emotion, moods, drives, sentiments and attitudes are often used interchangeably by researchers and it can be unclear what is being referred to at times. These terms have meanings of their own and have been discussed at length in the literature [40]. Here we will focus primarily on the relationship between emotions and moods, since distinction between the two can be particularly difficult (people often use similar words, such as ‘happy’, to describe both). One obvious difference between the two is the duration for which each lasts. Despite disagreement about exactly how long emotions last, [39] suggests that they are very brief in comparison to moods and typically last a few seconds or minutes at most, whereas moods tend to last for hours or days.

[51] distinguishes between emotions and moods by arguing that emotions are ‘intentional’ and involve relationships between people and objects: “one is afraid of something, angry at someone, happy about something.” Moods, however, are ‘nonintentional’ and experienced more generally than emotions. Unlike emotions, they are not directed at any object in particular (although an object does have the potential to indirectly cause moods).

[33] suggests that emotions and moods can be distinguished through a functional analysis of each. Some emotion theorists have argued that the main

function of emotion is to bias the action we take in reaction to a particular situation. These emotions prepare the body to act quickly to these events and are usually very brief. However, the key function of moods is to bias cognition over extended periods of time. [33] further suggests that moods are always present and can affect our evaluation of events encountered both internally and externally. For example, someone in a positive mood is likely to view everything more positively, while somebody who is in a negative mood is likely to view everything more negatively.

Moods also appear to lower the threshold for experiencing other mood-related emotion. For example, an individual in an irritated mood can become more readily angry than they usually would. Situations or objects that would not normally cause such anger can do so more easily because of the mood of the person [39].

2.3 Expression of Emotion

Humans can express emotion in a variety of ways, the primary ones being written language, facial expressions, speech, and body language (such as posture and gait).

Written Language

Written language is a powerful medium for expressing emotion. People often express their emotions through stories, poetry and personal letters. People can literally state how they are feeling using emotive words such as ‘happy’, ‘sad’, or ‘ecstatic’. The colour, size, and shape of words can also be manipulated to add emotional emphasis to content (for instance, by animating text [106]). Symbols such as emoticons – for example, :-) or :-(– can also be used to convey emotion, and are particularly popular within domains where emotional information is lacking, such as email, instant messaging or text messaging.

Speech

Another powerful method for communicating and expressing emotion is through speech. In some scenarios, it is the only channel available for communication (for example, telephone conversations). Speech can also provide other information about a speaker such as their identity, age and gender. People can also use speech to simply communicate the emotions they are experiencing. Pitch (level, range and variability), tempo and loudness are considered the most influential parameters for expressing emotion through speech [4]. [75] have defined the general characteristics of a range of basic emotions (Table 1).

Table 1. Summary of emotional effects in speech (relative to neutral speech)

	Anger	Happiness	Sadness	Fear	Disgust
Speech rate	slightly faster	faster or slower	slightly slower	much faster	very much slower
Pitch average	very much higher	much higher	slightly lower	very much higher	very much lower
Pitch range	much wider	much wider	slightly narrower	much wider	slightly wider
Intensity	higher	higher	lower	normal	lower
Voice quality	breathy, chest tone	breathy, blaring	resonant	irregular voicing	wide, downward terminal inflections
Articulation	tense	normal	slurring	precise	normal

Facial Expressions

Facial expressions are one of the primary ways in which we can detect emotions in others. [41] have detailed the typical facial features that are associated with six basic emotions with the mouth, cheeks, eyes, eyebrows and forehead making up the core components of the face that are used to express emotion. [42] have also produced the Facial Action Coding System (FACS), which details the specific set of muscular movements for each of the basic emotions. FACS helped to develop the Facial Definition Parameter (FDP) and Facial Animation Parameter (FAP) sets which were designed according to the ISO MPEG-4 standard to enable the animation of faces, expressions, emotions, and lip movement. Humans are particularly adept at recognizing emotion in facial expression, and research has shown that people can accurately identify the emotional expressions of faces represented by as little as 37 lines, concentrating on the eye-brows, eyelids and mouth [44].

Gestures and Body Language

An overview and explanation of the meaning of different head, hand and body movements is provided in [52]. For example, a vertical (up and down) ‘head-nod’ often displays agreement or comprehension while listening. Clenched fists can signal an aroused emotional state, such as fear, anger, or excitement (for instance celebrating your team scoring at a sports event). Another example is arm-crossing, which is seen as a self-comforting and stimulating posture that is unconsciously used to ease anxiety and social stress [52].

2.4 Influence of Emotion on Human behavior

Attention

Emotion and attention are closely related. Paying attention to something can trigger an emotion while an emotion can influence what we focus our attention on. When an emotion is triggered it focuses our attention and mental focus onto an external object or event that produced the emotion [82]. For example, when we are angry we focus on the thing that angered us; when we are frightened we concentrate on what scared us; when sad we focus on what upset us.

Moods can also influence attention by focusing thoughts on the object or event that caused the mood. For example, when feeling upset, depressed or down, we tend to focus our thoughts on what made us feel this way. However, we can still experience moods without our attention being focused on anything in particular. For example, being in an anxious mood when walking down a dark alley at night helps us to keep alert for any potential signs of danger [45].

Numerous studies have focused on the effects of anxiety on attention. Anxiety narrows attention to whatever has caused the anxious feelings and little attention is given to almost everything else. Researchers have examined the effects of anxiety on attention through an experiment known as the ‘emotional Stroop test.’ [98] found that if people are asked to look at the printed text of colour names that were printed in a different colour from the text (for example, the printed text ‘red’ in a *blue* colour) and were then asked to name the colour of the text for each word, they take longer when the colour of the word and the colour of the print do not match. When the colour of the text and the printed word are mismatched it causes confusion and thus it takes longer to say the colour of the text.

The idea of the emotional Stroop test is that the words shown are both neutral and (potentially) emotionally arousing, to test whether it takes longer to name the colour of the words which are emotionally arousing. For example, [47] found that subjects who had been victims of rape were slower at naming the coloured words that were related to rape. This suggests that the anxiety caused by seeing a word associated with a traumatic experience focuses attention on that word, making it difficult to focus on other details such as the colour of the print.

Memory

Emotions and moods influence what we remember. We are able to recall events that are associated with either a strong positive or negative emotional reaction more easily than neutral events. For example, [24] reported on five experiments where groups of students watched a set of fifteen colour slides of what someone might see when walking to work. All groups of students saw the same slides,

except for the eighth one, of which there were three variants, namely: (i) a woman riding a bicycle, (ii) the same woman carrying the bicycle over her shoulder, and (iii) the same woman lying on the side of the road as if she had been hit by a passing car. After viewing the slides, students were then asked to recall what they had seen. Results found that people who had seen the woman lying on the side of the road could remember details like the colour of her coat more accurately than other groups. However, they struggled to remember other (peripheral) details, such as the colour of the car in the distance as well as the other groups.

The mood that we are in when attempting to remember something also influences our ability to recall it. For example, [12] illustrated that when we are in a happy mood, we seem to be able to recall pleasant events more easily than unpleasant ones. The opposite appears to apply when we are in a sad mood. In this experiment, subjects were asked to recall and describe incidents from when they were a child. The following day, when the subjects were in a neutral mood, they were asked to rate each incident as 'pleasant', 'unpleasant', or 'neutral'. The next day, happy or sad moods were induced in subjects and they were then asked to recall as many of their incidents as possible. Results found that people who had been induced into a good mood could remember more incidents that they classed as 'happy', but remembered less of the ones they classed as 'sad'. A similar (opposite) effect was also found for people who had been induced into a sad mood. This effect is often referred to as 'mood-congruent recall.'

Judgement and Decision Making

Emotions also have a strong influence on our judgement and the decisions we make. For example, one experiment [2] suggested that our judgement of other people is often influenced by the mood we are in when we meet them. In this study, same sex subjects were paired together to practice a job interview. Unknown to subjects was that their partners were actually helping the experimenters. The subjects were chosen to be the interviewers while their partners were the interviewees. Subjects were put into a good or bad mood by the experimenter by giving them problems to solve and then commenting on their performance, telling them that they had either performed much better than others, had performed averagely, or had done far worse than other people. The subjects were then asked to interview their partner through asking a set of pre-scripted questions, such as "What are your most important traits?" The interviewee replied with positive (for example, "I'm ambitious and reliable") and negative answers (such as "I'm impatient"). After the interview, the interviewers were requested to assess the interviewee on both work and personal aspects. It was found that subjects who were in a good mood had a tendency to rate the interviewees more positively and were more likely to employ them, while the subjects in a bad mood had a tendency to rate people

more negatively and were less likely to hire people [2]. This is despite the answers received by subjects being the same.

Creative Problem Solving

Moods have been found to have an influence on problem solving. For example, in one experiment [58], subjects were induced into either a good or bad mood and then asked to solve Dunker's candle task [36]. Given only a box of thumbtacks, the goal of this problem is to attach a lighted candle to the wall in such a way that no wax falls to the floor. It was found that subjects who were put in a good mood before the start of the task were more successful at solving this problem [58]. Another study which suggested the influence of emotions and moods on problem solving was that of [59]. In this study, medical students had either positive, negative, or neutral moods induced, and were then asked to diagnose patients based on their X-rays. Results from this study found that subjects who had a positive mood induced were able to make the correct diagnosis faster than subjects who had either negative or neutral moods induced.

Persuasion

Emotions also play a vital role in persuading people to do things. [71] investigated this by questioning students about whether or not they were in favour of gun control. Half of the students had a positive mood induced by watching a short comedy show, while the other half watched a more neutral programme about wine. Both groups were then provided with an argument about gun control that contradicted their own view on the subject – people who were in favour of greater control were presented with an argument against further restrictions, while people against greater control read an argument in favour of this. Half of the subjects were also presented with strong arguments, while the other half were provided with weak arguments. Furthermore, some subjects were informed that the person presenting the argument was a first-year student, while others were told that an expert was making the argument. Some subjects were also given only a short period of time to read the argument, while others were allowed to take as long as they desired. Once subjects had finished reading the argument, they were tested again to see if there were any changes in their view on the subject area. Results found that subjects were generally more influenced by the strong arguments than the weak ones. However, there was only a small difference for subjects who were put into a positive mood and had only a short period of time to read the argument, while the other groups found the weak arguments much less persuasive.

Emotional responses can also be used to manipulate the emotions and perceptions of others, for their own purposes. For example, sales people often try and build rapport through appearing empathic and understanding of their

potential customer's needs and concerns to make themselves appear more likeable. By making themselves appear more friendly, warm and likeable, they can increase the likelihood that people will comply with their requests [25]. Advertisers often attempt to play with our emotions to help sell their products. They use emotionally evocative images and music alongside the presentation of their product, in the hope that we will associate positive emotions with it. The same is also true when attempting to persuade people to stop participating in potentially dangerous and harmful things – for example, hard-hitting television advertisements which contain highly evocative graphical images of car crashes to warn about the dangers of drink driving. The hope here is that viewers will associate strong negative emotions with such behavior and thus avoid doing it.

2.5 Emotional Intelligence

The notion of emotional intelligence was first introduced by [95], and later popularized by [53]. Emotional intelligence is defined as: "...an ability to recognize the meanings of emotion and their relationships, and to reason and problem-solve on the basis of them. Emotional intelligence is involved in the capacity to perceive emotions, assimilate emotion-related feelings, understand the information of those emotions, and manage them." [72]

As can be seen from the above definition, the concept of emotional intelligence has been divided into four different areas to create a four-branch model, these being [73]:

- Accurately perceiving emotion
- Using emotions to facilitate thinking
- Understanding emotional meanings
- Managing emotions

Goleman's Emotional Competence Framework, on the other hand, divides emotional intelligence into five core competencies [54]:

- *Self Awareness*: knowing one's internal states, preferences, resources and intuitions
- *Self-Regulation*: managing one's internal states, impulses, and resources
- *Motivation*: emotional tendencies that guide or facilitate reaching goals
- *Empathy*: awareness of others' feelings, needs, and concerns
- *Social skills*: adeptness at inducing desirable responses in others

Goleman suggests that the above emotional intelligence capacities make up a hierarchy in which they build on each other [54]. For example, self-awareness is essential for self-regulation and empathy, while self-regulation and self-awareness are crucial for motivation. All four preceding competencies are required for social skills.

These different categorizations are not contradictory; instead, they reflect slightly different perspectives from the researchers as to the focus and extent of emotion, and serve to illustrate that emotions are complex things with multiple effects on ourselves, on our perceptions of the world and others in it, on our desires and actions, and on our behavior and responses.

2.6 Approaches Used in Developing Emotionally Intelligent Computers

So how can we build emotionally intelligent computers? The previous Section discussed some of the core competencies required for humans to be considered emotionally intelligent. Therefore, if we wish to simulate human emotional intelligence, computers will also need to be adept in these areas. But this is problematic. For example, one of the major disadvantages that computers have over humans is that they do not have the sensory organs of humans – such as eyes and ears – for recognizing emotional responses in others. Recent progress has been made in building emotional voice recognition software and applications that can track subtle facial movements and measure physiological signals associated with emotion, but much of this work is still in its infancy. This Section provides an overview of the different approaches being taken with regard to building emotionally intelligent computers.

Computational Emotion Models

The goal in developing a model of emotion is to enable machines to evaluate events and objects in such a way that they exhibit believable emotional reactions to what has been evaluated (in other words, an emotional response similar to that of a human). For these emotional reactions to be convincing, emotion models should enable computers to express emotions believably at both the right intensity and at the appropriate time. The following references provide an overview of work that has been completed in this research area [20, 21, 23, 35, 84, 87, 103, 104].

Appraisal theories have had a strong influence on the development of computational models of emotion. The term ‘appraisal’ refers to the evaluation of antecedent events that result in a particular emotion being experienced. The model most often used to incorporate emotion into agents (that is based on appraisal theory) is the OCC model [83], which is a computational model of emotion that contains 22 emotion categories based on valenced reactions to the consequences of goal relevant events, actions of another agent (as well as itself), or according to the attractiveness of objects. The model also provides variables for determining the intensity of an emotion.

The model, however, does have its limitations. [5] suggests that this model provides a useful starting point for incorporating emotion into agents, but is too complex for creating believable characters. For example, if facial

expressions are used as the medium to express emotion, it then becomes very difficult to map the 22 emotional categories of the OCC model to the 6 facial expressions identified by [38] as being used to convey emotion. Therefore, when the model is used in this context, it needs to be simplified to match the abilities of the interface agents. It has been further suggested that the OCC model requires extended features, including a history function, a personality designer, and the interaction of emotional categories (in other words, how the emotional value of an event should affect the current emotional state of the agent) [5].

An alternative model is that of [55], who have developed a domain-independent model based on a framework of appraisal and coping which is used in the design of autonomous virtual humans. The usefulness of this model has been demonstrated with a Mission Rehearsal Exercise (MRE) system that trains people for peacekeeping activities.

Computational models of emotion are often tested visually through the use of embodied agents and virtual worlds. Advances in 3D graphics have enabled developers to create realistic embodied agents that can be used for testing purposes to examine whether emotion models are providing the desired emotional responses (see, for example, [85]).

Detecting Emotions to Aid Interaction

Knowing whether a user is experiencing frustration, satisfaction or some other emotional state provides the computer with the opportunity to intelligently adapt itself in an attempt to enhance its interaction with people. However, simply detecting user emotions is far from easy and even if achieved, there is then perhaps the larger issue of how should computers appropriately adapt themselves to these emotional states? For computers to be able to express useful emotions, they need to be able to understand how a user is feeling. This section provides an overview of the different ways in which a computer can detect human emotion.

Autonomic Responses

Emotion can be measured (to an extent) through measuring automatic physiological activity such as heart rate, blood pressure, blood pulse volume, respiration, temperature, pupil dilation, skin conductivity and muscle tension. Emotion can also be measured through neurological changes, with the most common measure for this being the electroencephalogram (EEG). However, while it is now relatively easy to measure many of the above, it is still very difficult to distinguish between different emotions. For example, a number of different emotional reactions such as fear, anger, surprise and happiness involve an increase in heart rate [17]. Therefore, when an increase in heart rate is observed by a computer, it has no way of knowing which of these emotions is being experienced. Despite problems such as these, some success has

been achieved through the use of multiple measures of autonomic signals. For example, [88] achieved 81% recognition accuracy on eight emotions by combining a variety of measures such as facial muscle tension, respiration, skin conductance and blood pressure volume.

Another study with promising results was that by [94] who reported on an experimental system that aimed to automatically recognize user frustration from physiological signs of skin conductivity and blood volume pressure. In this study, subjects played a competitive computer game where they had to efficiently and accurately solve as many visual puzzles as possible, in order to win a cash prize. To induce frustration in subjects, the computer game experienced a deliberate delay, at irregular intervals, in which the mouse appeared not to work. Whilst playing the game, the subject's skin conductivity and blood volume pressure were measured to observe if frustration that was likely to be caused by the game (namely, when there was a delay) could be told apart from times when the game was less likely to cause frustration (in other words, when the game was running without any delays). It was found that this approach worked significantly better than having a random guess at when the game might cause frustration. Additionally, results found a correlation between the mouse clicking behavior of subjects and frustration-eliciting events.

Lie detectors – so beloved of older police and spy dramas – provide us with an example of a physiological stress measurement, with the assumption being that under stress, galvanic skin response is altered and can easily be detected. Their accuracy is critically dependent on the skill of the operator, and even then is able to be fooled by practiced participants.

Facial Expression

Another potential way for computers to detect emotion in users is through monitoring facial expressions. FACS [41] is often the foundation used by designers when attempting to give machines the ability to recognize facial expressions [102]. One approach that has attracted a lot of interest and has provided some promising results is that of pattern recognition of different images, with recognition accuracy approaching 100% with some basic emotions [27, 43, 68, 70]. Another method for recognizing facial expressions that has had some success is facial electromyography (EMG). EMG signals have shown promise in being able to distinguish between different basic emotions [18].

Speech

Speech provides another opportunity for computers to detect a user's emotional state. As mentioned previously, our voices can express emotion through changes in speech such as pitch range, speech rate and rhythm [77]. Few systems have been built which attempt to autonomously detect emotion from

speech, but some have shown promise, such as [1] and the ASSESS (Automatic Statistical Summary of Elementary Speech Structures) system [26]. Autonomously extracting emotional content from speech can be a difficult process. For example, [71] conducted a study that examined autonomous detection of a small set of emotions expressed (in a highly emotive fashion) in an echo-free and quiet environment. The authors mention a number of issues in attempting to do this, such as having to create large databases of emotional content, using a method that produces appropriate emotional content for analysis (such as getting people to read emotive text, as opposed to using spontaneous emotional speech) and assessing the quality of emotional speech. A detailed review of emotion recognition systems is provided in [27].

Questioning Users

Another approach for determining the emotional state of a user is to simply ask them. An often-used approach is to ask subjects to choose an emotional adjective that best describes the way they are feeling. Profile of Mood States (POMS) [75] is an adjective-based measure of mood that is often used. Another example is the Differential Emotion Scale (DES) [60] which is a questionnaire that contains twenty-four emotional adjectives that people rate on seven-point scales as a means of detailing their affective feelings. Other questionnaires are based on dimensional theories of emotion where the assumption is that emotion can be described through two different dimensions: arousal and valance (see, for instance, [64]).

Using questionnaires such as these to determine the emotional states of users raises a number of issues. For example, people often find it difficult to articulate how they are feeling, and using a single adjective to do this can make it more difficult for them. Also, if questionnaires are used after the completion of an experiment to determine the emotions experienced, then, as previously discussed, people's memories are likely to have been influenced by the emotions they experienced. Asking subjects how they are feeling during the experiment is likely to interrupt that emotion and thus influence their response.

Simulating Human Expressions of Emotion

This Section provides an overview of the three main ways in which computers can simulate emotion: (1) through written language and manipulation of static text, (2) through synthetic or recorded speech, and (3) through the use of embodied agents which can simulate human facial expressions and body language.

Emotive Text

A number of studies have shown that emotive textual content displayed by a computer can have a significant impact on our perceptions, behavior and

performance. For example, subjects in [48] played a guessing game with a text-based interface and received one of three differing types of feedback during the interaction: *sincere praise*, *flattery* (insincere praise) or *generic feedback*. In the *sincere praise* and *flattery* conditions, the computer would display responses like “your question makes an interesting and useful distinction”, or “you seem to have an uncommon ability to structure data logically”. In the *generic feedback* condition, subjects simply saw a message that said “begin next round”. The flattering comments made by the interface agent were found to have a similar effect as flattery from another person, even though subjects were fully aware that their participation was with a computer. In this case, subjects found the interaction with an agent that flattered them to be more enjoyable than with one which did not. That is, the textual content displayed by the computer had a significant influence on user’s perceptions and emotions.

Another example of how emotive text can influence people is a study which found that computers have the potential to alleviate feelings of frustration [62]. In this study, subjects participated in a game which froze at random intervals (to frustrate subjects) when competing for a cash prize. To help ease the subject’s frustration, an interactive ‘affect support’ agent was designed in the form of a text-based questionnaire. Subjects were split into three groups, with each group interacting with a different type of agent: a support agent, an agent that allowed subjects to ‘vent’ their anger, and an agent which ignored their feelings completely. During the first phase of interaction, subjects initially played the game and then interacted with one of the agents. After this interaction, subjects were then asked to play another version of the game (which did not freeze) for at least another three minutes. After this time had elapsed subjects were free to continue playing or to leave.

In the *ignore* condition, subjects were asked close-ended questions that did not involve emotions or provide an opportunity to report a problem like web delays. In the *vent* condition, subjects were asked open-ended questions that gave them the opportunity to report the relevant problem, as well as their emotional state. In the *affect-support* condition, subjects were asked mostly the same questions as in the vent condition; however, after the computer asked how frustrated the user was feeling, the computer gave feedback based on the user’s reported frustration level. Feedback included comments like, “wow, it sounds like you felt really frustrated playing this game”, and “that must feel lousy. It is no fun trying to play a simple game, only to have the whole experience derailed by something out of your control.” It was found that subjects who had initially interacted with the support agent, spent significantly more time in the second phase interacting with the computer that had tried to frustrate them, than subjects who interacted with agents that had either ignored their feelings completely or allowed them to ‘vent’ their frustrations. This experiment used simple plain text to manipulate subjects’ behavior.

Emotive Speech

There has been a lot of interest in speech interfaces over the last decade with the intelligibility of synthetic speech nearing that of human speech [78]. Incorporating emotion into speech has proved to be quite a challenge, although [19] illustrated that synthetic speech can contain recognizable affect by copying the effects of emotion on human speech. Computers do not always need to use synthetic speech to communicate with users – they can also be programmed to use recorded human speech, which can be used to convey emotion more clearly (for example, through a happy or sad voice). The choice of words used by a computer can also be used to give an indication of its *feelings*. For example, when asking a computer to do something, if it replies with “if I must”, as opposed to “of course, no problem”, this can suggest how the computer feels.

Facial Expressions

As mentioned previously, research has consistently provided evidence that humans are capable of identifying and distinguishing between different basic emotions (independent of their culture) including anger, fear, joy, sadness and surprise [38]. The FACS system [42] (and other similar systems) which detail the specific set of muscular movements required for the facial expression of each basic emotion, have been used by researchers as a basis for giving embodied agents emotional expressions. Embodied agents that have used systems like this include **Baldi** [11] and Perlin’s responsive face [86]. Research has also provided evidence that despite current technology not being sufficiently advanced to dynamically generate facial expressions exactly the same as human ones, humans can still consistently identify the facial expressions being displayed. For example, [3] used **Baldi** (Fig. 2) to test whether or not an embodied agent’s emotional expressions were as convincing as human expressions. It was found that the emotional expressions of **Baldi** were as convincing as human expression and that knowledge about the source of the emotional expression had no major impact upon the convincingness. Although **Baldi** is an animated agent, some studies – such as [13] – have successfully used still images to get embodied agents to express emotion.

Body Language and Gesture

While the relationship between emotion and gestures is not as well understood as that of facial expressions and emotions, a number of affective embodied agents can still use body language and gestures to convey information. An exercise advisor named **Laura** [10] uses a range of non-verbal behaviors including hand gestures, body posture shifts, gazing at (away from) the user, raising and lowering of eye brows, head nods, and walking on and off the screen. **Laura** also has the ability to convey immediacy behavior and when expressing empathy can appear nearer to the screen to show an empathic facial expression.

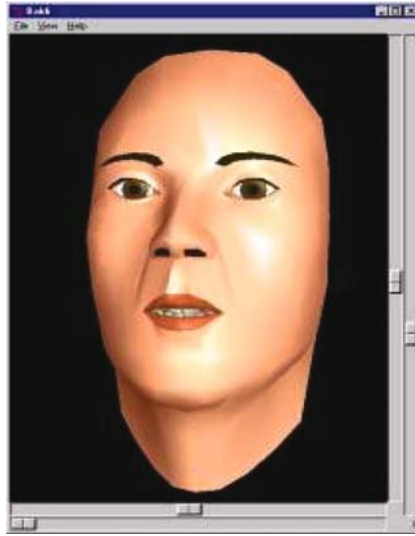


Fig. 2. Screen shot of Baldi

Other embodied agents that use body language and gestures to communicate with users include *Herman the Bug* [66] and the Real Estate Agent (REA) [9].

2.7 Ethics

In order for computers to build social and emotional relationships with users, they require certain capabilities to allow them to detect and manipulate user emotion, as well as being able to express emotional states of their own through different channels. This raises numerous ethical and technical issues, many of which have been discussed in the literature [89]. In this Section, we highlight some of the main issues involved.

Genuine Emotional Expressions

One ethical issue that arises from incorporating affective capabilities into computers is whether or not emotional support offered by computers is *genuine*? That is, does it matter that when computers express or communicate emotion to users that they do not actually *feel* the emotions as humans would? Looking to human-human interaction, it would suggest not, as we often interact with people who are trained to use certain relational strategies to build trust with us, despite them not genuinely feeling sympathetic or empathic toward us. For example, social workers, nurses, general practitioners (doctors), and psychotherapists are all trained to use certain relational strategies, such as empathy, to enhance relations. Also, consider a help desk assistant who has to deal with numerous queries every day and who is trained (and expected)

to empathise with each customer's individual problem. Are these expressions of emotion genuine or are these people just performing their everyday tasks? At times they may build a genuine rapport with a customer and feel bad that the product purchased by the customer is faulty, but on most occasions it is likely that they are empathising since this is what is expected of them, both by the customer and their employer.

[87] uses the scenario of a dog greeting its master to suggest that expressed emotions do not necessarily have to be authentic to meet some of our basic emotional needs. When the master arrives home the dog is often *happy* and will start wagging its tail. However, if the master appears to be *sad*, dogs somehow have the ability to recognize this and they will often put their heads down and flick back their ears in response to their master's emotional state. The master, in seeing the empathic response, will often change their posture and begin to feel a little better. Once the dog recognizes this, it too will raise its head and start wagging its tail. It is not known how dogs can perceive the emotional states of others, or whether they have their master's best interests at heart, but this simple interaction often has the effect of meeting some of the simple emotional needs that we as humans have.

Should HCI Replace Human-Human Interaction?

Another important question is that of whether HCI should ever replace human-human interaction. For example, in the future, should teaching agents replace human teachers? At present, it is hard to argue in favour of computers replacing important roles requiring social interaction, as they do not have the social and emotional intelligence required. They struggle in building rapport with users and cannot inspire or motivate people outside of a narrow social dialogue. Technological advances over the coming years may change this, resulting in more socially astute agents, but would this ever warrant replacing their human equivalent? They would likely be cost-effective, require little maintenance, and would not complain about how much they get paid. However, while it is often easy to envisage fully embodied and socially competent agents conversing with people in natural language, it is not so easy to predict how people would respond to these entities. People may feel uncomfortable interacting with such agents and reject the technology outright. Alternatively, they may find it novel, entertaining, and a natural way to interact, and thus embrace such agents. Using computers to help supplement the roles that humans perform would perhaps be more practical and useful. Agents that can help with exam revision, explain more about particular illnesses after you have visited the doctor, or help you practice important interviews or presentations, could potentially be of use.

Manipulation

One issue which arises from building computers with affective capabilities, is the opportunity for manipulation [29]. Computers that can accurately and reliably detect emotional states have access to some very personal and private information, which could potentially be used to manipulate how we feel. Recent work has illustrated that agents which are programmed to be empathic toward the user are perceived as more caring and trustworthy [1, 10]. In human-human interaction, someone who we perceive to care about us can have more of an influence over our behavior and we generally trust information more when it is from such a source [25]. Therefore, *caring* computers may have increased persuasive powers that can be used to influence us. Is it acceptable for agents to manipulate (and possibly deceive) people in this way to help companies sell more products? Perhaps, as long as the user feels they have received good value for their money and do not feel manipulated. Human sales people often present the products they sell in their best light, even when they are fully aware that the product has certain features that are not desirable for the customer. Most people are aware of this and while they may not be overly keen about it, they generally do not mind if they feel that they have received good service and value for money. The same is likely to apply with computers; if users feel that they have received a good deal and service then they will be happy, otherwise, if they feel manipulated and cheated, they will be unhappy and unlikely to return with their money.

Negative Emotions

Assuming that computers will one day have the ability to detect user emotion, should they try to eliminate all (so-called) ‘negative’ emotions and attempt to make a user feel better on a consistent basis? A problem with this question is that it is hard to define what is meant by ‘negative’ emotions, and in any case, if there was an appropriate definition, negative emotions are not necessarily all bad. [16] are investigating how an embodied teaching agent can help users work through frustration, which is often regarded as a negative state. However, the ability to work through frustration is essential in learning environments, as the adage ‘no pain, no gain’ suggests. This is particularly clear when observing people playing computer games, in which a difficult passage of the game is attempted again and again until the user manages to crack the techniques needed: the frustration felt there serves to motivate them to continue until they succeed. The skill in designing good games comes in pitching the level of difficulty such that the frustration levels are not too great to cause the player to give up, but hard enough so that their sense of achievement and relief is sufficiently high. Thus, computers should not necessarily try to restrict users from experiencing certain emotional states, but instead should attempt to help them understand and make use of their emotions, to help them achieve their goals. For example, a computer could help alleviate anger through teaching

users anger management strategies. In essence, computers would be helping user's to build their *emotional intelligence*.

Privacy

Privacy of emotional data is another issue raised from computers detecting emotion. If a computer detects that a user is suicidal, should it inform somebody, such as the person's doctor, the police or a friend? If the user is just feeling a little depressed after a hard day, should the computer contact a friend or family member in an attempt to cheer the person up? Or should it not interfere? These are hypothetical questions as computers are still not capable of accurately and reliably detecting human emotion, nonetheless, if computers do one day have this ability, then how *responsible* the computer is for managing the user's emotional state becomes an important issue.

Human Relationships

Another important concern surrounding computers and their attempts to build social relationships with people is whether or not they will have an impact upon people having healthy social relationships with others. Many argue that we should be spending more time away from our computers and should be interacting more with other people. This is a valid point, but it could be argued that it is unlikely that just because a computer becomes more considerate and makes more of an effort to consider your feelings (through raised emotional intelligence), that people will want to spend more time interacting with it. For example, people often interact with pets, but while they meet some of the basic emotional needs that we have, most people still crave the company of others. However, the research of [63] – highlighted previously – suggests that people may be more willing to spend time interacting with computers that have some form of emotional intelligence. Thus, it remains difficult to predict how people will respond to emotionally intelligent machines.

3 Evaluating Affective Embodied Agents

This Section provides an overview of embodied agent research in the field of affective computing. We start by providing an overview of the use of embodied agents and research that has looked at incorporating emotion into them. We then move on to discuss research which has looked at how people respond to simulated displays of emotion and the effects it has been reported to have on them. Following this, we discuss the importance of evaluating interface agents of all types over extended periods of interaction, and then proceed to discuss an embodied agent that we have developed which is capable of simulating emotional facial expressions.

3.1 What are Affective Embodied Agents?

Terms such as ‘embodied agents’, ‘virtual humans’, ‘interactive characters’, ‘software agents’, ‘interface agents’, and ‘intelligent agents’ are among many that are often used interchangeably when talking about similar entities. On many occasions, this can lead to confusion and difficulty in understanding exactly what is being referred to. Therefore, it is important to clarify what is meant by the term ‘embodied agent’ for the purposes of this Chapter. Embodied agents are essentially animated or static entities that are based on a computer screen and attempt to interact with users in some way. They can use a number of techniques to interact with users including written language, speech, gesture, facial expressions, head nods, and eye gazes. These agents can also have a variety of different representations (for instance, human, alien, paperclips, dogs, cats, and so on). Affective embodied agents are agents that exhibit, express and/or act on emotion. They are often based on an emotional model that determines their emotional reactions, but this is not always the case. A wide range of affective embodied agents have been developed over the last decade including the pedagogical agents *Cosmo* [65] and *Herman the Bug* [67], *Steve* [94], *PPP Persona* [105], *Gandalf* [101], *MACK* [22], *Olga* [7], *Laura* [8] and the *REA* [9].

3.2 Psychological Responses to Simulated Emotion

In order to understand if the emotions that designers have incorporated into an embodied agent actually aid an interaction, it is important to understand how people respond to synthetic displays of emotion. How do we respond to *synthetic* displays of joy, happiness, sadness, frustration, fear and anger? Can we catch emotions from computers? Do we like agents that are always happy, or does this annoy us after a while?

Social-Emotional Relationships with Computers

Numerous studies have suggested that we interact with computers as though they are social entities. [92] developed the *Computers Are Social Actors (CASA)* paradigm, which implies that the social rules that apply in human-human interaction also apply to HCI. The reason for this, they suggest, is that our ‘old brains’ have not evolved to deal with current technology and therefore we treat all media as if it were a social entity. It may seem a little strange to suggest that we respond to computers like people, but it has been shown that the response is particularly strong and often unconscious. Even when we are consciously aware that the entity is not human (for instance, a computer or television) the response is still not weakened.

For example, [81] suggested that humans are polite to computers. Research completed in social psychology has found that interviewers who ask about their own performance are likely to receive more positive feedback than if

feedback is received from another source. This study tested if the same politeness rules also applied to human-computer interaction. The study involved subjects completing a task on a text-based computer and upon completion they were interviewed about the performance of the computer by either the same computer, a pencil-and-paper questionnaire, or a different but identical computer. Similar to that of human-human interaction, results found that subjects evaluated the computer more positively when the computer asked for feedback about its own performance, compared with subjects who evaluated the computer through a pencil-and-paper questionnaire or another computer.

Another study found that we seem to attribute personalities to computers and also respond to those computer personalities as if they are human [80]. In this study, properties associated with dominance and submissiveness were incorporated into computers using plain text. It was found that subjects not only recognized the computer's personality, but also reported being more satisfied with the interaction they had with the computer that shared a similar personality to their own. Again, this finding is similar to human-human interaction, where research has found that people tend to prefer interacting with other people who have a similar personality to their own. Similarly, [79] found the same attraction using computer-generated speech by incorporating the properties associated with introversion and extraversion. When the personality of the computer voice matched the personality of the subject, subjects reported the voices as being more attractive, informative, and trustworthy. Moreover, they were more likely to buy books reviewed by the computer.

Simulated Emotion

Our tendency to respond to computational entities as social actors suggests that we may well respond to synthetic displays of emotion in a similar way to human emotion, and a number of researchers have been investigating this. One of the main approaches used to investigate this is to compare different types of emotionally expressive agents with each other. For example, [13] examined how we respond to both self-oriented and other-oriented empathic emotion. Subjects played a blackjack game and were matched up with an agent that either exhibited or lacked self-oriented or other-oriented empathic emotion. The agent was a static photograph of a human face, which could communicate with subjects via a speech bubble next to the photograph. When a round of the game had finished, the agent would always communicate to the subjects an evaluation of its own performance (for instance, "I'm glad *I* won"), and then followed that with an evaluation of the subject's performance (for example, "I'm glad *you* won"). Also, in the conditions where empathic emotion was used, the agent's evaluation of the subject's performance included an emotional response: the agent would express negative emotion if the user lost and positive emotion if the user won. The results from this study found that people generally preferred the agents that were empathic to them more than ones which were not.

Another approach is to compare an emotional entity with a different type of entity. For example, [100] used a card matching game to compare the impact of an emotional face and a 3D arrow on a subject's eye movements and response times. The arrow and the face were both used to provide feedback to the user during the game. For example, through pointing or gazing at the player whose turn it was next. Results from this study found that the emotional face elicited more eye contact from subjects than the 3D arrow. The authors therefore concluded that the emotional face was more engaging to subjects than the arrow.

There have been a lack of studies which have explicitly tested an emotional agent against an unemotional one. In many related studies, emotional agents are used, but the incorporation of emotion into the agent is not the main focus of the study. Therefore, the potential impact of simulated emotion has to be inferred from the reported results. This can make it particularly difficult to attribute any of the effects found to the inclusion (or exclusion) of emotion in agents. For example, [66] examined the effect of different types of animated agents on the learning performance and experience of middle school children by asking them to design a plant that would be able to survive in a particular environment. The children received varying levels of help from the animated agents and the results of the experiment found that a fully expressive agent (that is, an agent which offered advice using both animation and speech) was perceived to be equally as entertaining as a muted agent (in other words, an agent which offered no advice whatsoever). However, the incorporation of emotion was not explicitly tested in this experiment, so caution must be applied when analyzing the results.

3.3 Evaluating Agents over Extended Interactions

Very few studies have focused on how we respond to emotionally expressive embodied agents over extended periods of interaction. As discussed above, some recent studies have suggested that we seem to perceive emotional agents as more likeable and trustworthy than unemotional agents, but does this effect remain consistent over five, six, seven or forty separate interactions? The Microsoft Office *Paperclip* was an emotionally expressive agent that many people tended to find novel to interact with initially, but after further interactions, it began to frustrate people and was ultimately rejected by users. As we move more towards managing computer systems rather than directly manipulating them, we will work more closely with agents in everyday activities as they undertake tasks on our behalf. We are likely to start interacting with them on multiple occasions spanning days, weeks and months. Therefore, we need to understand in more detail how our perceptions towards affective embodied agents change over numerous interactions and extended periods of time.

[10] are some of the few researchers who have started to investigate this space. They developed an embodied exercise advisor named *Laura*, which attempted to maintain a relationship with subjects who interacted with the agent daily over the period of a month. People who were not completing required levels of exercise recommended for United States' adults (namely 30 minutes or 10,000 steps per day) were chosen as subjects, in an attempt to help them improve their exercise habits through interacting with *Laura*. A variety of different strategies were used by the agent to help maintain a relationship with subjects, including social, empathic and polite communication, talking about the relationship, humour and appropriate forms of address. *Laura* also used a range of non-verbal behaviors, as discussed earlier. Results from this study found that subjects liked *Laura* more when they interacted with the relational version as opposed to a non-relational one (that is, where no relational strategies were used).

A similar effect was found by [69], who asked subjects to use a mobile health device for eight days and then examined the effect it had on subjects' perceptions and behavior. Subjects were split into two groups: one group interacted with an empathic version of the device for four days and then switched to the non-empathic device for the final four days, while the other group did the opposite. The system made use of a text-based agent which would interrupt subjects at different times of the day to discuss issues relating to their health. The empathetic agent would make empathic comments when interacting with the subject while the non-empathic agent would not. Results found that a significant number of subjects who were asked which device they would like to continue interacting with at the completion of the study, stated that they would prefer to continue interacting with the empathic device. Subjects also reported that they felt less stress when interacting with the empathic device.

3.4 Our Affective Embodied Agent

In order to investigate our responses to synthetic displays of emotion in embodied agents, we have developed our own for testing purposes (Fig. 3). Agents such as these used to be costly to develop in both terms of time and expense; however, it is now possible to easily develop such agents using affordable software which automates much of the process [90]. Our agent simulates the role of a human health professional through making use of many of the skills and strategies that human health professionals use (discussed in the next Section). The agent can move its head, speak (via a recorded voice), and can display a wide range of (facial) emotional expressions. There are a number of applications that can be used to develop agents such as these, but one of the most popular for specifically building virtual characters is *Poser* [37]. We used *Poser 5* for developing our agent, along with *Mimic 3* [34], which is compatible with *Poser* and can be used for automatically generating facial animation and lip synchronization from imported audio files containing recorded



Fig. 3. Our embodied agent

speech. As we are conducting our experiments over the World Wide Web, we converted the animations produced by *Poser* and *Mimic* to the *Macromedia Flash* format, so that we can incorporate the animations into a web page.

4 Application of Affective Embodied Agents

Affective embodied agents have often been touted as one of the primary ways in which we will interact with computers in the future. Advocates of embodied agents believe that they will make an interaction more natural and engaging, while opponents believe that they will raise expectations of what the computer is capable of, and thus hinder interaction [30]. The future of agents such as these is still unclear, as many different fields of research need to mature sufficiently before we can really assess their potential. The agents that have been developed to date are unable to interact naturally with people, and as a result they quickly lose credibility. Areas where embodied agents seem to have found their niche is within computer games and simulations. These are likely to be areas where affective embodied agents will be of real use, unlike work-based tasks where an agent of this sort is not really required (as exemplified by the Microsoft Office *Paperclip*). Another area where affective embodied agents could be of use is where human relationships are known to be important. For example, in the behavior change domain, the relationship between a helper and client has been shown on numerous occasions to be fundamental in helping people to change problematic behavior. In this Section, we detail our research into how an affective embodied agent could be used to simulate the role of a human health professional to help people change problematic behavior such as eating, smoking, and exercising.

4.1 Affective Embodied Agents for behavior Change

behavior change is one domain where affective embodied agents may prove useful. Changing problematic behavior in humans can often be a long and difficult process. Exercise regimes, healthy dieting, smoking cessation, and a number of other behavior change plans are regularly initiated with much enthusiasm, but all too often are abandoned before the new behavior replaces the old. People who have difficulties in changing unhealthy behavior often seek professional advice to help them achieve their behavioral change goals. The effective management and use of emotion in a therapist-client relationship is essential to building a strong working alliance, which is critical for increasing the likelihood of a successful outcome [61]. Therapists need to make use of a wide range of skills and theory-based health strategies to help evoke emotions in clients that enhance motivation toward behavior change. These skills and strategies could potentially be utilized by computational agents. While the most effective way of helping someone to change problematic behavior is often a face-to-face interaction with a highly-trained and experienced (human) health expert, this approach can only have a small impact within a large population of people, as therapists are limited in the number of people they can see and help. Attempts to automate such behavior change techniques have been applied through using a wide-range of media (for instance, desktop computers, telecommunications and mobile devices) to a number of different behaviors (such as nutrition, exercise, and smoking) with varying degrees of success [14, 49, 50, 57, 93]. For example, MoodGym [76] is a Cognitive behavior Therapy (CBT) website aimed at young people for the treatment of depression and anxiety and/or as an adjunct to therapy.

Therapists and Counselling Services have also started to provide computer mediated counselling and therapy through the use of email, instant messaging, video-conferencing, and virtual environments [62, 99], but there is still a limit on the number of people a single therapist can help. One potentially fruitful avenue that has received little attention to date is in the development of affective embodied agents that attempt to closely simulate the actions of a human therapist. Working on the premise that we treat computers as social actors [92], agents that can closely match the actions of human therapists may be able to provide many of the psychological benefits (for example, evocation of constructive emotions in clients which encourage motivation) that result from interacting with therapists. Agents of this type may also be used to help therapists in their everyday tasks. For instance, they could be used to automate the initial assessment of a client's symptoms and to assess which type of therapy (if any) might potentially help clients most.

Computer mediated therapy provides a number of advantages over more traditional forms of face-to-face therapy and many of these advantages are also likely to apply to synthetic therapists. For example, [46] suggests that computer mediated and online interventions provided by therapists can be

of great help to people who are unable to visit therapists because of physical (disabled, say), personal (for example, sexuality) or geographical issues. Moreover, some people might like the anonymity that interacting with a synthetic therapist would offer as it would enable them to avoid the anxiety related to disclosing uncomfortable feelings and emotions to human therapists and may encourage them to be more open, expressive, and honest about how they feel. Therefore, an online interaction with a synthetic therapist may provide an important opportunity to those who have reservations, fears or doubts about a face-to-face interaction with a human therapist.

4.2 Behavior Change Models

Our approach in getting affective agents to simulate the skills and strategies that human health professionals use is to make use of a behavior change model. The four most commonly used behavior change models are the Health Belief Model, Theory of Reasoned Action/Planned behavior, Social Cognitive Theory, and the Transtheoretical Model (TTM) [91]. Initial work in this area has concentrated on using the TTM and has had limited success. [10] made use of the TTM when designing *Laura* and despite finding that subjects did more exercise whilst interacting with the agent, after the experiment had reached its completion, subjects tended to return to their old habitual exercise patterns. However, the fact that people *did* change their exercise behavior whilst interacting with the agent highlights the potential for computational agents to influence people's behavior.

Overview of TTM

The TTM works on the assumption that behavior change involves people moving through a number of different stages before change is achieved. The main stages of the model are:

- *precontemplation* – when people have no intention of changing their behavior,
- *contemplation* – when people intend to change within the next six months,
- *preparation* – when individuals intend to take action within the next month,
- *action* – people who have done something to change their behavior within the past six months, and
- *maintenance* – when the desired change has remained for at least six months.

Whilst in the *maintenance* stage, there are two possible outcomes: (1) a relapse into old behavioral patterns, which usually results in moving back into one of the other stages (most often *contemplation*), or (2) termination of behavior, which is said to take place when strong urges to return to old behavioral patterns no longer exist. As well as assessing each stage of change, the model

also defines other core constructs including change processes (activities that are used to help progress through the stages), decisional balance (the ability to weigh the pros and cons of changing a behavior), and self-efficacy (the confidence felt about performing an activity). The model suggests that certain change processes are more useful at different stages. This information is particularly helpful for therapists as they can potentially help their clients more effectively through assessment of which stage a client is in, and then emphasizing the appropriate change processes for that stage.

Stage-matching (linking the correct process with the correct stage) increases the likelihood that a person will effectively progress through the different stages of change. Conversely, linking the wrong process with a stage increases the probability that people will relapse and return to their old behavioral patterns. Making use of the model within a clinical setting initially involves assessing which stage of change a client is at. This can be completed by using a number of different methods, including staging algorithms, which assess the stage people are at through the use of questionnaires [31]. The next step is to provide the client with advice and information that is appropriate for the stage they are at. For those who are in the early stages of change, it is imperative to concentrate on the need to change, not necessarily on *how* they intend to change. For those in the later stages, interventions should focus on strategies that will help maintain the new behavior.

Use of Emotion in TTM

The use of the TTM in therapy can have a huge influence on the emotions that people experience. The processes recommended for use in the *pre-action* stages, such as consciousness raising, dramatic relief, and self re-evaluation, all have the potential to evoke constructive emotions and increase motivation to change problematic behavior. By constructive emotions, it is not necessarily meant that attempts are made at only eliciting typically positive emotions (for instance happiness or satisfaction), but that emotions that are often perceived to be negative (such as fear or anxiety) can also help the change process. For example, the consciousness raising process might induce emotional feelings of fear and anxiety at the health risks associated with a high fat diet and the increased likelihood of premature death. However, these emotions do not always have a derogatory effect; they can spur people into action and help motivate them to change their unhealthy behavior. Conversely, processes such as dramatic relief and attempts at increasing self-efficacy encourage people to focus on experiencing positive emotions such as satisfaction, pleasure, and fulfilment to help increase people's confidence about changing their behavior.

Whilst some processes concentrate specifically on evoking beneficial emotions, others increase the likelihood that helpful emotions will be experienced at a later date. For example, the processes of increasing self-efficacy, social re-evaluation, and the minimization of barriers, are all unlikely to initially

evoke intense emotional responses, but their emphasis in the *pre-action* stages increases the likelihood of progress through the stages of change and thus the experiencing of constructive emotions that facilitate change. In the *action* stages, the processes concentrate more on inducing emotional feelings of determination and resolve. Processes such as coping with relapse and building 'helping' relationships focus on using strategies that will help people to feel motivated during the difficult *maintenance* stage. Other processes such as self-liberation and increasing self-efficacy are all about concentrating on the positive emotional feelings that changes in behavior evoke, in an attempt to aid motivation toward behavior change goals. Like the *pre-action* stages, there are also processes such as reinforcement and enhancing the benefits of change, which directly attempt to help people feel emotions of satisfaction and achievement, and thus facilitate forward movement through the different stages of change.

Using TTM with Synthetic Therapists

Agents that are able to autonomously and correctly determine at which stage of change a person is in, and effectively apply the appropriate processes, have the potential to induce helpful emotions in people that will enable them to change their behavior. However, an agent will also need to consider the impact of its own emotional expressions on the emotional feelings evoked in clients. For example, if a therapist was to respond with strong emotional expressions of disappointment, frustration, and anger at somebody who consistently relapses into old unhealthy behavioral patterns, this could have a detrimental effect on how motivated that person feels and might result in emotional feelings of shame, distress, and hopelessness. These feelings could escalate and inevitably result in the client leaving therapy altogether. Conversely, if a therapist's emotional responses to the relapses of clients were more supportive, understanding, and encouraging (for example, empathic responses), this could have a more positive impact on the emotional feelings experienced by clients and thus their future behavior. While this is a very basic example of how the emotional expressions of a therapist can influence the emotional feelings in clients, it is clear that a therapist's emotional expressions can have a huge influence on how successful therapy will be. For example, in physician-patient interactions, [15] found that patients generally prefer physicians who express more positive emotion. The same is also likely to apply with agents that play the role of a therapist. It is not enough for them to be able to correctly assess at which stage a client is in and to emphasize the correct processes; they also need to be able to deliver their interventions in a manner which is helpful and appropriate for the client.

Despite the effective management and manipulation of emotions in a therapist-client relationship being of fundamental importance, the role of

agent emotion simulation (within a behavior change domain) has not been explicitly studied or tested. While [10] incorporated emotional capabilities into their agent, they did not explicitly test whether it had any impact on the interaction. Instead, they tested the incorporation of a number of different relational strategies (as discussed above) into their agent, which makes it difficult to ascertain the individual impact that emotion had on subjects. Several recent studies have suggested that simulated emotion can have a psychological impact on subjects, but it is still largely unknown how strong that response is [28]. For example, a number of studies have suggested that we generally seem to rate emotional agents more positively than unemotional agents [13], but how *strong* is this influence? In human-human interaction, we are more likely to act on the advice offered by a person we like and trust than someone we dislike and distrust [25]. Does the same apply in agent-human relationships? More research is required to understand how simulation of emotion influences people's attitudes and emotions, and whether these responses can be beneficially manipulated to help assist people with behavioral change.

5 Summary

This Chapter has discussed the busy research area of affective computing, with a particular focus on how we can build emotionally intelligent computers that can aid our interactions with them. We started by detailing emotion theory that was most related to the building of emotional computers through introducing the notions of basic, culturally-specific and higher cognitive emotions. We then highlighted the influence that emotions can have on our attitudes and behavior, with particular emphasis on how they can influence our attention, memory, judgement and decision-making capabilities and creative problem solving skills. The means by which we express emotion through written language, speech, facial expressions and body language were also described. To conclude Sect. 2, we detailed the different approaches taken when attempting to build emotionally intelligent computers. These included building computational models of emotion, enabling computers to autonomously detect user emotion, and simulating human emotional expressions (through 3D graphics, synthetic speech, and so forth), as well as highlighting some of the ethical issues involved in building emotional computers.

In Sect. 3 we discussed research related to evaluating affective embodied agents over extended periods of interaction. This included defining what was meant by the term 'affective embodied agents' and discussing research which has investigated how we respond to synthetic displays of emotion. Following this, we highlighted the importance of conducting longitudinal studies when developing emotionally intelligent agents and also described our own affective embodied agent that we have developed for experimental purposes.

In Sect. 4 we discussed the application of our approach to the real world by describing how such an agent could make use of a behavior change model to simulate a human health professional, to help people change problematic behavior such as smoking, eating, and (lack of) exercise.

When attempting to build emotional capabilities into computers, it is essential to consider how this will influence their functioning and our interactions with them. Emotional computers present both opportunities and dangers and it is imperative that we concentrate on how we can develop applications and systems that aid interaction, and discuss fully the issues and concerns related to the dangers of such computers. The goal of building emotionally intelligent computers is an extremely complex and difficult one, but nonetheless, a worthy goal that can enhance human-computer relationships, making them more productive, satisfying and enjoyable.

References

1. Banse R, Scherer K (1996) Acoustic profiles in vocal emotion expression. *J. Personality and Social Psychology*, 70: 614–636.
2. Baron RA (1996) Interviewer's mood and reaction to job applicants. *J. Applied Social Psychology*, 17: 911–926.
3. Bartneck C (2001) How Convincing is Mr. Data's Smile: affective expressions of machines. *User Modeling and User-Adapted Interaction*, 11: 279–295.
4. Bartneck C (2002) eMuu – an embodied emotional character for the ambient intelligent home. *PhD Thesis*, Eindhoven University of Technology, Eindhoven.
5. Bartneck C (2002) Integrating the OCC model of emotions in embodied characters. *Proc. Workshop on Virtual Conversational Characters: Applications, Methods, and Research Challenges* (available online at: http://www.bartneck.de/work/bartneck_hf2002.pdf – last accessed: 14 October 2006).
6. Bates J (1994) How Convincing is Mr. Data's Smile: affective expressions of machines. *Communications ACM*, 37(7): 122–125.
7. Beskow J, McGlashan S (1997) Olga – a conversational agent with gestures. In: André E, et al. (eds) *Proc. Intl. Joint Conf. AI'97, Workshop on Animated Interface Agents – Making them Intelligent*. 25 August, Nagoya, Japan. Morgan Kaufmann, San Francisco, CA.
8. Bickmore T (2003) Relational agents: effecting change through human-computer relationships. *PhD Thesis*, Department of Media Arts and Sciences, Massachusetts Institute of Technology.
9. Bickmore T, Cassell J (2001) Relational agents: a model and implementation of building user trust. In: Beaudouin-Lafon M, Jacob R (eds) *Proc. ACM CHI 2001 – Human Factors in Computing Systems Conf.*, 31 March–5 April, Seattle, WA. ACM Press, New York, NY: 396–403.
10. Bickmore T, Picard R (2005) Establishing and maintaining long-term human-computer relationships. *ACM Trans. Computer-Human Interaction*, 12(2): 293–327.
11. Bosseler A, Massaro DW (2003) Development and evaluation of a computer-animated tutor for vocabulary and language learning in children with autism. *J. Autism and Developmental Disorders*, 33(6): 653–672.

12. Bower GH (1981) Mood and memory. *American Psychologist*, 36: 129–148.
13. Brave S, Nass C, Hutchinson K (2005) Computers that care: investigating the effects of orientation of emotion exhibited by an embodied computer agent. *Intl. J. Human-Computer Studies*, 62(2): 161–178.
14. Brug J, Steenhuis I, Assema PV, Vries HD (1996) The Impact of a Computer-Tailored Nutrition Intervention. *Preventive Medicine*, 25(52): 236–242.
15. Buller D, Street R (1992) Physician-patient relationships. In: Feldman R (ed) *Application of Nonverbal behavioral Theories and Research*. Lawrence Erlbaum, Hillsdale, NJ: 119–141.
16. Bursleson W, Picard R (2004) Affective agents: sustaining motivation to learn through failure and a state of stuck. *Proc. Social and Emotional Intelligence in Learning Environments Workshop*(in conjunction with the 7th Intl. Conf. Intelligent Tutoring Systems), 31 August, available online at: <http://affect.media.mit.edu/pdfs/04.bursleson-picard.pdf> (last accessed: 14 October 2006).
17. Cacioppo JT, Bernston GG, Klein DJ, Poehlmann KM (1997) Psychophysiology of emotion across the life span. *Annual Review of Gerontology and Geriatrics*, 17: 27–74.
18. Cacioppo JT, Bernston GG, Larsen JT, Poehlmann KM, Ito TA (2000) The psychophysiology of emotion. In: Lewis M, Haviland-Jone (eds) *Handbook of Emotions*. The Guildford Press, New York, NY: 173–191.
19. Cahn J (1990) The generation of affect in synthesised speech. *J. American Voice I/O Society*, 8: 1–19.
20. Canamero LD (ed) (1998) *Emotional and Intelligent: The Tangled Knot of Cognition (Papers from the 1998 AAAI Fall Symposium)*. AAAI Press, Menlo Park, CA.
21. Canamero LD (ed) (2001) *Emotional and Intelligent II: The Tangled Knot of Social Cognition (Papers from the 2001 AAAI Fall Symposium)*. AAAI Press, Menlo Park, CA.
22. Cassell J, Stocky T, Bickmore T, Gao Y, Nakano Y, Ryokai K, Tversky D, Vilhjalmsson CVH (2002) Mack: Media lab autonomous conversational kiosk. *Proc. Intl. Festival for Digital Images – IMAGINA '02*, 12–15 February, available online at: <http://www.soc.northwestern.edu/justine/publications/imagina02.pdf> (last accessed: 14 October 2006).
23. Cassell J, Sullivan J, Prevost S, Churchill E (eds) (2000) *Embodied Conversational Agents*. MIT Press, Cambridge, MA.
24. Christianson SA, Loftus E (1991) Remembering emotional events: the fate of detailed information. *Cognition and Emotion*, 5: 81–108.
25. Cialdini R (2003) *Influence: Science and Practice*. Allyn and Bacon, Boston, MA.
26. Cowie R, Douglas-Cowie E (1996) Automatic statistical analysis of the signal and prosodic signs of emotion in speech. In: Bunnell T, Idsardi W (eds) *Proc. 4th Intl. Conf. Spoken Language Processing*. 3–6 October, Philadelphia, PA. IEEE Press, Piscataway, NJ: 1989–1992.
27. Cowie R, Douglas-Cowie E, Tsapatsoulis N, Votsis G, Kollias S, Fellenz W (2001) Emotion recognition in human-computer interaction. *IEEE Signal Processing Magazine*, 18(1): 32–80.
28. Creed C (2006) Using Computational Agents to Motivate Diet Change. In: IJsselsteijn W, de Kort Y, Midden C, van den Hoven E (eds) *Proc. 1st Intl.*

- Conf. Persuasive Technology for Human Well-being*, 18–19 May, Eindhoven University of Technology, the Netherlands. Springer, Berlin: 100–103.
29. Creed C, Beale R (2006) Agent Abuse: The Potential Dangers of Socially Intelligent Embodied Agents. In: De Angeli A, et al. (eds) *Proc. Workshop on Misuse and Abuse of Interactive Technologies* (in cooperation with Conf. Human Factors in Computing Systems – CHI2006). 22 April, Montreal, Canada: 17–20.
 30. Creed C, Beale R (2006) Embodied Interfaces: The Next Generation of HCI? In: Jacobs R (ed) *Proc. Workshop on The Next Generation of HCI* (in cooperation with Conf. Human Factors in Computing Systems – CHI2006). 23 April, Tufts University, Montreal, Canada: 96–99.
 31. Curry S, Kristal A, Bowen D (1992) An application of the stage model of behavior change to dietary fat reduction. *Health Education Research*, 7: 97–105.
 32. Damasio A (1994) *Descartes Error*. Macmillan, London, UK.
 33. Davidson RJ (1994) On emotion, mood, and other related affective constructs. In: Ekman P, Davidson RJ (eds) *The Nature of Emotion: Fundamental Questions*. Oxford University Press, New York, NY: 51–55.
 34. DAZ-Productions (2006) Mimic 3. <http://www.daz3d.com/Mimic/poser.php> (last accessed: 2 August 2006).
 35. de Rosis F (2002) Toward merging cognition and affect in HCI. *Applied Artificial Intelligence*, 16(7–8): 487–494.
 36. Duncker K (1945) On problem-solving. *Psychological Monographs*, 85(5): 1–113.
 37. E-Frontier (2006) Poser 5 Software. <http://www.e-frontier.com/> (last accessed: 2 August 2006).
 38. Ekman P (1972) Universals and cultural differences in facial expressions of emotion. In: Cole J (ed) *Proc. Nebraska Symposium on Motivation*, University of Nebraska Press, Lincoln: 207–283.
 39. Ekman P (1994) All emotions are basic. In: Ekman P, Davidson RJ (eds) *The Nature of Emotion: Fundamental Questions*. Oxford University Press, New York, NY: 7–19.
 40. Ekman P, Davidson RJ (eds) *The Nature of Emotion: Fundamental Questions*. Oxford University Press, New York, NY.
 41. Ekman P, Friesen WV (1975) *Unmasking the Face*. Prentice Hall, Englewood Cliffs, NJ.
 42. Ekman P, Friesen WV (1977) *Facial Action Coding System*. Consulting Psychologists Press, Palo Alto, CA.
 43. Essa IA, Pentland AP (1997) Coding, analysis, interpretation, and recognition of facial expressions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(7): 757–763.
 44. Etcoff NL, Magee JJ (1992) Categorical perception of facial expressions. *Cognition*, 44: 227–240.
 45. Evans D (2001) *Emotion: the Science of Sentiment*. Oxford University Press, New York, NY.
 46. Fenichel M, Suler JR, Barak A, Zelvin E, Jones G, Munro K, Meunier V, Walker-Schmucker W (2002) Myths and realities of online clinical work. *CyberPsychology and behavior*, 5(5): 481–497.

47. Foa EB, Feske U, Murdoch TB, Kozak MJ, McCarthy PR (1989) Processing of threat-related information in rape victims. *J. Abnormal Psychology*, 45: 1183–1187.
48. Fogg BJ, Nass C (1997) Silicon sycophants: the effects of computers that flatter. *Intl. J. Human Computer Studies*, 46(5): 551–561.
49. Friedman RH (1997) Automated Telephone Conversations to Assess Health Behavior and Deliver Behavioral Interventions. *J. Medical Systems*, 22(2): 95–102.
50. Friedman RH, Stollerman J, Mahoney D, Rozenblyum L (1997) The Virtual Visit: Using Telecommunications Technology to Take Care of Patients. *J. American Medical Informatics Association*, 4(5): 413–425.
51. Frijda N (1994) Varieties of affect: emotions and episodes, moods and sentiments. In: Ekman P, Davidson RJ (eds) *The Nature of Emotion: Fundamental Questions*. Oxford University Press, New York, NY: 59–67.
52. Givens DB (2002) *The Nonverbal Dictionary of Gestures, Signs, and Body Language Cues*. Center for Nonverbal Studies Press, Washington, DC.
53. Goleman D (1996) *Emotional Intelligence: Why it can matter more than IQ*. Bloomsbury Publishing, London, UK.
54. Goleman D (2004) *Emotional Intelligence: Why it can matter more than IQ & Working with Emotional Intelligence*. Bloomsbury Publishing, London, UK.
55. Gratch J, Marsells S (2004) A domain-independent framework for modeling emotion. *J. Cognitive Systems Research*, 5(4): 269–306.
56. Griffiths P (1997) *What Emotions Really Are*. The University of Chicago Press, Chicago, IL.
57. Hirst G, Dimarco C, Hovy E, Parsons K (1997) Authoring and generating health-education documents that are tailored to the needs of the individual patient. In: Jameson A, Paris C, Tasso C (eds) *Proc. 6th Intl. Conf. User Modeling – UM97*. 2–5 June, Vienna, Austria. Springer-Verlag, Berlin: 107–118.
58. Isen AM, Daubman KA, Nowicki GP (1987) Positive affect facilitates creative problem solving. *J. Personality and Social Psychology*, 52(6): 1122–1131.
59. Isen AM, Rosenzweig AS, Young MJ (1991) The influence of positive affect on clinical problem solving. *Medical Decision Making*, 11: 221–227.
60. Izard CE (1971) *The Face of Emotion*. Appleton-Century Crofts, New York, NY.
61. Jacobs M (1999) *Psychodynamic Counselling in Action*. Sage Publications, London, UK.
62. Kids Helpline <http://kidshelp.com.au/home-KHL.aspx?S=6> (last accessed 14 August 2006).
63. Klein J, Moon Y, Picard R (2002) This computer responds to user frustration: theory, design, and results. *Interacting with Computers*, 14(2): 119–140.
64. Lang PJ (1995) The emotion probe. *American Psychologist*, 50(5): 372–385.
65. Ledoux J (1996) *The Emotional Brain*. Simon and Schuster, New York, NY.
66. Lester J, Converse S, Kahler S, Barlow T, Stone B, Bhogal R (1997) The persona effect: affective impact of animated pedagogical agents. In: Pemberton S (ed) *Proc. SIGCHI Conf. Human Factors in Computing Systems – CHI '97* 22–27 March, Atlanta, GA. ACM Press, New York: 359–366.
67. Lester JC, Stone B, Stelling G (1999) Lifelike pedagogical agents for mixed-initiative problem solving in constructivist learning environments. *User Modeling and User-Adapted Interaction*, 9(1–2): 1–44.

68. Lisetti CL, Schiano DJ (2000) Automatic facial expression interpretation: where human-computer interaction, artificial intelligence and cognitive science intersect. *Pragmatics and Cognition* (Special Issue on Facial Information Processing: A Multidisciplinary Perspective), 8(1): 185–235.
69. Liu K, Pickard R (2005) Embedded Empathy in Continuous, Interactive Health Assessment. *Proc. Computer-Human Interaction Workshop on Computer-Human Interaction Challenges in Health Assessment*. 4 April (available online at: <http://affect.media.mit.edu/pdfs/05.liu-picard.pdf> – last accessed: 14 October 2006).
70. Lyons M, Kamachi M, Gyoba J (1998) Coding facial expressions with gabor wavelets. In: Yachida M (ed) *Proc. 3rd IEEE Intl. Conf. Automatic Face and Gesture Recognition*. 14–16 April, Nara, Japan: 200–205.
71. Mackie DM, Worth LT (1989) Processing deficits and the mediation of positive affect in persuasion. *J. Personality and Social Psychology*, 57: 27–40.
72. Mayer JD, Caruso DR, Salovey P (2000) Emotional Intelligence Meets Traditional Standards for an Intelligence. *Intelligence*, 27(4): 267–298.
73. Mayer JD, Salovey P (1997) What is emotional intelligence? In: Salovey P, Sluyter D (eds) *Emotional Development and Emotional Intelligence: Educational Implications*. Basic Books, New York, NY.
74. McGilloway S, Cowie R, Douglas-Cowie E, Gielen S, Westerdijk M, Stroeve S (2000) Approaching Automatic Recognition of Emotion from Voice: A Rough Benchmark. In: Cowie R, Douglas-Cowie E, Schroder M (eds) *Proc. Intl. Speech Communication Association (ISCA) – Workshop on Speech and Emotion*. 5–7 September, Newcastle, UK: 207–212.
75. McNair DM, Lorr M, Droppleman LF (1981) *Manual of the Profile of Mood States*. Educational and Industrial Testing Services, San Diego, CA.
76. MoodGym <http://www.moodgym.anu.edu.au/> (last accessed: 14 August 2006).
77. Murray IR, Arnott JL (1993) Toward the simulation of emotion in synthetic speech: a review of the literature on human vocal emotion. *J. Acoustical Society of America*, 93(2): 1097–1108.
78. Nass C, Brave S (2005) *Wired for Speech*. MIT Press, Cambridge, MA.
79. Nass C, Lee KM (2001) Does computer-synthesized speech manifest personality? Experimental tests of recognition, similarity-attraction, and consistency-attraction. *J. Experimental Psychology: Applied*, 7(3): 171–181.
80. Nass C, Moon Y, Fogg BJ, Reeves B, Dryer DC (1995) Can computer personalities be human personalities? *Intl. J. Human Computer Studies*, 43(2): 223–239.
81. Nass C, Moon Y, Fogg BJ, Reeves B, Dryer DC (1999) Are respondents polite to computers? social desirability and direct responses to computers. *J. Applied Social Psychology*, 29(5): 1093–1110.
82. Oatley K, Jenkins JM (1996) *Understanding Emotions*. Educational and Industrial Testing Blackwell Publishers, Oxford, UK.
83. Ortony A, Clore G, Collins A (1988) *The Cognitive Structure of Emotions*. Cambridge University Press, Cambridge, UK.
84. Pavia A (ed) (2000) *Affective Interactions: Towards a New Generation of Computer Interfaces*. Springer-Verlag, Berlin.
85. Paleari M, Lisetti CL (2006) Psychologically Grounded Avatars Expressions. In: Reichardt D, Levi P, Meyer C (eds) *Emotion and Computing – Current Research and Future Impact* (Workshop at 29th Annual German Conf. Artificial

- Intelligence – KI 2006). 19 June, Bremen, Germany. University of Bremen: 39–42.
86. Perlin K (1997) Responsive Face. <http://mrl.nyu.edu/perlin/facedemo/> (last accessed: 2 August 2006).
 87. Picard R (1997) *Affective Computing*. MIT Press, Cambridge MA.
 88. Picard R, Vyzas E, Healy J (2001) Toward Machine Emotional Intelligence: Analysis of Affective Physiological State. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(10):1175–1191.
 89. Picard RW, Klein J (2002) Computers that recognise and respond to user emotion: theoretical and practical implications. *Interacting with Computers*, 14(2): 141–169.
 90. Plantect P (2004) *Virtual Humans*. Amacon, New York, NY.
 91. Redding CA, Rossi JS, Rossi SR, Velicer WF, Prochaska JO (2000) Health behavior Models. *Intl. Electronic J. Health Education*, 3: 180–193.
 92. Reeves B, Nass C (1996) *The media equation: How people treat computers, televisions, and new media like real people and places*. Cambridge University Press, Cambridge, UK.
 93. Revere D, Dunbar P (2001) Review of oomputer-generated outpatient health behavior interventions: clinical encounters ‘in absentia’. *J. American Medical Informatics Association*, 8(1): 62–79.
 94. Rickel J, Johnson WL (1999) Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13: 343–382.
 95. Salovey P, Mayer JD (1990) Emotional Intelligence. *Imagination, Cognition and Personality*, 9: 185–211.
 96. Scheirer J, Fernandez R, Klein J, Picard RW (2002) Frustrating the user on purpose: a step towards building an affective computer. *Interacting with Computers*, 14: 93–118.
 97. Sloman A (ed) (1999) *Architectural requirements for human-like agents both natural and artificial (what sorts of machines can love?)*. John Benjamins Publishing, Amsterdam, The Netherlands.
 98. Stroop JR (1935) Studies of interference in serial verbal reactions. *J. Experimental Psychology*, 18: 643–662.
 99. Suler JR (2000) Psychotherapy in cyberspace: a five-dimensional model of online and computer-mediated psychotherapy. *CyberPsychology and behavior*, 3(2): 151–159.
 100. Takeuchi A, Naito T (1995) Situated facial displays: towards social interaction. In: Katz IR, et al. (eds) *Proc. SIGCHI Conf. Human factors in Computing Systems – CHI ’95* Denver, CO, 7–11 May. ACM Press, New York, NY: 450–455.
 101. Thorisson KR (1999) A mind model for multimodal communicative creatures and humanoids. *Applied Artificial Intelligence*, 13(4–5): 449–486.
 102. Tian YI, Kanade T, Cohn JF (2001) Recognizing action units for facial expression analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(2): 97–115.
 103. Trappl R, Petta P (eds) (1997) *Creating Personalities for Synthetic Actors*. Springer-Verlag, New York, NY.
 104. Trappl R, Petta P, Payr S (eds) (2003) *Emotions in Humans and Artifacts*. MIT Press, Cambridge, MA.

105. van Mulken S, André E, Müller (1998) The persona effect: how substantial is it? In: Johnson H, Laurence N, Roast C (eds) *Proc. HCI Conf. on People and Computers XIII - HCI '98* 1-4 September, Sheffield, UK. Springer-Verlag, London: 53-66.
106. Wang H, Predinger H, Igarashi T (2004) Communicating Emotions in Online Chat Using Physiological Sensors and Animated Text. In: Dykstra-Erickson E, Tscheligi M (eds) *Human Factors in Computing Systems - Late Breaking Results*. (Proc. ACM CHI 2004 Conf.) 24-29 April, Vienna, Austria, ACM Press, New York, NY: 1171-1174.

Resources

1 Key Books

Cassell J, et al. (eds) (2000) *Embodied Conversational Agents*. MIT Press, Cambridge, MA.

Damasio A (1994) *Descartes Error*. Macmillam Publishers, London, UK.

Evans D (2001) *Emotion: the Science of Sentiment*. Oxford University Press, New York, NY.

LeDoux J (1996) *The Emotional Brain*. Simon and Schuster, New York, NY.

Oatley K, Jenkins JM (1996) *Understanding Emotions*. Blackwell Publishers, Oxford, UK.

Picard R (1997) *Affective Computing*. MIT Press, Massachusetts, MA.

Plantec P (2004) *Virtual Humans*. Amacon, New York, NY.

Prendinger H, Ishizuka M (2004) *Life-Like Characters. Tools, Affective Functions, and Applications*. Springer, Berlin.

Reeves B, Nass C (1996) *The Media Equation: How People Treat Computers, Televisions, and New Media Like Real People and Places*. Cambridge University Press, New York, NY.

2 Key Survey/Review Articles

Bates J (1994) The role of emotion in believable agents. *Communications ACM*.37(7): 122–125.

Cowie R, et al. (2001) Emotion recognition in human-computer interaction. *IEEE Signal Processing Magazine* 18(1): 32–80.

Dehn D, Van Mulken S (2000) The impact of animated interface agents: a review of empirical research. *Intl. J. Human-Computer Studies* 52(1): 1–22.

Brave S, Nass C (2002) Emotion in human-computer interaction, In: Jacko JA, Sears A (eds) *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Lawrence Erlbaum Associates, Mahwah, NJ: 81–96.

Picard RW, Klein J (2002) Computers that recognise and respond to user emotion: theoretical and practical implications. *Interacting with Computers*. 14(2): 141–169.

<http://emotion-research.net/deliverables> HUMAINE (Human-Machine Interaction Network on Emotion) deliverable Dxx: Proposed exemplar and work towards it:

Scherer K et al. (2005) _____D3e: Theory of Emotion.
Douglas-Cowie E, et al. (2005) _____D5e: Data and Databases.
Kollias S, et al. (2005) _____D4d: Signals and Signs of Emotion.
Pelachaud C, et al. (2005) _____D6d: Emotion in Interaction.
Canamero L, et al. (2005) _____D7d: Emotion in Cognition and Action.
Stock O, et al. (2005) _____D8d: Communication and Emotions.
Hook K, et al. (2005) _____D9d: Usability.
Goldie P, et al. (2005) _____D10b: Interim report on ethical frameworks for emotion-oriented systems.

3 Organisations, Societies, Special Interest Groups

CHIL (Computers in the Human Loop)
<http://chil.server.de/serolet/is/101/>

COSY (Cognitive Systems for Cognitive Assistants)
<http://www.cognitivesystems.org/>

Design and Emotion Society
<http://www.designandemotion.org/>

Enactive Interfaces (EU Network of Excellence)

<http://www.reflex.lth.se/enactive/>

HUMAINE (Human-Machine Interaction Network on Emotion)

<http://emotion-research.net/>

International Society for Research on Emotion

<http://isre.org/prd/index.php>

SIMILAR (The European taskforce creating human-machine interfaces SIMILAR to human-human interfaces)

<http://www.similar.cc/>

Virtual Human (Anthropomorphic Interaction Agents)

http://www.virtual-human.org/start_en.html

4 Research Groups

Affective Computing at MIT Media Lab

<http://affect.media.mit.edu/>

Cognition and Affect Project at University of Birmingham (UK)

<http://www.cs.bham.ac.uk/research/projects/cogaff>

Geneva Emotion Research Group

<http://www.unige.ch/fapse/emotion/>

LeDoux Lab, New York University

<http://www.cns.nyu.edu/home/ledoux/>

Relational Agents Group, Northeastern University

<http://www.ccs.neu.edu/research/rag/>

RITL (Center for Research of Innovative Technologies for Learning, Florida State University)

<http://ritl.fsu.edu/>

Virtual Reality Lab, Swiss Federal Institute of Technology

<http://ligwww.epfl.ch/>

5 Discussion Groups, Forums

The Emotion Forum

<http://homepages.feis.herts.ac.uk/comqlc/emotion.html>

Emotional Intelligence Information Website

http://www.unh.edu/emotional_intelligence/

Facial Action Coding System (FACS) Manual

<http://face-and-emotion.com/dataface/facs/description.jsp>

Facial Expressions Resources Page

http://www.kasrl.org/facial_expression.html

Socially Intelligent Agents

<http://homepages.feis.herts.ac.uk/comqkd/aaai-social.html>

Stanford University Persuasive Technology Lab

<http://captology.stanford.edu/>

Virtual Humans

<http://www.ordinarymagic.com/v-people/#>

6 Key International Conferences/Workshops

ACII 2005: 1st Intl. Conf. Affective Computing and Intelligent Interaction

<http://www.affectivecomputing.org/2005/>

ACE 2006: Agent Construction and Emotions: Modeling the Cognitive Antecedents and Consequences of Emotion

<http://www.ofai.at/paolo.petta/conf/ace2006/>

Theories and Models of Emotion (HUMAINE Workshop – 2004)

<http://emotion-research.net/ws/wp3>

From Signals to Signs of Emotion and Vice Versa (HUMAINE Workshop – 2004)

<http://emotion-research.net/ws/wp4>

Data and Databases (HUMAINE Workshop – 2004)

<http://emotion-research.net/ws/wp5>

Emotion in Interaction (HUMAINE Workshop – 2005)

<http://emotion-research.net/ws/wp6/>

Emotion in Cognition and Action (HUMAINE Workshop – 2005)

<http://emotion-research.net/ws/wp7>

Emotion in Communication (HUMAINE Workshop – 2005)

<http://emotion-research.net/ws/wp8/proceedings-wswp8.pdf>

Innovative Approaches for Evaluating Affective Systems (HUMAINE Workshop – 2006)

<http://emotion-research.net/ws/wp9/>

7 (Open Source) Software

Croquet (Software for creating 3D collaborative multi-user online applications)

<http://www.opencroquet.org/>

Emofilt (Simulate emotional arousal with speech synthesis)

<http://felix.syntheticspeech.de/publications/emofiltInterspeech05.pdf>

FEELTRACE (Tool for rating the emotion expressed in audio-visual stimuli)

<http://emotion-research.net/download/Feeltrace%20Package.zip>

OpenAL (Cross Platform 3D Audio)

<http://www.openal.org/>

OpenGL (Graphics API)

<http://www.opengl.org/>

OpenMary (Open Source Emotional Text-to-Speech Synthesis System)

<http://mary.dfki.de>

TraceTools (Tools for tracing the presence of emotion)

<http://emotion-research.net/download/ECatPack.zip>

8 Data Bases

8.1 Multimodal Databases

Belfast Naturalistic Database

<http://www.idiap.ch/mmm/corpora/emotion-corpus>

ISLE project corpora

<http://isle.nis.sdu.dk/>

SMARTKOM

<http://www.phonetik.uni-muenchen.de/Bas/BasMultiModaleng.html#SmartKom>

SALAS

<http://www.image.ntua.gr/ermis/>

8.2 Face Databases

AR Face Database

http://cobweb.ecn.purdue.edu/~aleix/aleix_face_DB.html

CMU Facial Expression Database (Cohn-Kanade)

http://vasc.ri.cmu.edu/~ldb/html/face/facial_expression/index.html

CMU PIE (Pose, Illumination and Expression) Database

http://www.ri.cmu.edu/projects/project_418.html

CVL Face Database

<http://www.lrv.fri.uni-lj.si/facedb.html>

Psychological Image Collection at Stirling

<http://pics.psych.stir.ac.uk/>

Japanese Female Facial Expression (JAFFE) Database

<http://www.kasrl.org/jaffe.html>

Yale Face Database

<http://cvc.yale.edu/projects/yalefaces/yalefaces.html>

Yale Face Database B

<http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html>

Logic and Reasoning

The Paraconsistent Annotated Logic Program EVALPSN and its Application

Kazumi Nakamatsu

School of Human Science and Environment, University of Hyogo, 1-1-12 Shinzaike, Himeji 670-0092, Japan, nakamatu@shse.u-hyogo.ac.jp

1 Introduction

1.1 Background

The main purpose of paraconsistent logic is to deal with inconsistency in a framework for consistent logical systems. It has been almost six decades since the first paraconsistent logical system was proposed by [13]. It was four decades later that a family of paraconsistent logic called ‘annotated logics’ was proposed by [6, 51]. It can deal with inconsistency by introducing many truth values called ‘annotations’ that should be attached to each atomic formula, although their semantics is basically two-valued.

The paraconsistent annotated logic was developed from the viewpoint of logic programming, aiming at application to Computer Science such as the semantics for knowledge bases by [5, 14, 50]. Furthermore, in order to deal with inconsistency and non-monotonic reasoning in a framework of annotated logic programming, the paraconsistent annotated logic program [5] was developed to have ontological (strong) negation and stable model semantics [10] by [20], and was named Annotated Logic Program with Strong Negation (ALPSN for short). It has been applied to some non-monotonic systems, default logic [48], autoepistemic logic [18] and a non-monotonic ATMS (Assumption Based Truth Maintenance System) [7] as their computational models [21, 39, 40].

ALPSN can deal not only with inconsistency, such as conflict, but also non-monotonic reasoning, such as default reasoning. However, it seems to be more important and useful from a practical viewpoint to deal with conflict resolution in a logical way than just to express conflicts consistently. It is not so adequate for ALPSN to deal with conflict resolution or decision making based on conflict resolution in its own logical framework. Defeasible logic is known as one formalization for non-monotonic reasoning called defeasible reasoning that can deal with conflict resolution easily in a logical way [3, 42, 43]. However, defeasible logic cannot treat inconsistency in its syntax, and its

inference rules are too complicated to implement easily. Therefore, in order to deal with defeasible reasoning in a framework of paraconsistent annotated logic programming, a new version of ALPSN called Vector Annotated Logic Program with Strong Negation (VALPSN for short) was also proposed and applied to conflict resolution by [22–24].

It also has been shown that VALPSN provides a computational model of a defeasible logic [25]. Moreover, VALPSN has been extended to deal with deontic notions (obligation, forbiddance and permission), and this extended version of VALPSN was named Extended VALPSN (EVALPSN for short) [26, 27]. EVALPSN can deal with defeasible deontic reasoning and has been applied to various kinds of safety verification-based control, railway interlocking safety verification [30], robot action control [28, 31, 32, 41], safety verification for air traffic control [29], traffic signal control [33], discrete event control [34–36] and pipeline valve control [37, 38].

1.2 Overview

This Chapter focuses on the development of EVALPSN toward treating before-after relations and applications of EVALPSN to safety verification for process control and process order control with simple practical examples. The basic ideas of EVALPSN safety verification are that each control system has norms such as guidelines for safe control; such norms can be formulated in EVALPSN deontic expression; then the safety verification for control systems can be carried out by EVALPSN defeasible deontic reasoning. Compared to conventional safety verification methods, EVALPSN safety verification has some advantages, such as providing a formal safety verification method and a computational framework for control systems such as logic programming. We introduce an EVALPSN safety verification method for pipeline process control in this Chapter.

Considering safety verification for process control, there is an occasion in which the safety verification for process order is significant. We develop EVALPSN toward treating safety verification for process order control by providing new paraconsistent meanings to some EVALPSN annotations to express before-after relationships. Since the newly developed EVALPSN called bf(before-after)-EVALPSN can deal with before-after relations between processes systematically, it can be easily applied to safety verification for process order control just as well as EVALPSN. As far as we know there seems to be no other efficient computational tool dealing with safety verification for process order control than bf-EVALPSN. We also introduce a bf-EVALPSN safety verification method for pipeline process order control in this Chapter.

This Chapter is organized as follows: in Sect. 2, background knowledge in terms of EVALPSN, paraconsistent annotated logic, paraconsistent annotated logic programs, defeasible reasoning, defeasible deontic reasoning, and the like

are introduced, along with simple examples; Sect. 3 describes how to apply EVALPSN to safety verification for pipeline process control by way of a simple brewery pipeline process control example; lastly, in Sect. 4, bf-EVALPSN and its application to safety verification for process order control is described in terms of the same example of the brewery pipeline control.

2 Preliminary

This Section is devoted to clarifying the formal background of this Chapter. We assume that the reader is familiar with basic knowledge of classical logic and logic programming [15]. The following items are introduced as preliminary for understanding the paraconsistent logic program EVALPSN and its application:

- Paraconsistent Annotated Logics PT ,
- Paraconsistent Annotated Logic Programs,
 - GHP (Generally Horn Program),
 - ALPSN (Annotated Logic Program with Strong Negation),
 - VALPSN (Vector Annotated Logic Program with Strong Negation),
 - EVALPSN (Extended Vector Annotated Logic Program with Strong Negation),
- Stable Model Semantics for ALPSN,
- Defeasible Reasoning,
- Defeasible Deontic Reasoning.

2.1 Paraconsistent Annotated Logics PT

Here we recapitulate the syntax and semantics for propositional paraconsistent annotated logics PT proposed by [6].

Generally, a truth value called an *annotation* is attached to each atomic formula explicitly in annotated logic, and the set of annotations constitutes a complete lattice. The paraconsistent annotated logic denoted by PT has the complete lattice \mathcal{T} of annotations.

Definition 1

The primitive symbols of PT are:

1. propositional symbols $p, q, \dots, p_i, q_i, \dots;$
2. each member of \mathcal{T} is an *annotation constant* (which we may simply call an ‘annotation’);
3. ‘connectives’ and ‘parentheses’ $\wedge, \vee, \rightarrow, \neg, (,)$.

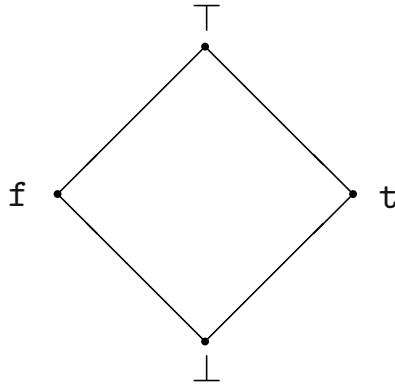


Fig. 1. The 4-valued complete lattice \mathcal{T}

Formulae are defined recursively as follows:

1. if p is a propositional symbol and $\mu \in \mathcal{T}$ is an annotation constant, then $p:\mu$ is an *annotated atomic formula (atom)*;
2. if F, F_1, F_2 are formulae, then $\neg F, F_1 \wedge F_2, F_1 \vee F_2, F_1 \rightarrow F_2$ are formulae.

We suppose the four-valued lattice in Fig.1 as the complete lattice \mathcal{T} , where the annotation \mathbf{t} may be intuitively interpreted as the truth value *true* and the annotation \mathbf{f} as the truth value *false*. It may be comprehensible that the annotations $\perp, \mathbf{t}, \mathbf{f}$ and \top correspond to the truth values $*, T, F$ and TF in [52] and **None**, **T**, **F**, and **Both** in [2], respectively. Moreover, the complete lattice \mathcal{T} can be viewed as a bi-lattice in which the vertical direction $\overrightarrow{\perp\top}$ indicates *knowledge amount* ordering and the horizontal direction $\overrightarrow{\mathbf{f}\mathbf{t}}$ does *truth* ordering [8]. We use the symbol \leq to denote the ordering in terms of knowledge amount (the vertical direction $\overrightarrow{\perp\top}$) over the complete lattice \mathcal{T} , and the symbols \perp and \top are used to denote the bottom and top elements, respectively. In the paraconsistent annotated logic PT , each annotated atomic formula can be interpreted epistemically, for example, $p:\mathbf{t}$ may be interpreted epistemically as “the proposition p is known to be true”.

There are two kinds of negation in the annotated logic, one of them represented by the symbol \neg in Definition 1 is called *epistemic negation*, and epistemic negation followed by an annotated atomic formula is defined as a mapping between the elements of the complete lattice \mathcal{T} as follows:

$$\neg(\perp) = \perp, \quad \neg(\mathbf{t}) = \mathbf{f}, \quad \neg(\mathbf{f}) = \mathbf{t}, \quad \neg(\top) = \top.$$

This definition shows that epistemic negation maps annotations to themselves without changing the knowledge amounts of the annotations, and epistemic negation followed by an annotated atomic formula can be eliminated by syntactical mapping. For example, the knowledge amount of the annotation \mathbf{t} is

the same as that of the annotation \mathbf{f} as shown in the complete lattice \mathcal{T} , and we have the epistemic negation $\neg(p:\mathbf{t}) = p:\neg(\mathbf{t}) = p:\mathbf{f}$.¹ Which shows that the knowledge amount in terms of the proposition p cannot be changed by the epistemic negation. There is another negation called *ontological (strong) negation* that is defined by the epistemic negation in PT .

Definition 2 (Strong Negation)

Let F be any formula,

$$\sim F =_{def} F \rightarrow ((F \rightarrow F) \wedge \neg(F \rightarrow F)).$$

The epistemic negation in the above definition is not interpreted as the mapping between annotations since it is not followed by an annotated atomic formula. Therefore, the strongly negated formula $\sim F$ is intuitively interpreted so that if the formula F exists, the contradiction $((F \rightarrow F) \wedge \neg(F \rightarrow F))$ is implied. Usually, strong negation is used for denying the existence of the formula following it.

The semantics for the paraconsistent annotated logics PT are defined as follows: We consider an intuitive interpretation for the strong negation of annotated atoms with the complete lattice \mathcal{T} in Fig.1. For example, the strongly negated literal $\sim(p:\mathbf{t})$ implies the knowledge “ p is false (\mathbf{f}) or unknown (\perp)” since it denies the existence of knowledge that “ p is true (\mathbf{t})”. This intuitive interpretation is proven by Definition 3 as follows: if $v_I(\sim(p:\mathbf{t})) = 1$, $v_I(p:\mathbf{t}) = 0$ and for any annotation $\mu \in \{\perp, \mathbf{f}, \mathbf{t}, \top\} \leq \mathbf{t}$, $v_I(p:\mu) = 1$, therefore $\mu = \mathbf{f}$ or $\mu = \perp$.

2.2 Generally Horn Program(GHP)

We review a basic paraconsistent annotated logic program called *Generally Horn Program* (GHP for short) introduced by [5], which will serve as the basis of ALPSN. We assume the same complete lattice \mathcal{T} in Fig.1 and its ordering \leq .

The left part A_0 and right part $A_1 \wedge \dots \wedge A_n$ of the symbol \leftarrow are called the *head* and *body* of the program clause, respectively. Furthermore, the body $A_1 \wedge \dots \wedge A_n$ is called the *definition* of the literal A_0 . A program clause $A_0 \leftarrow$ that consists of only head part is called a *unit program clause*, which is denoted by just A_0 without the symbol \leftarrow . A *logic program* is a set of program clauses.

¹ An expression $\neg p:\mu$ is conveniently used for expressing a negative annotated literal instead of $\neg(p:\mu)$ or $p:\neg(\mu)$.

Definition 3

Let ν be the set of all propositional symbols and \mathcal{F} be the set of all formulae. An interpretation I is a function,

$$I : \nu \longrightarrow \mathcal{T}.$$

To each interpretation I , we can associate the valuation function such that

$$v_I : \mathcal{F} \longrightarrow \{0, 1\},$$

which is defined as:

1. let p be a propositional symbol and μ an annotation,

$$v_I(p:\mu) = 1 \text{ iff } \mu \leq I(p),$$

$$v_I(p:\mu) = 0 \text{ iff } \mu \not\leq I(p);$$

2. let A and B be any formulae, and A not an annotated atom,

$$v_I(\neg A) = 1 \text{ iff } v_I(A) = 0,$$

$$v_I(\sim B) = 1 \text{ iff } v_I(B) = 0;$$

other formulae $A \rightarrow B, A \wedge B, A \vee B$ are evaluated as usual.

Definition 4 (annotated literal)

If A is a literal and $\mu \in \mathcal{T}$ is an annotation, then $A:\mu$ is called an *annotated literal*. If μ is one of $\{\mathbf{t}, \mathbf{f}\}$, $A:\mu$ is called a *well-annotated literal*, and μ is called a *w-annotation*.

Definition 5 (Logic Program)

Let A_0 be a positive literal and A_1, \dots, A_n literals.

$$A_0 \leftarrow A_1 \wedge \dots \wedge A_n$$

is called a *Horn clause* or a *program clause*.

In this Chapter, we use the symbol \leftarrow to distinguish with a logical connective (implication) in the paraconsistent annotated logics PT in program clauses as far as no confusion occurs. For example, the set $\{p, q \leftarrow p\}$ is a logic program and p is a unit program clause. The paraconsistent annotated logic program GHP is defined as follows:

Definition 6 (Generally-Horn Program)

If A_0, \dots, A_n are literals and μ_0, \dots, μ_n are w-annotations,

$$A_0 : \mu_0 \leftarrow A_1 : \mu_1 \wedge \dots \wedge A_n : \mu_n$$

is called a *generalized Horn clause* (gh-clause for short). A *generally-Horn program* (GHP for short) is a set of gh-clauses.

Herbrand-like interpretation [15] is considered for GHP as its semantics. The universe of individuals in the interpretation consists of all ground terms of the language being interpreted. Let the set of individuals in models and interpretations be a Herbrand universe. A Herbrand interpretation I can be regarded as an interpretation such that

$$I : B_P \longrightarrow \mathcal{T},$$

where B_P is the Herbrand base (the set of all variable-free atoms) of the GHP P . The complete lattice of annotations and the epistemic negation in GHP are defined as well as the paraconsistent annotated logic PT . Usually, an interpretation I for a GHP is denoted by the set

$$\{(p : \sqcup \mu_i) \mid I \models p : \mu_1 \wedge \dots \wedge p : \mu_n\},$$

where $\sqcup \mu_i$ is the least upper bound of the set $\{\mu_1, \dots, \mu_n\}$. Then the ordering \leq over the complete lattice \mathcal{T} is extended to an ordering over the set of interpretations.

Example 1

Consider the following GHP

$$P = \{p(a) : \mathfrak{t} \leftarrow p(b) : \mathfrak{f} \\ p(b) : \mathfrak{t} \leftarrow p(a) : \mathfrak{f}\}.$$

The GHP P has several models and some of them are listed below:

Definition 7

Let I_1 and I_2 be any interpretations for a GHP P , and A an atom in the Herbrand base B_P ,

$$I_1 \leq_I I_2 =_{def} (\forall A \in B_P)(I_1(A) \leq I_2(A)).$$

$$\begin{aligned}
I_1 : I_1(p(a)) = \mathbf{t}, \quad I_1(p(b)) = \mathbf{t} ; \\
I_2 : I_2(p(a)) = \mathbf{t}, \quad I_2(p(b)) = \mathbf{f} ; \\
I_3 : I_3(p(a)) = \mathbf{f}, \quad I_3(p(b)) = \mathbf{t} ; \\
I_4 : I_4(p(a)) = \perp, \quad I_4(p(b)) = \perp ; \\
I_5 : I_5(p(a)) = \perp, \quad I_5(p(b)) = \mathbf{t} ; \\
I_6 : I_6(p(a)) = \mathbf{t}, \quad I_6(p(b)) = \perp ; \\
I_7 : I_7(p(a)) = \top, \quad I_7(p(b)) = \top ; \\
I_8 : I_8(p(a)) = \mathbf{t}, \quad I_8(p(b)) = \top ; \\
I_9 : I_9(p(a)) = \top, \quad I_9(p(b)) = \mathbf{t}.
\end{aligned}$$

Then the GHP P has the least model I_4 and the greatest model I_7 .

Example 2

Consider the following logic program P representing the definition of even numbers and its translation P' into GHP.

$$P = \{ \text{even}(0), \\
\text{even}(s(s(x))) \leftarrow \text{even}(x), \\
q, \neg q \},$$

$$P' = \{ \text{even}(0):\mathbf{t}, \\
\text{even}(s(s(x))):\mathbf{t} \leftarrow \text{even}(x):\mathbf{t} \\
q:\mathbf{t}, q:\mathbf{f} \}.$$

The logic program P is inconsistent. Nonetheless, there is an intuition that tells us that the definition of q has nothing to do with the definition of even . Therefore, we should still be able to compute even numbers, however, classical logic program model theory does not permit us to do so. The GHP P' has the least model (even though the logic program P does not have any model) that assigns \mathbf{t} to all the atoms of the form $\text{even}(0)$, $\text{even}(s(s(0)))$, $\text{even}(s(s(s(0))))$, \dots , and assigns \top to q . It captures correctly the intuition that the definition of even is sensible, while the definition of q is inconsistent.

The fixpoint semantics [15] for GHP is defined as well as usual logic programs. Associate with each GHP P over the complete lattice \mathcal{T} , a monotonic operator T_P from Herbrand interpretations to themselves is defined.

Definition 8

Let P be a GHP, A, B_1, \dots, B_k literals, I an interpretation and μ, μ_1, \dots, μ_k annotations. Then the monotonic operator T_P is defined as:

$$T_P(I)(A) = \sqcup \{ \mu \mid A : \mu \leftarrow B_1 : \mu_1 \wedge \dots \wedge B_k : \mu_k \text{ is a ground instance of a gh-clause in } P \text{ and } I \models B_1 : \mu_1 \wedge \dots \wedge B_k : \mu_k \},$$

where the symbol \sqcup is used for denoting the least upper bound.

Definition 9

Let Δ be a special interpretation for GHP that assigns the annotation \perp to all members of B_P . Then the *upward iteration* of T_P is defined iteratively :

$$\begin{aligned} T_P \uparrow 0 &= \Delta, \\ T_P \uparrow \alpha &= T_P(T_P \uparrow (\alpha - 1)), \\ T_P \uparrow \lambda &= \sqcup \{ T_P \uparrow \eta \mid \eta < \lambda \}, \end{aligned}$$

where α is a successor ordinal and λ is a limit one.

Some well-known characteristics of the operator T_P are presented without proofs; the proofs for the following theorem are found in [5].

- Theorem 1.** 1. T_P is a monotonic function.
 2. Any GHP P has the least model \mathcal{M}_P that is identical to the least fixpoint of the function T_P .
 3. $T_P \uparrow \omega = \mathcal{M}_P$.

2.3 ALPSN (Annotated Logic Program with Strong Negation) and Stable Model Semantics

Now we introduce the Annotated Logic Program with Strong Negation (ALPSN for short), which is obtained by merging the strong negation into GHP, and the stable model semantics for ALPSN [20].

Definition 10 (ALPSN)

Let L_0, \dots, L_n be well-annotated literals over the complete lattice \mathcal{T} in Fig. 1, then

$$L_0 \leftarrow L_1 \wedge \dots \wedge L_i \wedge \sim L_{i+1} \wedge \dots \wedge L_n$$

is called an *annotated clause with strong negation* clause (ALPSN clause or asn-clause for short), where the symbol \sim is the strong negation. ALPSN is defined as a finite set of ALPSN-clauses.

Note: we assume that ALPSN is a set of ground ALPSN-clauses, then there is no loss of generality in making this assumption, since any logic program in the sense of [15] may be viewed as such a set of ALPSN-clauses by instantiating all variables occurring in the ALPSN-clauses.

The stable model semantics for general logic program (logic program with strong negation) was proposed by [10] and has been taken up as the general semantics for various non-monotonic reasonings [44]. Here we extend the stable model semantics for general logic program to ALPSN. First of all, in order to eliminate the strong negation in ALPSN the Gelfond-Lifschitz transformation for general logic program is modified.

Definition 11 (Gelfond-Lifschitz Transformation for ALPSN)

Let I be a Herbrand interpretation for an ALPSN P . Then P^I , the Gelfond-Lifschitz transformation of the ALPSN P with respect to I , is a GHP obtained from the ALPSN P by deleting

1. each ALPSN-clause that has a strongly negated annotated literal $\sim(C:\mu)$ in its body with $I \models C:\mu$, and
2. all strongly negated annotated literals in bodies of the remaining ALPSN-clauses.

The the Gelfond-Lifschitz transformation P^I has the unique least model given by $T_{P^I} \uparrow \omega$ (See. Theorem 2.1) since it includes neither epistemic nor strong negations. Then the stable model for ALPSN is defined as follows:

Definition 12 (Stable Model for ALPSN)

Let I be a Herbrand interpretation for an ALPSN P . Let I be a Herbrand interpretation of an ALPSN P ,

$$I \text{ is called a stable model of } P \quad \text{iff} \quad I = T_{P^I} \uparrow \omega$$

We have shown that ALPSN can provide annotated semantics for some non-monotonic logics such as Reiter's default logic [48] in [20, 21, 40]. For example, we proposed a translation from Reiter's default theory into ALPSN and proved that there is a one-to-one correspondence between the extension of the original default theory and the stable model for the ALPSN translation. This result shows that default theory extension can be computed by the corresponding ALPSN stable model computation. However, it is not so appropriate for formalizing the semantics for other kinds of non-monotonic reasoning, such as defeasible reasoning.

2.4 VALPSN (Vector Annotated Logic Program with Strong Negation)

A new version of ALPSN called Vector Annotated Logic Program with Strong Negation (VALPSN for short), which is able to deal with defeasible reasoning, is introduced in this Section. Annotations in ALPSN represent simple truth values such as **t**(true) and **f**(false) in ALPSN. On the other hand, annotations in VALPSN called *vector annotations* are 2-dimensional vectors such as (i, j) whose components i and j are non-negative integers.

Definition 13 (Vector Annotation)

A vector annotation is a 2-dimensional vector whose components are non-negative integers, and the complete lattice $\mathcal{T}_v(n)$ of vector annotations is defined as follows:

$$\mathcal{T}_v(n) = \{(i, j) \mid 0 \leq i \leq n, 0 \leq j \leq n, i, j \text{ and } n \text{ are integers}\}.$$

The ordering of the lattice $\mathcal{T}_v(n)$ is denoted by the symbol \preceq_v and defined as follows : let $\mathbf{v}_1 = (x_1, y_1)$ and $\mathbf{v}_2 = (x_2, y_2)$ be vector annotations,

$$\mathbf{v}_1 \preceq_v \mathbf{v}_2 \quad \text{iff} \quad x_1 \leq x_2 \quad \text{and} \quad y_1 \leq y_2.$$

The first component i of the vector annotation (i, j) denotes the amount of positive information to support the literal and the second one j does the amount of negative one as well. Vector annotated literals also can be interpreted epistemically as well as usual annotated literals in ALPSN. For example, the vector annotated literal $p : (2, 0)$ can be interpreted positively as “ p is known to be true of strength 2”, the vector annotated literal $p : (0, 1)$ negatively as “ p is known to be false of strength 1”, the vector annotated literal $p : (2, 1)$ paraconsistently as “ p is known to be true of strength 2 and false of strength 1”, and the vector annotated literal $p : (0, 0)$ as “ p is known to be neither true nor false” (there is no information in terms of the literal p). Therefore, the epistemic negation for vector annotated literals can be defined as a mapping to exchange the first and second components of vector annotations.

Definition 14 (Epistemic Negation in VALPSN)

Let (i, j) be a vector annotation. The epistemic negation \neg for vector annotated literals is defined as the following mapping over the complete lattice $\mathcal{T}_v(n)$,

$$\neg(i, j) = (j, i)$$

The epistemic negations followed by vector annotated literals can be eliminated as well as the case of ALPSN. VALPSN is defined.

Definition 15 (Well Vector Annotated Literal)

Let p be a literal.

$$p:(i, 0) \quad \text{or} \quad p:(0, j)$$

are called *well vector annotated literals* (wva-literals for short), where i and j are positive integers.

Defeasible logic is known as a non-monotonic formalism that can deal with defeasible reasoning [9]. A defeasible logic was firstly introduced by [42]. Since then defeasible logic has been developed by [3, 4, 43, 44]. It has been shown that VALPSN can provide a vector annotated semantics for defeasible reasoning based on a syntactical translation from Billington's defeasible theory [4] into VALPSN [25]. We will show details of the relation between defeasible reasoning and VALPSN after introducing EVALPSN.

Definition 16 (VALPSN)

Let L_0, \dots, L_n be well vector annotated literals over the complete lattice $\mathcal{T}_v(n)$, then,

$$L_0 \leftarrow L_1 \wedge \dots \wedge L_i \wedge \sim L_{i+1} \wedge \dots \wedge L_n$$

is called a *vector annotated clause with strong negation* (VALPSN clause for short). A *Vector Annotated Logic Program with Strong Negation* (VALPSN clause for short) is a finite set of VALPSN-clauses. If a VALPSN or a VALPSN clause contain no strong negation, they may be called just a VALP or a VALP clause, respectively.

2.5 EVALPSN (Extended Vector Annotated Logic Program with Strong Negation) and Defeasible Deontic Reasoning

Deontic logic is known as one of the modal logics that reason about normative behavior by modal operators such as obligation, permission and forbiddance. it has been applied to Computer Science [16, 17] and to modeling legal argument [46]. Usually, the symbol \bigcirc is used to represent obligation in deontic logic as follows: let p be a literal, then the formulas p , $\bigcirc p$, $\bigcirc \neg p$ and $\neg \bigcirc \neg p$ denote that the literal p is a fact, obligatory, forbidden and permitted, respectively. Furthermore, a defeasible deontic logic to deal with deontic notions

and defeasible deontic reasoning has been developed by [45] We have proposed EVALPSN by extending VALPSN to deal with such deontic notions and shown that it can deal with defeasible deontic reasoning [26, 27].

An extended annotation in EVALPSN is called an *extended vector annotation* and represented in a form of $[(i, j), \mu]$. The first component (i, j) of an extended vector annotation is a vector annotation and the second one μ is an index representing $\text{fact}(\alpha)$, $\text{obligation}(\beta)$, $\text{non-obligation}(\gamma)$, $\text{unknown}(\perp)$ and $\text{paraconsistency}(*_1, *_2, *_3 \text{ and } \top)$. The ordering of \mathcal{T}_d is denoted by a symbol

Definition 17

The complete lattice $\mathcal{T}_e(n)$ of extended vector annotations is defined as the product of the two complete lattices $\mathcal{T}_v(n)$ and \mathcal{T}_d ,

$$\mathcal{T}_e(n) = \mathcal{T}_v(n) \times \mathcal{T}_d,$$

where

$$\mathcal{T}_d = \{\perp, \alpha, \beta, \gamma, *_1, *_2, *_3, \top\}.$$

\preceq_d . The complete lattices $\mathcal{T}_e(2) = \mathcal{T}_v(2) \times \mathcal{T}_d$ are described as the Hasse's diagrams in Fig. 2. The intuitive meanings of the elements in the complete lattice \mathcal{T}_d (cube) in Fig. 2 are: \perp (no knowledge), α (fact), β (obligation), γ (non-obligation), $*_1$ (both fact and obligation), $*_2$ (both obligation and non-obligation), $*_3$ (both fact and non-obligation) and \top (fact, obligation and non-obligation). The complete lattice \mathcal{T}_d also is a quatro-lattice in which the direction $\overrightarrow{\perp\top}$ measures *knowledge amount*, the direction $\overrightarrow{\gamma\beta}$ *deontic truth*, the direction $\overrightarrow{\perp*_2}$ *deontic knowledge amount* and the direction $\overrightarrow{\perp\alpha}$ *factuality*. For example, the annotation β (obligation) can be intuitively interpreted to be more obligatory than the annotation γ (non-obligation), and the annotations \perp (no knowledge) and $*_2$ (obligation and non-obligation) are deontically neutral, that is to say, it cannot be said whether they represent obligation or non-obligation.

Definition 18

The ordering for the complete lattice \mathcal{T}_e of extended vector annotations is denoted by a symbol \preceq_e and defined as follows: let $[(i_1, j_1), \mu_1]$ and $[(i_2, j_2), \mu_2]$ be extended vector annotations,

$$[(i_1, j_1), \mu_1] \preceq_e [(i_2, j_2), \mu_2] \quad \text{iff} \quad (i_1, j_1) \preceq_v (i_2, j_2) \text{ and } \mu_1 \preceq_d \mu_2.$$

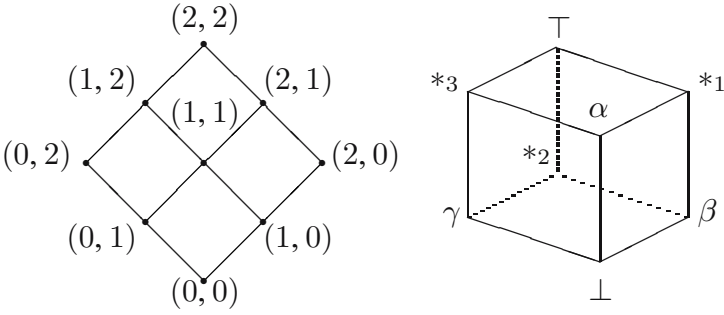


Fig. 2. The complete lattice $\mathcal{T}_e(2) = \mathcal{T}_d(2) \times \mathcal{T}_d$

EVALPSN has two kinds of epistemic negation \neg_1 and \neg_2 , which are defined as mappings over the complete lattices $\mathcal{T}_v(n)$ and \mathcal{T}_d , respectively. The epistemic negations, \neg_1 and \neg_2 , followed by extended vector annotated literals can be eliminated by the syntactic operations in the above definition,

Definition 19 (Epistemic Negations in EVALPSN, \neg_1 and \neg_2)

$$\begin{array}{ll}
 \neg_1([(i, j), \mu]) = [(j, i), \mu], & \forall \mu \in \mathcal{T}_d \\
 \neg_2([(i, j), \perp]) = [(i, j), \perp], & \neg_2([(i, j), \alpha]) = [(i, j), \alpha], \\
 \neg_2([(i, j), \beta]) = [(i, j), \gamma], & \neg_2([(i, j), \gamma]) = [(i, j), \beta], \\
 \neg_2([(i, j), *1]) = [(i, j), *3], & \neg_2([(i, j), *2]) = [(i, j), *2], \\
 \neg_2([(i, j), *3]) = [(i, j), *1], & \neg_2([(i, j), \top]) = [(i, j), \top].
 \end{array}$$

and the strong negation \sim in EVALPSN can be defined by one of the epistemic negations \neg_1 or \neg_2 (see Definition 2).

Definition 20 (Well Extended Vector Annotated Literal)

Let p be a literal.

$$p: [(i, 0), \mu] \quad \text{or} \quad p: [(0, j), \mu]$$

are called *well extended vector annotated literals* (*weva*-literals for short), where i and j are non-negative integers ($1 \leq i, j \leq n$) and $\mu \in \{ \alpha, \beta, \gamma \}$.

Definition 21 (EVALPSN)

If L_0, \dots, L_n are well extended vector annotated literals,

$$L_0 \leftarrow L_1 \wedge \dots \wedge L_i \wedge \sim L_{i+1} \wedge \dots \wedge \sim L_n$$

is called an *Extended Vector Annotated Logic Program Clause with Strong Negation* (EVALPSN clause for short). An *Extended Vector Annotated Logic Program with Strong Negation* (EVALPSN for short) is a finite set of EVALP- SN clauses. If an EVALPSN or an EVALPSN clause contain no strong negation, they may be called just an EVALP or an EVALP clause, respectively.

Facts, as well as the deontic notions obligation, forbiddance and permission, can be represented by the following extended vector annotations:

- “*fact* of strength m ” is denoted by an extended vector annotation $[(m, 0), \alpha]$,
- “*obligation* of strength m ” by an extended vector annotation $[(m, 0), \beta]$,
- “*forbiddance* of strength m ” by an extended vector annotation $[(0, m), \beta]$,
- “*permission* of strength m ” by an extended vector annotation $[(0, m), \gamma]$.

where m is a positive integer. For example, $p : [(2, 0), \alpha]$ can be intuitively interpreted as “ p is known to be a fact of strength 2”, and $q : [(0, 1), \beta]$ as “ q is known to be forbidden of strength 1”.

2.6 Defeasible Reasoning and VALPSN

First of all, we begin by briefly introducing defeasible reasoning in Nute’s defeasible deontic logic [45]. Defeasible deontic reasoning will be introduced in the following Section.

The notations and terminologies of his logic are a little bit modified from his original paper to avoid the confusion with notations in other logics.

Definition 22

If ϕ is a literal;

- $\bigcirc\phi$ and $\neg\bigcirc\phi$ are *deontic formula*,
- ϕ and $\neg\phi$ are the *complements* each other,
- $\bigcirc\phi$ and $\neg\bigcirc\phi$ are also the *complements* each other,
- $\overline{\phi}$ denotes the complement of any formula ϕ .

All and only literals and deontic formulas are formulas of the language. Rules are a class of expressions distinct from formulas, which are constructed by using three primitive symbols: \rightarrow , \Rightarrow and \rightsquigarrow . If $A \cup \{\phi\}$ is a set of formulae:

- $A \rightarrow \phi$ is a *strict rule*,
- $A \Rightarrow \phi$ is a *defeasible rule*,
- $A \rightsquigarrow \phi$ is an *undercutting defeater* (or just a *defeater*),

in each case; A is called the *antecedent* of the rule and ϕ is called the *consequent* of the rule. Antecedents for strict rules and defeaters must be non-empty, however, antecedents for defeasible rules may be empty. Such a defeasible rule has a form $\{\} \Rightarrow \phi$ and is called a *presumption*.

The intuitive interpretations of a strict rule $A \rightarrow \phi$, a defeasible rule $A \Rightarrow \phi$, a presumption $\{\} \Rightarrow \phi$ and a defeater $A \rightsquigarrow \phi$ are that ; whenever all the literals in the antecedent \mathbf{A} are accepted then the consequent ϕ must be accepted, if all the literals in the antecedent \mathbf{A} are accepted then the consequent ϕ is accepted provided that there is an insufficient evidence against the literal ϕ , if all the literals in the antecedent \mathbf{A} are accepted then $\mathbf{A} \rightsquigarrow \bar{\phi}$ is an evidence against the literal ϕ but not for the complement $\bar{\phi}$ of ϕ ,² respectively.

Definition 23 (Defeasible Theory)

A *defeasible theory* is a quadruple $\langle F, R, C, \preceq \rangle$ such that

- F is a set of formulae,
- R is a set of rules,
- C is a set of finite sets of formulae such that for every formula ϕ , either $\{\phi\} \in C$ or $\{\neg\phi\} \in C$, or $\{\phi, \neg\phi\} \in C$, the members of the set C are called *conflict sets*, and
- \preceq is an acyclic binary relation over the non-strict rules in R .

The set C in a defeasible theory represents minimal sets of conflicting formulas. Conflicting formulas may be inconsistent such as ϕ and $\bar{\phi}$. The ultimate purpose of the set C of conflict sets is to determine the sets of competing rules. Two more notions that will be used for describing defeasible reasoning are defined.

Definition 24

Let $T = \langle F, R, C, \preceq \rangle$ be a defeasible theory, and let ϕ and S be a formula and a set of formulae, respectively. Then,

- $\phi \prec_T C_\phi$ **iff** $\{\phi\} \cup C_\phi \in C$ and $C_\phi \notin C$,
- C_R covers S in T **iff** C_R is a subset of R such that for each $\phi \in S - F$, there is exactly one rule in C_R that has ϕ as its consequent.

² The role of a defeater is only to interfere with the process of drawing an inference from a defeasible rule.

Four provability relations: strict derivability, strict refutability, defeasible derivability and defeasible refutability are considered as defeasible assertions.

Let us consider what relations are there between those four defeasible assertions. For instance, we may consider deriving a formula by using only defeasible rules although the formula cannot be derived by using only strict rules. Thus, a formula might be both defeasibly derivable and strictly refutable. The following can be considered as defeasible consequence relation.

Definition 25 (Defeasible Assertion)

Let T be a defeasible theory and ϕ a literal.

- $T \vdash +\Delta\phi$ denotes that ϕ is *strictly derivable* in the defeasible theory T ,
- $T \vdash -\Delta\phi$ denotes that it is proved in the defeasible theory T that ϕ cannot be strictly derived,
- $T \vdash +\partial\phi$ denotes that ϕ is *defeasibly derivable* in the defeasible theory T and
- $T \vdash -\partial\phi$ denotes that it is proved in the defeasible theory T that ϕ cannot be defeasibly derived.

Definition 26 (Defeasible Consequence Relation)

A *defeasible consequence relation* is a set Σ of defeasible assertions such that for every defeasible theory T and every formula ϕ ,

- if $T \vdash +\Delta\phi \in \Sigma$, then $T \vdash +\partial\phi \in \Sigma$,
- if $T \vdash -\partial\phi \in \Sigma$, then $T \vdash -\Delta\phi \in \Sigma$,
- if $T \vdash +\Delta\phi \in \Sigma$, then $T \vdash -\Delta\phi \notin \Sigma$ and
- if $T \vdash +\partial\phi \in \Sigma$, then $T \vdash -\partial\phi \notin \Sigma$.

Besides the coherence conditions in Definition 26, there are other principles for defeasible reasoning. Considering condition 3.(b) in the Defeasible Derivation Principle (Definition 27), a rule that has been defeated by superior strict and defeasible rules should not be allowed to defeat any other defeasible rules. Its power as a defeater has been *preempted*, and the principle described by this condition is called *preemption*.

Next a proof theory of the defeasible logic is defined.

It has been proved that **SD** is a defeasible consequence relation in [45]. We introduce two examples for defeasible reasoning.

Example 3 – Genetically Altered Penguin [4, 45]

It is known that

- penguins (p) are definitely birds(b),
- defeasibly birds fly(f), and
- defeasibly penguins do not fly ($\neg f$).

Definition 27 (Defeasible Reasoning Principles)

Let $T = \langle F, R, C, \preceq \rangle$ be a defeasible theory.

Strict Derivation

If $\phi \in F$ or if there is $A \rightarrow \phi \in R$ such that $T \vdash +\Delta A \in \Sigma$,
then $T \vdash +\Delta\phi \in \Sigma$.

Strict Refutation

If $\phi \notin F$ and for all $A \rightarrow \phi \in R$, $T \vdash -\Delta A \in \Sigma$,
then $T \vdash -\Delta\phi \in \Sigma$.

Immediate Defeasible Derivation

If $T \vdash +\Delta\phi \in \Sigma$, then $T \vdash +\partial\phi \in \Sigma$.

Semi-strict Derivation

If there is $A \rightarrow \phi \in R$ such that

1. $T \vdash +\partial A \in \Sigma$,
 2. for all $\phi \prec_T C_\phi$, $T \vdash -\Delta C_\phi \in \Sigma$,
and
 3. for all $\phi \prec_T C_\phi$ and C_R covering C_ϕ in T such that
every member of C_R is strict,
there is $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial B \in \Sigma$,
- then $T \vdash +\partial\phi \in \Sigma$.

Defeasible Derivation

If there is $A \Rightarrow \phi \in R$ such that

1. $T \vdash +\partial A \in \Sigma$,
2. for all $\phi \prec_T C_\phi$, $T \vdash -\Delta C_\phi \in \Sigma$,
and
3. for all $\phi \prec_T C_\phi$ and C_R covering C_ϕ in T ,
either
 - (a) there is $B \rightarrow \psi \in C_R$ [$B \Rightarrow \psi \in C_R$, $B \rightsquigarrow \psi \in C_R$] such
that $T \vdash -\partial B \in \Sigma$,
 - or
 - (b) for some $B \Rightarrow \psi \in C_R$ [$B \rightsquigarrow \psi \in C_R$], $B \Rightarrow \psi \preceq A \Rightarrow \phi$
[$B \rightsquigarrow \phi \preceq A \Rightarrow \phi$],

then, $T \vdash +\partial\phi \in \Sigma$.

Defeasible Refutation

If

1. (Case that Immediate Defeasible Derivation fails) $T \vdash -\Delta\phi \in \Sigma$ and

Definition 27 (continued)

2. (Case that Semi-strict Derivation fails) for each $A \rightarrow \phi \in R$, either
 - (a) $T \vdash -\partial A \in \Sigma$,
 - (b) there is $\phi \succsim_T C_\phi$ such that $T \vdash +\Delta C_\phi$,
 - or
 - (c) there is $\phi \succsim_T C_\phi$ and C_R covering C_ϕ in T such that each rule in C_R is strict and for each $B \rightarrow \psi \in C_R$, $T \vdash +\partial B \in \Sigma$;
 and
3. (Case that Defeasible Derivation fails) for each $A \Rightarrow \phi \in R$, either
 - (a) $T \vdash -\partial A \in \Sigma$,
 - (b) there is $\phi \succsim_T C_\phi$ such that $T \vdash +\Delta C_\psi$,
 - or
 - (c) there is $\phi \succsim_T C_\phi$ and C_R covering C_ϕ in T such that
 - i. for all $B \rightarrow \psi \in C_R$ [$B \Rightarrow \psi \in C_R, B \rightsquigarrow \psi \in C_R$],
 $T \vdash +\partial B$,
 - ii. for all $B \Rightarrow \psi \in C_R$ [$B \rightsquigarrow \psi \in R$], $B \Rightarrow \psi \not\leq A \Rightarrow \phi$
[$B \rightsquigarrow \psi \not\leq A \Rightarrow \phi$],
 then, $T \vdash -\partial \phi \in \Sigma$.

Definition 28

Let σ be a finite sequence of defeasible assertions and let $k \leq \text{length}(\sigma)$.

1. $T \vdash +\Delta A$ succeeds at σ_k **iff** for every $\phi \in A$, there is
 $j < k$ such that $\sigma_j = T \vdash +\Delta \phi$;
2. $T \vdash -\Delta A$ succeeds at σ_k **iff** there is $\phi \in A$ and $j < k$
such that $\sigma_j = T \vdash -\Delta \phi$;
3. $T \vdash +\partial A$ succeeds at σ_k **iff** for every $\phi \in A$, there is $j < k$
such that $\sigma_j = T \vdash +\partial \phi$;
4. $T \vdash -\partial A$ succeeds at σ_k **iff** there is $\phi \in A$ and $j < k$
such that $\sigma_j = T \vdash -\partial \phi$;
5. $T \vdash +\Delta \bigcirc A$ succeeds at σ_k **iff** for every literal $\phi \in A$
and deontic formula $\bigcirc \phi \in A$, there is $j < k$
such that $\sigma_j = T \vdash +\Delta \bigcirc \phi$;

Definition 28 (continued)

6. $T \vdash -\Delta \circ A$ succeeds at σ_k **iff** there is a literal $\phi \in A$ or
deontic formula $\circ \phi \in A$
such that for some $j < k$, $\sigma_j = T \vdash -\Delta \circ \phi$;
7. $T \vdash +\partial \circ A$ succeeds at σ_k **iff** for every literal $\phi \in A$ and
deontic formula $\circ \phi \in A$,
there is $j < k$ such that $\sigma_j = T \vdash +\partial \circ \phi$;
8. $T \vdash -\partial \circ A$ succeeds at σ_k **iff** there is a literal $\phi \in A$ or
deontic formula $\circ \phi \in A$
such that for some $j < k$, $\sigma_j = T \vdash -\partial \circ \phi$.

Now suppose that there is a penguin with large wings and flight muscles. Such a genetically altered penguin (*gap*) might fly. Suppose that $\text{Opus}(o)$ is a genetically altered penguin. Then, we may have a defeasible theory $T = \langle F, R, C, \preceq \rangle$ with

$$\begin{aligned}
F &= \{ \mathbf{F1} : \text{gap}(o) \}, \\
R &= \{ \mathbf{R1} : \text{gap}(o) \rightarrow p(o), \quad \mathbf{R2} : p(o) \rightarrow b(o), \\
&\quad \mathbf{R3} : b(o) \Rightarrow f(o), \quad \mathbf{R4} : p(o) \Rightarrow \neg f(o), \\
&\quad \mathbf{R5} : \text{gap}(o) \rightsquigarrow f(o) \}, \\
C &= \{ \{ f(o), \neg f(o) \} \}.
\end{aligned}$$

Definition 29 (SD-Proof)

An *SD-proof* in a defeasible theory $T = \langle F, R, C, \preceq \rangle$ is a sequence σ of defeasible assertions such that for each $k \leq \text{length}(\sigma)$, one of the following conditions holds.

- [M⁺] $\sigma_k = T \vdash +\Delta\phi$ and either $\phi \in F$ or there is $A \rightarrow \phi \in R$ such that $T \vdash +\Delta A$ succeeds at σ_k .
- [M⁻] $\sigma_k = T \vdash -\Delta\phi$, $\phi \notin F$, and for each $A \rightarrow \phi \in R$, $T \vdash -\Delta A$ succeeds at σ_k .
- [E⁺] $\sigma_k = T \vdash +\partial\phi$ and $T \vdash +\Delta\phi$ succeeds at σ_k .
- [SS⁺] $\sigma_k = T \vdash +\partial\phi$ and there is $A \rightarrow \phi \in R$ such that
1. $T \vdash +\partial A$ succeeds at σ_k ,
 2. for each $\phi \succ_T C_\phi$, $T \vdash -\Delta C_\phi$ succeeds at σ_k ,
- and

Definition 29 (continued)

3. for each $\phi \succsim_T C_\phi$ and C_R covering C_ϕ in T such that every member of C_R is strict, there is $B \rightarrow \psi \in C_R$ such that
 - $T \vdash -\partial B$ succeeds at σ_k .
- [SD⁺] $\sigma_k = T \vdash +\partial\phi$ and there is $A \Rightarrow \phi \in R$ such that
 1. $T \vdash +\partial A$ succeeds at σ_k ,
 2. for each $\phi \succsim_T C_\phi$ there is $\psi \in C_\phi$ such that $T \vdash -\Delta\psi$ succeeds at σ_k , and
 3. for each $\phi \succsim_T C_\phi$ and C_R covering C_ϕ in T ,
 - either
 - (a) there is $\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$ [$B \Rightarrow \psi \in C_R$, $B \rightsquigarrow \psi \in R$] such that $T \vdash -\partial B$ succeeds at σ_k ,
 - or
 - (b) for some $B \Rightarrow \psi \in C_R$ [$B \rightsquigarrow \psi \in R$], $B \Rightarrow \psi \preceq A \Rightarrow \phi$ [$B \rightsquigarrow \psi \preceq A \Rightarrow \phi$].
- [SD⁻] $\sigma_k = T \vdash -\partial\phi$ and
 1. (Case that [E⁺] fails) $T \vdash -\Delta\phi$ succeeds at σ_k ;
 2. (Case that [SS⁺] fails) for each $A \rightarrow \phi \in R$,
 - either
 - (a) $T \vdash -\partial A$ succeeds at σ_k ,
 - (b) there is $\phi \succsim_T C_\phi$ such that $T \vdash +\Delta C_\phi$,
 - or
 - (c) there is $\phi \succsim_T C_\phi$ and C_R covering C_ϕ such that every member of C_R is strict and for each $B \rightarrow \psi \in C_R$, $T \vdash +\partial B$ succeeds at σ_k ;

and

 3. (Case that [SD⁺] fails) for each $A \Rightarrow \phi \in R$,
 - either
 - (a) $T \vdash -\partial A$ succeeds at σ_k ,
 - (b) there is $\phi \succsim_T C_\phi$ such that $T \vdash +\Delta C_\phi$ succeeds at σ_k ,
 - or
 - (c) there is $\phi \succsim_T C_\phi$ and C_R covering C_ϕ in T such that
 - i. for all $B \rightarrow \psi \in C_R$ [$B \Rightarrow \psi \in C_R$, $B \rightsquigarrow \psi \in C_R$], $T \vdash +\partial B$ succeeds at σ_k , and
 - ii. for all $B \Rightarrow \psi \in C_R$ [$B \rightsquigarrow \psi \in R$], $B \Rightarrow \psi \not\preceq A \Rightarrow \phi$ [$B \rightsquigarrow \psi \not\preceq A \Rightarrow \phi$].

There are superiority relations $\mathbf{R4} \preceq \mathbf{R5}$ and $\mathbf{R3} \preceq \mathbf{R4}$. We show an intuitive derivation.

1. $T \vdash +\Delta gap(o)$ by [M⁺], with the fact **F1**.
2. $T \vdash +\Delta p(o)$ by [M⁺], with 1 and the strict rule **R1**.
3. $T \vdash +\Delta b(o)$ by [M⁺], with 2 and the strict rule **R2**.

4. Both the antecedents of the defeasible rules **R3** and **R4** are strictly derivable, however, the defeasible rule **R3** is defeated by the defeasible rule **R4** since there is the superiority relation $\mathbf{R3} \preceq \mathbf{R4}$. Thus, $T \vdash -\partial f(o)$ by $[\text{SD}^-]$.
5. Both the antecedents of the defeasible rule **R4** and the defeater **R5** are strictly derivable, however, the defeasible rule **R4** is defeated by the defeater **R5** since there is the superiority relation $\mathbf{R4} \preceq \mathbf{R5}$. Thus, $T \vdash -\partial \neg f(o)$ by $[\text{SD}^-]$.

In defeasible logic, defeaters cannot be used to derive their own consequents but to prevent their conflicting rules from deriving their conflicting consequents. Therefore, we could not obtain $T \vdash +\partial \neg f(o)$.

Let us consider another example of defeasible reasoning.

Example 4

Suppose a defeasible theory $T = \langle F, R, C, \preceq \rangle$ with two facts and two defeasible rules,

$$F = \{\mathbf{F2} : q, \mathbf{F3} : r\}, \quad R = \{\mathbf{R6} : q \Rightarrow p, \mathbf{R7} : r \Rightarrow \neg p\},$$

and the conflict sets $C = \{\{p, \neg p\}\}$. According to the superiority relation between the defeasible rules **R6** and **R7**, it is decided whether the literals p and $\neg p$ are defeasibly derivable or refutable. Apparently, $T \vdash +\Delta q$ and $T \vdash +\Delta r$ by $[\text{M}^+]$.

- If there is the superiority relation $\mathbf{R7} \preceq \mathbf{R6}$, $T \vdash +\partial p$ by $[\text{SD}^+]$ 1,2 and 3(b) since $T \vdash -\Delta \neg p$ by $[\text{M}^-]$, and $T \vdash -\partial \neg p$ by $[\text{SD}^-]$ 1, 2 and 3(c) since $T \vdash -\Delta \neg p$ by $[\text{M}^-]$.
- Conversely, if there is the superiority relation $\mathbf{R6} \preceq \mathbf{R7}$, $T \vdash +\partial \neg p$ by $[\text{SD}^+]$ 1, 2 and 3(b) since $T \vdash -\Delta p$ by $[\text{M}^-]$, and $T \vdash -\partial p$ by $[\text{SD}^-]$ 1,2 and 3(c)ii since $T \vdash -\Delta p$ by $[\text{M}^-]$.
- On the other hand, if there is no superiority relation between the defeasible rules **R6** and **R7**, both $T \vdash -\partial p$ and $T \vdash -\partial \neg p$ by $[\text{SD}^-]$ 1,2 and 3(c)ii since both $T \vdash -\Delta p$ and $T \vdash -\Delta \neg p$ by $[\text{M}^-]$, that is to say, neither the literals p nor $\neg p$ are defeasibly derivable if no superiority relation exists.

Now we will briefly introduce the basic idea of translation from defeasible theory into VALPSN. The basic idea is that: considering the defeasible consequence relations in Definition 26, if a literal is strictly derivable, there is stronger evidence to support the literal positively than defeasible derivable because if the literal is strictly derivable, it is also defeasibly derivable, but not vice versa; furthermore, if a literal is neither strictly nor defeasibly derivable, there is no evidence to support the literal; thus, we can consider three levels of information to support the literal and they can be expressed by simple integers 0, 1 and 2 in vector annotation; therefore, “a literal ϕ is strictly

derivable $+\Delta\phi$ ” can be translated to “a vector annotated literal $\phi:(2,0)$ is satisfiable” and “a literal ϕ is defeasibly derivable $+\partial\phi$ ” to “a vector annotated literal $\phi:(1,0)$ is satisfiable”.

We have already provided the syntactical translation rules from Billington’s defeasible theory into VALPSN, and proved that there are the following relations between defeasible theory derivability and VALPSN stable model satisfiability based on the translation in [25]. Let ϕ be a literal, T a defeasible theory, I an interpretation of a VALPSN P , and \mathcal{M}_P the set of all stable models of the VALPSN P . Then, we have the following relations:

$$\begin{array}{lll}
 T \vdash +\Delta\phi & \text{iff} & \forall I \in \mathcal{M}_P, I \models \phi:(2,0), \\
 T \vdash +\partial\phi & \text{iff} & \forall I \in \mathcal{M}_P, I \models \phi:(1,0), \\
 T \vdash -\Delta\phi & \text{iff} & \exists I \in \mathcal{M}_P, I \not\models \phi:(2,0) \text{ (i.e. } I \models \sim \phi:(2,0)\text{)}, \\
 T \vdash -\partial\phi & \text{iff} & \exists I \in \mathcal{M}_P, I \not\models \phi:(1,0) \text{ (i.e. } I \models \sim \phi:(1,0)\text{)}.
 \end{array}$$

We describe how to translate defeasible theory into VALPSN with taking the defeasible theory in Example 4 as an example. Details of the translation from Billington’s defeasible theory into VALPSN are provided in [25].

The literals q and r cannot be defeated by anything since they are facts, and $T \vdash +\Delta q$ and $T \vdash +\Delta r$ by $[M^+]$. Therefore, the two facts **F2** and **F3** can be translated to the VALP clauses,

$$q:(2,0) \quad \text{and} \quad r:(2,0) \tag{1}$$

respectively. In order to consider the translation of defeasible rules into VALPSN, we focus on only the defeasible rules **R6** and **R7** as an example, and suppose the superiority relation **R7** \preceq **R6**. Then, the following four cases **v1**, **v2**, **v3** and **v4** should be taken into account.

- Case v1:* if $T \vdash -\partial q$ and $T \vdash -\partial r$,
 then apparently $T \vdash -\partial p$ and $T \vdash -\partial \neg p$ by $[SD^-]$;
Case v2: if $T \vdash +\partial q$ and $T \vdash -\partial r$,
 then $T \vdash +\partial p$ by $[SD^+]$ and $T \vdash -\partial \neg p$ by $[SD^-]$ as $T \vdash -\Delta \neg p$;
Case v3: if $T \vdash -\partial q$ and $T \vdash +\partial r$,
 then $T \vdash -\partial p$ by $[SD^-]$ and $T \vdash +\partial \neg p$ by $[SD^+]$,
 even though **R7** \preceq **R6**, as $T \vdash -\Delta p$;
Case v4: if $T \vdash +\partial q$ and $T \vdash +\partial r$,
 then $T \vdash +\partial p$ by $[SD^+]$ and $T \vdash -\partial \neg p$ by $[SD^-]$
 since **R7** \preceq **R6**, as $T \vdash -\Delta p$.

Thus, the defeasible rules **R6** and **R7** can be translated to the VALPSN clauses,

$$\mathbf{R6} \quad p:(1,0) \leftarrow q:(1,0) \wedge \sim p:(0,2) \tag{2}$$

$$\mathbf{R7} \quad p:(0,1) \leftarrow r:(1,0) \wedge \sim q:(1,0) \wedge \sim p:(2,0) \tag{3}$$

Then, we have only one stable model

$$M_1 = \{q:(2, 0), r:(2, 0), p:(1, 0)\}$$

for the VALPSN $P_1 = \{(1), (2), (3)\}$. The stable model M_1 shows that q and r are strictly derivable, p is defeasibly derivable, and $\neg p$ is defeasibly refutable.

Conversely, suppose the superiority relation $\mathbf{R6} \preceq \mathbf{R7}$. Then, it is similarly inferred that the defeasible rules $\mathbf{R6}$ and $\mathbf{R7}$ can be translated to the VALPSN clauses,

$$\mathbf{R6} \quad p:(1, 0) \leftarrow q:(1, 0) \wedge \sim r:(1, 0) \wedge \sim p:(0, 2) \quad (4)$$

$$\mathbf{R7} \quad p:(0, 1) \leftarrow r:(1, 0) \wedge \sim p:(2, 0) \quad (5)$$

Then, we also have only one stable model

$$M_2 = \{q:(2, 0), r:(2, 0), p:(0, 1)\}$$

for the VALPSN $P_2 = \{(1), (4), (5)\}$. The stable model M_2 shows that q and r are strictly derivable, p is defeasibly refutable, and $\neg p$ is defeasibly derivable.

Moreover, if we suppose no superiority relation between rules $\mathbf{R6}$ and $\mathbf{R7}$, another case (a case in which both the antecedents q and r of the defeasible rules $\mathbf{R6}$ and $\mathbf{R7}$ are defeasibly derivable) should be considered instead of the case **v4**.

Case v5: if $T \vdash +\partial q$ and $T \vdash +\partial r$,
then $T \vdash -\partial p$ and $T \vdash -\partial \neg p$ by $[\text{SD}^-]$ since no superiority relation,
as $T \vdash -\Delta p$ and $T \vdash -\Delta \neg p$.

Thus, the defeasible rules $\mathbf{R6}$ and $\mathbf{R7}$ can be translated to the VALPSN clauses,

$$\mathbf{R6} \quad p:(1, 0) \leftarrow q:(1, 0) \wedge \sim r:(1, 0) \wedge \sim p:(0, 2) \quad (6)$$

$$\mathbf{R7} \quad p:(0, 1) \leftarrow r:(1, 0) \wedge \sim q:(1, 0) \wedge \sim p:(2, 0) \quad (7)$$

Then, we also have only one stable model

$$M_3 = \{q:(2, 0), r:(2, 0), p:(0, 0)\}$$

for the VALPSN $P_3 = \{(1), (6), (7)\}$. The stable model M_3 shows that both q and r are strictly derivable and both p and $\neg p$ are defeasibly refutable. The stable model M_3 also shows that defeasible reasoning is skeptical reasoning.

2.7 Defeasible Deontic Reasoning and EVALPSN

Nute's defeasible deontic reasoning principles are obtained by extending the defeasible reasoning principles in Definition 27. Let T be a defeasible theory and ϕ a literal, and $\bigcirc A$ denotes that the deontic operator \bigcirc is applied to all

the literals in A . Then the following principles are considered for defeasible deontic reasoning:

Strict Principles

$A \rightarrow \phi$	and	$T \vdash +\Delta A$	supports	$T \vdash +\Delta\phi,$
$A \rightarrow \phi$	and	$T \vdash +\Delta \circ A$	supports	$T \vdash +\Delta \circ \phi,$
$A \rightarrow \circ\phi$	and	$T \vdash +\Delta \circ A$	supports	$T \vdash +\Delta \circ \phi,$

Semi-strict Principles

$A \rightarrow \phi$	and	$T \vdash +\partial A$	supports	$T \vdash +\partial\phi,$
$A \rightarrow \phi$	and	$T \vdash +\partial \circ A$	supports	$T \vdash +\partial \circ \phi,$
$A \rightarrow \circ\phi$	and	$T \vdash +\partial \circ A$	supports	$T \vdash +\partial \circ \phi,$

Defeasible Factual Detachment

$A \Rightarrow \phi$	and	$T \vdash +\partial A$	supports	$T \vdash +\partial\phi,$
----------------------	-----	------------------------	----------	---------------------------

Defeasible Deontic Detachment

$A \Rightarrow \circ\phi$	and	$T \vdash +\partial \circ A$	supports	$T \vdash +\partial \circ \phi,$
---------------------------	-----	------------------------------	----------	----------------------------------

where the strength order of the principles is

Strict > Semi-strict > Defeasible Factual > Defeasible Deontic.

A proof theory for the defeasible deontic logic is developed by modifying the SD-proof conditions in Definition 29 and adding some new proof conditions for normative reasoning to the original proof conditions. The notion of a set of rules covering a set of literals is modified to define the new conditions. Here DSD-proof is defined as well as the SD-proof in Definition 29.

A precise statement of the proof theory is quite complicated, however the intuitions are clear.

1. Strict inferences are never defeated and defeat all non-strict competing principles. Familiar deontic inheritance is a strict inference.

Definition 30

Let $T = \langle F, R, C, \preceq \rangle$ be a defeasible theory and let S be a set of formulae.

C_R *d-covers* S in T **iff** C_R is a subset of R such that for each $\phi \in S - F$, there is exactly one ruler in C_R such that either

1. ϕ is the consequence of r , or
2. r is strict, ψ is the consequent of r , and $\phi = \circ\psi$.

2. Semi-strict principles can only be defeated by strict inferences and by other semi-strict inferences.
3. Defeasible factual detachment is weaker than any strict or semi-strict inferences. Defeasible factual detachments can also be defeated by other potential defeasible factual detachments.
4. Defeasible deontic detachment is the weakest kind of inference in the defeasible deontic logic.

Next, we briefly introduce an overview of the translation from defeasible deontic theory into EVALPSN by considering the so-called ‘Chisholm Example’ [45] – details of the translation are provided in [26]. Let ψ be a formula that has one of the forms $\{\phi, \bigcirc\phi, \neg\bigcirc\phi\}$, where ϕ is a literal, T a defeasible deontic theory, I an interpretation of the EVALPSN P translated from the defeasible deontic theory T , and \mathcal{M}_P the set of all stable models for the EVALPSN P . Then we have the following relations between defeasible deontic derivability and VALPSN stable model satisfiability:

$$\begin{array}{lll}
T \vdash +\Delta\psi & \text{iff} & \forall I \in \mathcal{M}_P, I \models \phi: [(2, 0), \mu], \\
T \vdash +\partial\psi & \text{iff} & \forall I \in \mathcal{M}_P, I \models \phi: [(1, 0), \mu], \\
T \vdash -\Delta\psi & \text{iff} & \exists I \in \mathcal{M}_P, I \not\models \phi: [(2, 0), \mu] \text{ (i.e. } I \models \sim\phi: [(2, 0), \mu] \text{)}, \\
T \vdash -\partial\psi & \text{iff} & \exists I \in \mathcal{M}_P, I \not\models \phi: [(1, 0), \mu] \text{ (i.e. } I \models \sim\phi: [(1, 0), \mu] \text{)},
\end{array}$$

provided that if $\psi = \phi$, then $\mu = \alpha$; if $\psi = \bigcirc\phi$, then $\mu = \beta$; if $\psi = \neg\bigcirc\phi$, then $\mu = \gamma$.

Definition 31 part-(i)

The condition $[M^+]$ has to be modified by supplementing a new condition incorporating deontic reasoning principles. Let $T = \langle F, R, C, \preceq \rangle$ be a defeasible theory.

$[DM^+]$ $\sigma_k = T \vdash +\Delta\bigcirc\phi$ and there is $A \rightarrow \phi \in R$ or $A \rightarrow \bigcirc\phi \in R$ such that $T \vdash \bigcirc A$ succeeds at σ_k .

The condition $[M^-]$ also has to be modified to address the case in which both $[M^+]$ and $[DM^+]$ fail.

$[DM^-]$ $\sigma_k = T \vdash -\Delta\bigcirc\phi$,

1. $\phi \notin F$,
2. for each $A \rightarrow \phi \in R$, $T \vdash -\Delta A$ succeeds at σ_k , and
3. if $\phi = \bigcirc\psi$, then for each $A \rightarrow \psi \in R$ or $A \rightarrow \bigcirc\psi \in R$, $T \vdash -\Delta\bigcirc A$ succeeds at σ_k .

The condition $[SS^+]$ is replaced by the following new conditions for semi-strict defeasible inheritance.

Definition 31 (continued)

- [SS_D⁺] $\sigma_k = T \vdash +\partial\phi$ and there is $A \rightarrow \phi \in R_T$ such that
1. $T \vdash +\partial A$ succeeds at σ_k ,
 2. for each $\phi \asymp_T C_\phi$ there is $\psi \in C_\phi$ such that $T \vdash -\Delta\psi$ succeeds at σ_k , and
 3. for each $\phi \asymp_T C_\phi$ and C_R d-covering C_ϕ in T such that every rule in C_R is strict, either
 - (a) there is a literal $\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial B$ succeeds at σ_k ,
 - (b) there is $\bigcirc\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial \bigcirc B$ succeeds at σ_k , or
 - (c) there is $\bigcirc\psi \in C_\phi$ and $B \rightarrow \bigcirc\psi \in C_R$ such that both $T \vdash -\partial B$ and $T \vdash -\partial \bigcirc B$ succeed at σ_k .
- [DSS⁺] $\sigma_k = T \vdash +\partial \bigcirc\phi$ and there is $A \rightarrow \phi \in R$ or $A \rightarrow \bigcirc\phi \in R$ such that
1. $T \vdash +\partial \bigcirc A$ succeeds at σ_k ,
 2. for each $\bigcirc\phi \asymp_T C_{\bigcirc\phi}$ there is $\psi \in C_{\bigcirc\phi}$ such that $T \vdash -\Delta\psi$ succeeds at σ_k , and
 3. for each $\bigcirc\phi \asymp_T C_{\bigcirc\phi}$ and C_R d-covering $C_{\bigcirc\phi}$ in T such that every rule in C_R is strict, either
 - (a) there is a literal $\psi \in C_{\bigcirc\phi}$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial B$ succeeds at σ_k ,
 - (b) there is $\bigcirc\psi \in C_{\bigcirc\phi}$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial \bigcirc B$ succeeds at σ_k , or
 - (c) there is $\bigcirc\psi \in C_{\bigcirc\phi}$ and $B \rightarrow \bigcirc\psi \in C_R$ such that both $T \vdash -\partial B$ and $T \vdash -\partial \bigcirc B$ succeed at σ_k .

The condition [SD⁺] should be reformulated since it can be defeated by semi-strict deontic inferences.

- [DSD⁺] $\sigma_k = T \vdash +\partial\phi$ and there is $A \Rightarrow \phi \in R$ such that
1. $T \vdash +\partial A$ succeeds at σ_k ,
 2. for each $\phi \asymp_T C_\phi$ there is $\psi \in C_\phi$ such that $T \vdash -\Delta\psi$ succeeds at σ_k , and
 3. for each $\phi \asymp_T C_\phi$ and C_R d-covering C_ϕ in T , either
 - (a) there is a literal $\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial B$ succeeds at σ_k ,
 - (b) there is $\bigcirc\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial \bigcirc B$ succeeds at σ_k ,
 - (c) there is $\bigcirc\psi \in C_\phi$ and $B \rightarrow \bigcirc\psi \in C_R$ such that both $T \vdash -\partial B$ and $T \vdash -\partial \bigcirc B$ succeed at σ_k , or
 - (d) there is $B \Rightarrow \psi \in C_R$ [$B \rightsquigarrow \psi \in C_R$] such that either
 - i. $T \vdash -\partial B$ succeeds at σ_k , or
 - ii. $B \Rightarrow \psi \preceq A \Rightarrow \phi$ [$B \rightsquigarrow \psi \preceq A \Rightarrow \phi$].

Definition 32 part-(ii)

Next, the condition for defeasible deontic detachment is formulated.

- [DDD⁺] $\sigma_k = T \vdash +\partial \bigcirc \phi$ and there is $A \Rightarrow \bigcirc \phi \in R$ such that
1. $T \vdash +\partial A$ succeeds at σ_k ,
 2. for each $\bigcirc \phi \succ_T C_{\bigcirc \phi}$ there is $\psi \in C_{\bigcirc \phi}$ such that $T \vdash -\Delta \psi$ succeeds at σ_k , and
 3. for each $\bigcirc \phi \succ_T C_{\bigcirc \phi}$ and C_R d-covering $C_{\bigcirc \phi}$ in T , either
 - (a) there is a literal $\psi \in C_{\bigcirc \phi}$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial B$ succeeds at σ_k ,
 - (b) there is $\bigcirc \psi \in C_{\bigcirc \phi}$ and $B \rightarrow \psi \in C_R$ such that $T \vdash -\partial \bigcirc B$ succeeds at σ_k ,
 - (c) there is $\bigcirc \psi \in C_{\bigcirc \phi}$ and $B \rightarrow \bigcirc \psi \in C_R$ such that both $T \vdash -\partial B$ and $T \vdash -\partial \bigcirc B$ succeed at σ_k ,
 - (d) there is $B \Rightarrow \psi \in C_R$ [$B \rightsquigarrow \psi \in C_R$] such that either
 - i. $T \vdash -\partial B$ succeeds at σ_k , or
 - ii. $B \Rightarrow \psi \preceq A \Rightarrow \bigcirc \phi$ [$B \rightsquigarrow \psi \preceq A \Rightarrow \bigcirc \phi$]
 - (e) there is $B \Rightarrow \bigcirc \psi \in C_R$ [$B \rightsquigarrow \bigcirc \psi \in C_R$] such that either
 - i. both $T \vdash -\partial B$ and $T \vdash -\partial \bigcirc B$ succeed at σ_k , or
 - ii. $B \Rightarrow \bigcirc \psi \preceq A \Rightarrow \bigcirc \phi$ [$B \rightsquigarrow \bigcirc \psi \preceq A \Rightarrow \bigcirc \phi$].

Finally, a new proof condition for defeasible refutation is defined. The condition is very complicated since the defeasible refutation must include all cases in which derivation rules succeed.

- [DSD⁻] $\sigma_k = T \vdash -\partial \phi$ and
1. (Case that [M⁺],[DM⁺] and [E⁺] fail.) $T \vdash -\Delta \phi$ succeeds at σ_k ,
 2. (Case that [SS_D⁺] fails.) for each $A \rightarrow \phi \in R$, either
 - (a) $T \vdash -\partial A$ succeeds at σ_k ,
 - (b) there is $\phi \succ_T C_\phi$ such that $T \vdash -\partial C_\phi$ succeeds at σ_k , or
 - (c) there is $\phi \succ_T C_\phi$ and C_R d-covering C_ϕ in T such that each rule in C_R is strict,
 - i. for each literal $\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$, $T \vdash +\partial B$ succeeds in σ_k ,
 - ii. for each $\bigcirc \psi \in C_\phi$ and $B \rightarrow \psi \in C_R$, $T \vdash +\partial \bigcirc B$ succeeds at σ_k , and
 - iii. for each $\bigcirc \psi \in C_\phi$ and $B \rightarrow \bigcirc \psi \in C_R$, either $T \vdash +\partial B$ or $T \vdash +\partial \bigcirc B$ succeed at σ_k ,
 3. (Case that [DSS⁺] fails) for each $A \rightarrow \psi \in R$ or $A \rightarrow \bigcirc \psi \in R$ such that $\phi = \bigcirc \psi$, either
 - (a) both $T \vdash -\partial A$ and $T \vdash -\partial \bigcirc A$ succeed at σ_k ,
 - (b) there is $\phi \succ_T C_\phi$ such that for each $\psi \in C_\phi$, $T \vdash +\Delta \psi$ succeeds at σ_k , or
 - (c) there is $\phi \succ_T C_\phi$ and C_R d-covering C_ϕ in T such that each rule in C_R is strict,

Definition 32 (continued)

- i. for each literal $\chi \in C_\phi$ and $B \rightarrow \chi \in C_R$,
 $T \vdash +\partial B$ succeeds in σ_k ,
- ii. for each $\bigcirc\chi \in C_\phi$ and $B \rightarrow \chi \in C_R$,
 $T \vdash +\partial \bigcirc B$ succeeds at σ_k , and
- iii. for each $\bigcirc\chi \in C_\phi$ and $B \rightarrow \bigcirc\chi \in C_R$,
either $T \vdash +\partial B$ or $T \vdash +\partial \bigcirc B$ succeed at σ_k ,

Definition 33 part-(iii)

- 4. (Case that [DSD⁺] fails.) for each $A \Rightarrow \phi \in R$, either
 - (a) $T \vdash -\partial A$ succeeds at σ_k ,
 - (b) there is $\phi \asymp_T C_\phi$ such that $T \vdash +\Delta C_\psi$ succeeds at σ_k ,
or
 - (c) there is $\phi \asymp_T C_\phi$ and C_R d-covering C_ϕ in T such that
 - i. for each literal $\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$,
 $T \vdash +\partial B$ succeeds at σ_k ,
 - ii. for each $\bigcirc\psi \in C_\phi$ and $B \rightarrow \psi \in C_R$,
 $T \vdash +\partial \bigcirc B$ succeeds at σ_k ,
 - iii. for each $\bigcirc\psi \in C_\phi$ and $B \rightarrow \bigcirc\psi \in C_R$,
either $T \vdash +\partial B$ or $T \vdash +\partial \bigcirc B$ succeed at σ_k , and
 - iv. for each $B \Rightarrow \psi \in C_R$ [$B \rightsquigarrow \psi \in C_R$],
 - A. $T \vdash +\partial B$ succeeds at σ_k , and
 - B. $B \Rightarrow \psi \not\leq A \Rightarrow \phi$ [$B \rightsquigarrow \psi \not\leq A \Rightarrow \phi$],
- 5. (Case that [DDD⁺] fails.) if $\phi = \bigcirc\psi$, then for each $A \Rightarrow \bigcirc\psi \in R$, either
 - (a) $T \vdash -\partial \bigcirc A$ succeeds at σ_k ,
 - (b) there is $\bigcirc\psi \asymp_T C_{\bigcirc\psi}$ such that $T \vdash +\Delta C_{\bigcirc\psi}$ succeeds at σ_k , or
 - (c) there is $\bigcirc\psi \asymp_T C_{\bigcirc\psi}$ and C_R d-covering $C_{\bigcirc\psi}$ in T such that
 - i. for each literal $\chi \in C_{\bigcirc\psi}$ and $B \rightarrow \chi \in C_R$,
 $T \vdash +\partial B$ succeeds at σ_k ,
 - ii. for each $\bigcirc\chi \in C_{\bigcirc\psi}$ and $B \rightarrow \bigcirc\chi \in C_R$,
 $T \vdash -\partial B$ succeeds in σ_k ,
 - iii. for each $\bigcirc\chi \in C_{\bigcirc\psi}$ and $B \rightarrow \bigcirc\chi \in C_R$,
either $T \vdash +\partial B$ or $T \vdash +\partial \bigcirc B$ succeed at σ_k ,
 - iv. for each $B \Rightarrow \chi \in C_R$ [$B \rightsquigarrow \chi \in C_R$],
 - A. $T \vdash +\partial B$ succeeds at σ_k , and
 - B. $B \Rightarrow \chi \not\leq A \Rightarrow \bigcirc\psi$ [$B \rightsquigarrow \chi \not\leq A \Rightarrow \bigcirc\psi$],
and
 - v. for each $B \Rightarrow \bigcirc\chi \in C_R$ [$B \rightsquigarrow \bigcirc\chi \in C_R$],
 - A. either $T \vdash +\partial B$ or $T \vdash +\partial \bigcirc B$ succeed at σ_k ,
and
 - B. $B \Rightarrow \bigcirc\chi \not\leq A \Rightarrow \bigcirc\psi$ [$B \rightsquigarrow \bigcirc\chi \not\leq A \Rightarrow \bigcirc\psi$].

Definition 34 (DSD-Proof)

A *DSD-proof* in a defeasible theory $T = \langle F, R, C, \preceq \rangle$ is a sequence of defeasible assertions such that for each $k \leq \text{length}(\sigma)$, one of $[M^+]$, $[DM^+]$, $[DM^-]$, $[E^+]$, $[SS_D^+]$, $[DSS^+]$, $[DSD^+]$, $[DDD^+]$ or $[DSD^-]$ holds.

Example 5 – Chisholm Example in EVALPSN

We consider a problem consisting of one fact, one obligation and two defeasible rules with no superiority relation.

- F4** In fact, Jones does not visit his mother.
O Jones ought to visit his mother.
R8 If Jones visits his mother, he ought to call her and tell her he is coming.
R9 It ought to be that if Jones does not visit his mother, he does not call her and tell her he is coming.

If we formalize the above problem as a defeasible deontic theory, it includes

- F4** $\neg v$,
O $\bigcirc v$,
R8 $v \Rightarrow \bigcirc c$,
R9 $\neg v \Rightarrow \bigcirc \neg c$,

where the letters v and c denote ‘visiting his mother’ and ‘calling his mother’. Apparently, the defeasible theory includes the conflicting defeasible rules **R8** and **R9** whose consequents constitute a conflict set $\{\{\bigcirc c, \bigcirc \neg c\}\}$. With respect to the conflict set, defeasible deontic reasoning is carried out. Intuitively, we obtain the conclusion $T \vdash +\partial \bigcirc \neg c$ by $[DSD^+]$ and $T \vdash -\partial \bigcirc c$ by $[DDD^-]$, because the application of $[DDD^+]$ to the obligation **O** and the defeasible rule **R8** is prevented since the factual detachment $[DSD^+]$ takes precedence over the deontic detachment $[DDD^+]$.

Now we show how to translate the defeasible theory into EVALPSN. It should be considered that the negation \neg followed by a literal p in the defeasible theory is translated to the epistemic negation \neg_1 in EVALPSN and the negation \neg followed by the deontic operator \bigcirc is translated to the epistemic negation \neg_2 . Obviously, the fact **F4** and the obligation **O** are translated to the EVALPSN clauses,

$$v: [(0, 2), \alpha] \quad \text{and} \quad v: [(2, 0), \beta], \quad (8)$$

respectively. In order to translate the defeasible rule **R8** to EVALPSN clauses, we take the proof conditions $[DSD^+]$ (Defeasible Factual Detachment) and

[DDD⁺](Defeasible Deontic Detachment) to carry out defeasible deontic reasoning. We consider two cases **e1** and **e2** in which the consequent of the rule **R8** can be defeasibly derivable.

Case e1 (Derivation by [DSD⁺]):

Suppose that the consequent $\bigcirc c$ of the defeasible rule **R8** is defeasibly derived by [DSD⁺]. The consequent $\bigcirc\neg c$ of the competing defeasible rule **R9** should not be defeasibly derived by [DSD⁺]. We take only this situation since [DSD⁺] is precedent to [DDD⁺]. Therefore, we need the following conditions to derive $\bigcirc c$ defeasibly :

- the antecedent v of the defeasible rule **R8** is strictly or defeasibly derivable,
- the antecedent $\neg v$ of the competing defeasible rule **R9** is not defeasibly derivable, and
- the consequent $\bigcirc\neg c$ of the competing defeasible rule **R9** is not strictly derivable.

Case e2 (Derivation by [DSD⁺]):

Suppose that the consequent $\bigcirc c$ of the defeasible rule **R8** is defeasibly derived by [DDD⁺]. The consequent $\bigcirc\neg c$ of the competing defeasible rule **R9** should be defeasibly derived by neither [DSD⁺] nor [DDD⁺]. Therefore, we need the following conditions to derive $\bigcirc c$ defeasibly :

- the obligatory antecedent $\bigcirc v$ of the defeasible rule **R8** is defeasibly derivable,
- neither the antecedent $\neg v$ of the competing defeasible rule **R9** nor its obligation $\bigcirc\neg v$ is defeasibly provable, and
- the consequent $\bigcirc\neg c$ of the competing defeasible rule **R9** is not strictly provable.

If we formalize the cases **e1** and **e2** in EVALPSN, the following EVALPSN clauses are obtained as the translation of the defeasible rule **R8** in the defeasible theory T .

$$c: [(1, 0), \beta] \leftarrow v: [(1, 0), \alpha] \wedge \sim v: [(0, 1), \alpha] \wedge \sim c: [(0, 2), \beta], \quad (9)$$

$$c: [(1, 0), \beta] \leftarrow v: [(1, 0), \beta] \wedge \sim v: [(0, 1), \alpha] \wedge \sim v: [(0, 1), \beta] \\ \wedge \sim c: [(0, 2), \beta]. \quad (10)$$

The defeasible rule **R9** is translated to the following EVALPSN clauses as well as the defeasible rule **R8**,

$$c: [(0, 1), \beta] \leftarrow v: [(0, 1), \alpha] \wedge \sim v: [(1, 0), \alpha] \wedge \sim c: [(2, 0), \beta], \quad (11)$$

$$c: [(0, 1), \beta] \leftarrow v: [(0, 1), \beta] \wedge \sim v: [(1, 0), \alpha] \wedge \sim v: [(1, 0), \beta] \\ \wedge \sim c: [(2, 0), \beta]. \quad (12)$$

Then, the EVALPSN $P = \{(8), (9), (10), (11), (12)\}$ has only one stable model

$$M_4 = \{v: [(2, 0), \beta], v: [(0, 2), \alpha], c: [(0, 1), \beta]\},$$

which shows $T \vdash +\Delta \circ v$, $T \vdash +\Delta -v$, $T \vdash +\partial \circ \neg c$ and $T \vdash -\partial \circ c$ since $M_4 \models \sim c: [(1, 0), \beta]$.

Generally, if an EVALPSN contains the strong negation \sim , it has stable model semantics just as with VALPSN. However, the stable model semantics may have the problem that some EVALPSNs may have more than two stable models, whereas others have *no* stable model. Moreover, the stable model computation takes a long time compared to usual logic programming such as PROLOG. Therefore, it does not seem to be so appropriate for practical application such as real time processing. However, we fortunately have cases to implement EVALPSN practically, if an EVALPSN is a *stratified* program, it has a tractable model called a *perfect model* [47] and strong negation in EVALPSN can be treated as *Negation as Failure* in normal logic programming. The details of stratified program and some tractable models for normal logic programs and databases can be found in [1, 11, 47, 49]. We define stratified EVALPSN by modifying the definition of stratified logic program [1] and introduce stratification of EVALPSN that might be used in EVALPSN safety verification.

Definition 35 (Stratified EVALPSN)

An EVALPSN P is called *stratified* if there is a partition,

$$P = P_1 \cup \dots \cup P_n$$

such that the following two conditions hold for $i = 1, \dots, n$:

1. if a positive weva-literal occurs in an EVALPSN clause in P_i , then its definition is contained within $\bigcup_{j < i} P_j$;
2. if a strongly negated weva-literal in an EVALPSN clause in P_i , then its definition is contained within $\bigcup_{j < i} P_j$, and P_1 can be empty.

Then it is said that P is *stratified* by $P_1 \cup \dots \cup P_n$ and each P_i is called a *stratum* of P .

In the above definition of EVALPSN stratification, for weva-literals $q : [\mu_1, \mu_2]$ and $q : [\lambda_1, \lambda_2]$ with the same predicate q , if $[\mu_1, \mu_2] \not\leq_e [\lambda_1, \lambda_2]$ and $[\lambda_1, \lambda_2] \not\leq_e [\mu_1, \mu_2]$, they can be regarded as different weva-literals. We show an example of EVALPSN stratification.

Example 6

Suppose an EVALPSN

$$P = \{ p:[(1, 0), \alpha], \quad (13)$$

$$q:[(0, 2), \beta] \leftarrow \sim p:[(1, 0), \alpha], \quad (14)$$

$$q:[(0, 2), \gamma] \leftarrow \sim q:[(0, 2), \beta] \}, \quad (15)$$

which shows a stereotype of safety verification EVALPSN that will appear in the following Sections. The EVALPSN P can be intuitively interpreted as follows: the fact $p:[(1, 0), \alpha]$ (13) has been obtained; there are two rules; if there is not the fact $p:[(1, 0), \alpha]$ as sensor information, then the forbiddance $q:[(0, 2), \beta]$ is derived (Eqn. (14)), and if there is not forbiddance $q:[(0, 2), \beta]$, then the permission $q:[(0, 2), \gamma]$ is derived (Eqn.(15)). The EVALPSN P has a stratification:

$$P_1 = \{ p:[(1, 0), \alpha] \},$$

$$P_2 = \{ q:[(0, 2), \beta] \leftarrow \sim p:[(1, 0), \alpha] \},$$

$$P_3 = \{ q:[(0, 2), \gamma] \leftarrow \sim q:[(0, 2), \beta] \}$$

since $[(0, 2), \beta] \not\prec_e [(0, 2), \gamma]$ and $[(0, 2), \gamma] \not\prec_e [(0, 2), \beta]$. Then, the forbiddance

$$q:[(0, 2), \beta] \quad (16)$$

is not derived since there is the fact (Eqn.(13)). However, the permission $q:[(0, 2), \gamma]$ is derived since there is not the forbiddance (Eqn. (16)).

Therefore, inefficient EVALPSN stable model computation does not have to be taken into account in practice since all EVALPSNs that will appear in the following Sections are stratified.

3 EVALPSN Safety Verification for Control

Safety verification is a crucial issue for all control systems. We have applied EVALPSN to various control and safety verifications, and provided some simulation systems for them [28, 29, 32–34, 37]. In this Section, we introduce an application of EVALPSN to the safety verification of a brewery pipeline valve control.

3.1 Outline of EVALPSN Safety Verification

First of all, we briefly introduce the basic idea of EVALPSN-based safety verification. Usually, control systems have their own safety properties to be secured, which can be described in deontic expression and easily translated into EVALPSN. The general flow of EVALPSN safety verification is shown in Fig. 3 and the verification process is carried out according to the following three steps:

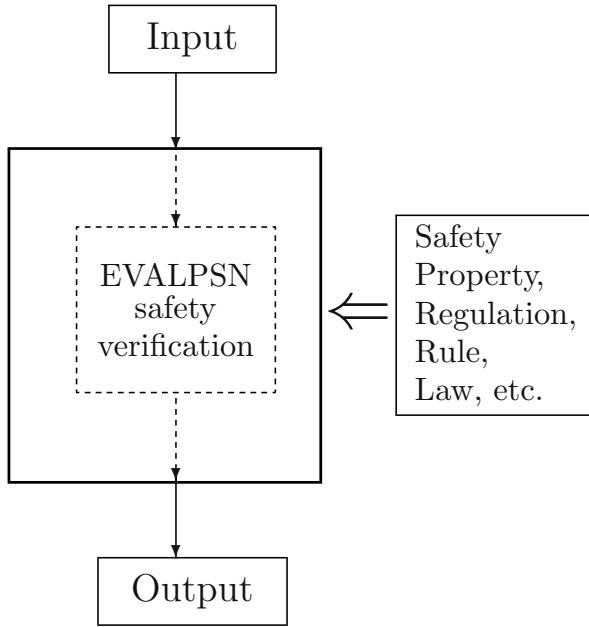


Fig. 3. The EVALPSN safety verification system

1. translate the safety control property into EVALPSN previously and construct the EVALPSN safety verification system;
2. various kinds of information for control – such as sensor information and the current physical state information of the system – are input to the EVALPSN safety verification system;
3. the EVALPSN safety verification system verifies the control system safety by EVALPSN defeasible deontic reasoning; if its safety is verified, we obtain permission for the verified process; otherwise we obtain forbiddance from processing.

3.2 EVALPSN Safety Verification for Pipeline Control

We will introduce an application of EVALPSN safety verification for pipeline valve control in a brewery plant by way of a simple example.

The basic idea of EVALPSN safety verification as a formal safety verification method originally stems from the safety verification method for railway interlocking by [19]. Morley's method has been proposed only for verifying railway interlocking safety, and it seems to be logically incomplete as it is implemented in the higher-order logical language HOL [12]. On the other hand, our method is logically complete and easy to implement since it is based on defeasible deontic reasoning in EVALPSN.

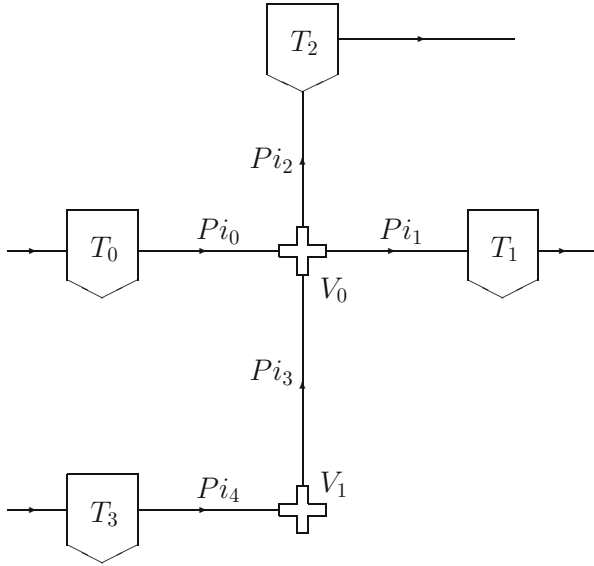


Fig. 4. Pipeline example

Brewery Pipeline Network

The pipeline network described in Fig. 4 is taken as a simple example for EVALPSN safety verification of brewery pipeline valve control. In Fig. 4, liquid flows are denoted by arrows, tanks by home-plate pentagons, and valves by cross figures.

In the pipeline network, physical and logical entities are considered, and the following items are defined as physical entities:

- four tanks, $T = \{T_0, T_1, T_2, T_3\}$;
- five pipes, $Pi = \{Pi_0, Pi_1, Pi_2, Pi_3, Pi_4\}$;
(a pipe includes neither valves nor tanks)
- two valves, $Val = \{V_0, V_1\}$.

Moreover, the following logical entities:

- four processes, $Pr = \{Pr_0, Pr_1, Pr_2, Pr_3\}$;
(a process is defined as a sequence of sub-processes and valves)
- five sub-processes,
 $SPr = \{SPr_0, SPr_1, SPr_2, SPr_3, SPr_4\}$

are defined. For example, process Pr_0 consists of the sequence (SPr_0, V_0, SPr_1) . Each entity is supposed to have logical, physical or both states. Sub-processes have two states *locked*(1) and *free*(f). Thus, if the sub-process is logically reserved with one sort of liquid, such a state is described as “the sub-process is locked by the liquid” and ‘free’ is defined as a state not being locked. Valves in the pipeline network are assumed to be able to control two

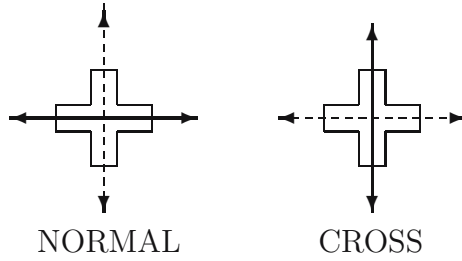


Fig. 5. Normal and cross directions

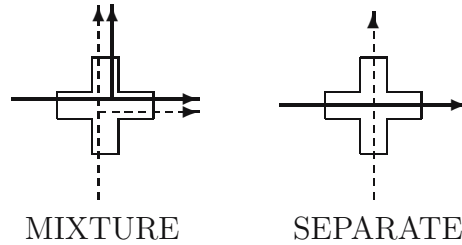


Fig. 6. Controlled mix and separate

liquid flows in the normal and cross directions, as shown in Fig. 5. Then, valves have two controlled states, *controlled mix*(cm) representing that it is controlled to mix two liquid flows in the normal and cross directions, and *controlled separate*(cs) representing that it is controlled to separate two liquid flows in the normal and cross directions as shown in Fig. 6. Processes have two states *set* (s) and *unset* (xs), then “the process is set” is defined as a logical state in which all the sub-processes in the process are locked, and ‘unset’ is as not set.

In the pipeline network the following four processes $Pr_{0,1,2,3}$ dealing with beer(b) and five sorts of pipe cleaning liquid, cold water(cw), warm water(ww), hot water(hw), nitric acid(na) and caustic soda(cs), are processed:

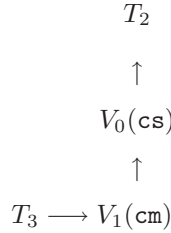
- Process Pr_0 : a beer process,

$$T_0 \longrightarrow V_0(cs) \longrightarrow T_1$$

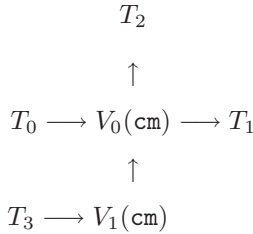
- Process Pr_1 : a cleaning process by nitric acid,

$$\begin{array}{c}
 T_2 \\
 \uparrow \\
 V_0(cs) \\
 \uparrow \\
 T_3 \longrightarrow V_1(cm)
 \end{array}$$

- Process Pr_2 : a cleaning process by cold water,



- Process Pr_3 : a brewery process with mixing.



In order to verify the safety for the processes $Pr_{0,1,2,3}$, the pipeline controller/operator issues a process request having an **if...then** form before starting each process. The **if**-part of the request describes the current state of the entities that should be used at the process, and the **then**-part describes the permission for setting the process. For example, a process request for the process Pr_1 can be described as:

if the sub-process SPr_0 is free,
 the sub-process SPr_1 is free,
 the valve V_0 is controlled separate,
then the process Pr_0 can be set ?

We consider the following process schedule for the four processes $Pr_{0,1,2,3}$.³

- PRS-0 the process Pr_0 starts before any other processes;
- PRS-1 the process Pr_1 starts immediately after the process Pr_0 ;
- PRS-2 the process Pr_2 starts immediately after the process Pr_1 ;
- PRS-3 the process Pr_3 starts immediately after both the processes Pr_0 and Pr_2 ,

which are shown as the process schedule chart in Fig. 7.

³ we shortly denote $A_{0,1,2,\dots,k}$ instead of A_0, A_1, A_2, \dots and A_k for any physical or logical entities A .

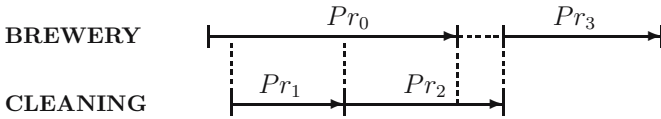


Fig. 7. Process schedule chart

Pipeline Safety Property

We assume three safety properties, **SPr** for sub-processes, **Val** for valves, and **Pr** for processes, to assure the pipeline valve safety – that is to say, to avoid unexpected mix of different sorts of liquid while processing in the pipeline network.

- SPr** is a forbidden case where the sub-process over a given pipe is simultaneously locked with different sorts of liquid;
- Val** is a forbidden case where valves are controlled for an unexpected mix of liquid;
- Pr** whenever a process is set, all its component sub-processes are locked and all its component valves are consistently controlled.

The safety of pipeline processes is assured by verifying whether process requests by operators contradict the safety properties or not in EVALPSN programming. Then, we have the following three steps as the EVALPSN safety verification process:

1. the safety properties **SPr**, **Val** and **Pr** for the pipeline processes $Pr_{0,1,2,3}$ are translated to EVALPSN clauses, which should be stored as an EVALPSN P_{sc} ;
2. the **if**-part and **then**-part of a process request are translated to EVALP clauses as EVALPs P_i and P_t , respectively;
3. the EVALP P_t is inquired from the EVALPSN $P_{sc} \cup P_i$, then if yes is returned, the safety for the process request is assured; otherwise, it is not assured.

Now, we introduce some literals used in the EVALPSN safety verification system.

- $Pr(i, l)$ denotes that the process i for the liquid l is set(**s**) or unset(**xs**), where $i \in \{p0, p1, p2, p3\}$ is a process id corresponding to one of the processes $Pr_{0,1,2,3}$, $l \in \{b, cw, ww, hw, na, cs\}$ indicates the sort of liquid used in the pipeline. We have an EVALP clause

$$Pr(i, l) : [\mu_1, \mu_2],$$

where

$$\begin{aligned} \mu_1 &\in \{\mathbf{s}, \mathbf{xs}\} & \mathcal{T}_v(1)_1 &= \{\perp_1, \mathbf{s}, \mathbf{xs}, \top_1\}, \\ \mu_2 &\in \{\alpha, \beta, \gamma\}. \end{aligned}$$

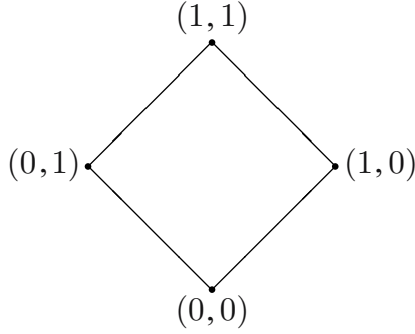


Fig. 8. The complete lattice $\mathcal{T}_v(1)$

The complete lattice $\mathcal{T}_v(1)_1$ is a variant interpretation of the complete lattice $\mathcal{T}_v(1)$ in Fig. 8.

Therefore, the annotations $\perp_1, \mathbf{s}, \mathbf{xs}$ and \top_1 stand for the vector annotations $(0, 0), (1, 0), (0, 1)$ and $(1, 1)$, respectively. The epistemic negation \neg_1 over $\mathcal{T}_v(1)_1$ is defined as the following mapping:

$$\begin{aligned} \neg_1([\perp_1, \mu_2]) &= [\perp_1, \mu_2], & \neg_1([\mathbf{s}, \mu_2]) &= [\mathbf{xs}, \mu_2], \\ \neg_1([\top_1, \mu_2]) &= [\top_1, \mu_2], & \neg_1([\mathbf{xs}, \mu_2]) &= [\mathbf{s}, \mu_2]. \end{aligned}$$

For example, an EVALP clause $Pr(p2, b) : [\mathbf{s}, \beta]$ can be intuitively interpreted as “it is obligatory that the beer process Pr_2 is set”.

- $SPr(i, j, l)$ denotes that the sub-process from valve i (or tank i) to valve j (or tank j) occupied by liquid l is locked(1) or free(\mathbf{f}). We assume that if the sub-process is free and the sort of liquid in the pipe is ‘don’t care’, then the liquid is logically represented by the symbol ‘0’(zero). Therefore,

$$l \in \{b, cw, ww, hw, na, cs, 0\},$$

and

$$i, j \in \{v0, v1, t0, t1, t2, t3\},$$

which are valve or tank IDs corresponding to the valves V_0 and V_1 , and the tanks $T_{0,1,2,3}$. Then, we have an EVALP clause,

$$SPr(i, j, l) : [\mu_1, \mu_2],$$

where

$$\begin{aligned} \mu_1 &\in \{1, \mathbf{f}, \} & \mathcal{T}_v(1)_2 &= \{\perp_2, 1, \mathbf{f}, \top_2\}, \\ \mu_2 &\in \{\alpha, \beta, \gamma\}. \end{aligned}$$

The complete lattice $\mathcal{T}_v(1)_2$ is a variant interpretation of the complete lattice $\mathcal{T}_v(1)$ in Fig. 8. Therefore, the annotations $\perp_2, 1, \mathbf{f}$ and \top_2 stand for the vector annotations $(0, 0), (1, 0), (0, 1)$ and $(1, 1)$, respectively.

The epistemic negation \neg_1 over $\mathcal{T}_v(1)_2$ is defined as the following mapping:

$$\begin{aligned}\neg_1([\perp_2, \mu_2]) &= [\perp_2, \mu_2], & \neg_1([1, \mu_2]) &= [\mathbf{f}, \mu_2], \\ \neg_1([\top_2, \mu_2]) &= [\top_2, \mu_2], & \neg_1([\mathbf{f}, \mu_2]) &= [1, \mu_2].\end{aligned}$$

For example, an EVALP clause $SPr(v0, t1, b) : [\mathbf{f}, \gamma]$ can be intuitively interpreted as “the sub-process from valve $v0$ to tank $t1$ is permitted to be locked by beer b ”.

- $Val(i, l_n, l_c)$ denotes that valve i occupied by two sorts of liquid $l_n, l_c \in \{b, cw, ww, hw, na, cs, 0\}$ is controlled separate(**cs**) or mix(**cm**), where $i \in \{v0, v1\}$. The arguments l_n and l_c denote the two liquids flowing in the normal and cross directions in the valve, respectively. Generally, if a valve is released from its controlled state such as a controlled mix, the liquid flow in the valve is represented by the symbol 0 that means the sort of the liquid is ‘don’t care’. We have an EVALP clause,

$$Val(i, l_n, l_c) : [\mu_1, \mu_2],$$

where

$$\begin{aligned}\mu_1 &\in \{\mathbf{cm}, \mathbf{cs}\} & \mathcal{T}_v(1)_3 &= \{\perp_3, \mathbf{cm}, \mathbf{cs}, \top_3\}, \\ \mu_2 &\in \{\alpha, \beta, \gamma\}.\end{aligned}$$

The complete lattice $\mathcal{T}_v(1)_3$ is a variant interpretation of the complete lattice $\mathcal{T}_v(1)$ in Fig. 8. Therefore, the annotations $\perp_3, \mathbf{cm}, \mathbf{cs}$ and \top_3 stand for the vector annotations $(0, 0), (1, 0), (0, 1)$ and $(1, 1)$, respectively. Epistemic negation \neg_1 over $\mathcal{T}_v(1)_3$ is defined as the following mapping:

$$\begin{aligned}\neg_1([\perp_3, \mu_2]) &= [\perp_3, \mu_2], & \neg_1([\mathbf{cs}, \mu_2]) &= [\mathbf{cm}, \mu_2], \\ \neg_1([\top_3, \mu_2]) &= [\top_3, \mu_2], & \neg_1([\mathbf{cm}, \mu_2]) &= [\mathbf{cs}, \mu_2].\end{aligned}$$

We assume that if a process finishes, all the valves included in the process are controlled separate(closed). For example, an EVALP clause, $Val(v0, 0, 0) : [\mathbf{cs}, \alpha]$ can be intuitively interpreted as “valve $v0$ has been released from the state controlled separate”; an EVALP clause, $Val(v0, b, cw) : [\mathbf{cs}, \beta]$ can be intuitively interpreted as both “it is forbidden for valve $v0$ to be controlled mix with beer b being in the normal direction and cold water cw in the cross direction”, and “it is obligatory for valve $v0$ to be controlled separate with beer b being in the normal direction and cold water cw in the cross direction;”, and an EVALP clause $Val(v0, 0, b) : [\mathbf{cs}, \alpha]$ can be intuitively interpreted as “it is a fact that valve $v0$ is controlled separate with the free flow 0 in the normal direction and beer b in the cross direction.

- $Eq_l(l_1, l_2)$ denotes that the sorts of liquids l_1 and l_2 are the same(**sa**) or different(**di**), where $l_1, l_2 \in \{b, cw, ww, hw, na, cs, 0\}$. We have an EVALP clause

$$Eq_l(l_1, l_2):[\mu_1, \mu_2],$$

where

$$\begin{aligned} \mu_1 &\in \{\mathbf{sa}, \mathbf{di}\} & \mathcal{T}_v(1)_4 &= \{\perp_4, \mathbf{sa}, \mathbf{di}, \top_4\}, \\ \mu_2 &\in \{\alpha, \beta, \gamma\}. \end{aligned}$$

The complete lattice $\mathcal{T}_v(1)_4$ is a variant interpretation of the complete lattice $\mathcal{T}_v(1)$ in Fig. 8. Therefore, the annotations \perp_4 , **sa**, **di** and \top_4 stand for the vector annotations (0, 0), (1, 0), (0, 1) and (1, 1), respectively. The epistemic negation \neg_1 over $\mathcal{T}_v(1)_4$ is defined as the following mapping:

$$\begin{aligned} \neg_1([\perp_4, \mu_2]) &= [\perp_4, \mu_2], & \neg_1([\mathbf{di}, \mu_2]) &= [\mathbf{sa}, \mu_2], \\ \neg_1([\top_4, \mu_2]) &= [\top_4, \mu_2], & \neg_1([\mathbf{sa}, \mu_2]) &= [\mathbf{di}, \mu_2]. \end{aligned}$$

If the verified process has finished safely, the process should be released from the set state and all the sub-processes and valves in the process should be free. Considering to represent the process release conditions in EVALPSN, we need to define some more predicates. We assume that if the terminal tank T_i of a process Pr_j has been filled with liquid, the finish signal $Fin(pj)$ of the process Pr_j is issued.

- $Tan(ti, l)$ denotes that tank ti has been filled fully(**fu**) with liquid l or empty(**em**). Then, we have an EVALP clause,

$$Tan(ti, l):[\mu_1, \mu_2],$$

where $ti \in \{t0, t1, t2, t3\}$ $l \in \{b, cw, ww, hw, na, cs, 0\}$,

$$\begin{aligned} \mu_1 &\in \{\mathbf{fu}, \mathbf{em}\} & \mathcal{T}_v(1)_5 &= \{\perp_5, \mathbf{fu}, \mathbf{em}, \top_5\}, \\ \mu_2 &\in \{\alpha, \beta, \gamma\}. \end{aligned}$$

The complete lattice $\mathcal{T}_v(1)_5$ is a variant interpretation of the complete lattice $\mathcal{T}_v(1)$ in Fig. 8. Therefore, the annotations \perp_5 , **fu**, **em** and \top_5 stand for the vector annotations (0, 0), (1, 0), (0, 1) and (1, 1), respectively. Epistemic negation \neg_1 over $\mathcal{T}_v(1)_5$ is defined as the following mapping:

$$\begin{aligned} \neg_1([\perp_5, \mu_2]) &= [\perp_5, \mu_2], & \neg_1([\mathbf{fu}, \mu_2]) &= [\mathbf{em}, \mu_2], \\ \neg_1([\top_5, \mu_2]) &= [\top_5, \mu_2], & \neg_1([\mathbf{em}, \mu_2]) &= [\mathbf{fu}, \mu_2]. \end{aligned}$$

Note that the annotation \perp_5 can be intuitively interpreted to denote that “the tank is filled with some amount of liquid but not fully”, that is to say, “no information in terms of fullness”. For example, an EVALP clause $Tan(t2, 0):[\mathbf{em}, \alpha]$ can be interpreted as “it is a fact that tank $t2$ is empty”.

- $Str(pi)$ denotes that the start signal of the process Pr_i has been issued (**is**) or not (**ni**).
- $Fin(pj)$ denotes that the finish signal of the process Pr_j has been issued(**is**) or not (**ni**). Then, we have EVALP clauses

$$Str(pi):[\mu_1, \mu_2], \quad Fin(pi):[\mu_1, \mu_2],$$

where $pi, pj \in \{p0, p1, p2, p3\}$,

$$\begin{aligned} \mu_1 &\in \{\mathbf{ni}, \mathbf{is}\} & \mathcal{T}_v(1)_6 &= \{\perp_6, \mathbf{ni}, \mathbf{is}, \top_6\}, \\ \mu_2 &\in \{\alpha, \beta, \gamma\}. \end{aligned}$$

The complete lattice $\mathcal{T}_v(1)_6$ is a variant interpretation of the complete lattice $\mathcal{T}_v(1)$ in Fig. 8. Therefore, the annotations $\perp_6, \mathbf{is}, \mathbf{ni}$ and \top_6 stand for the vector annotations $(0, 0), (1, 0), (0, 1)$ and $(1, 1)$, respectively. The epistemic negation \neg_1 over $\mathcal{T}_v(1)_6$ is defined as the following mapping:

$$\begin{aligned} \neg_1([\perp_6, \mu_2]) &= [\perp_6, \mu_2], & \neg_1([\mathbf{is}, \mu_2]) &= [\mathbf{ni}, \mu_2], \\ \neg_1([\top_6, \mu_2]) &= [\top_6, \mu_2], & \neg_1([\mathbf{ni}, \mu_2]) &= [\mathbf{is}, \mu_2]. \end{aligned}$$

For example, an EVALP clause $Fin(p3):[\mathbf{ni}, \alpha]$ can be interpreted as “it is a fact that the finish signal for the process $p3$ has not been issued yet”.

Safety Property in EVALPSN

First of all, we introduce how to apply EVALPSN defeasible deontic reasoning to the safety verification by taking the sub-process SPr_0 on which beer is transferred from the tank T_0 to the valve V_0 via the pipe Pi_0 as a simple example. If the safety property for the sub-process SPr_0 is satisfied, the sub-process SPr_0 is allowed to be locked by beer. The safety property is described more concretely as:

if sub-process SPr_0 is not locked by any processes,

then Sub-process SPr_0 can be locked;

else sub-process SPr_0 should not be set.

The safety property can be translated to the following defeasible deontic rules **DR1** and **DR2**:

$$\begin{aligned} \mathbf{DR1} &\{ \neg SP_{r_{lock}}(t0, v0, l) \} \\ &\Rightarrow \neg \bigcirc \neg SP_{r_{lock}}(t0, v0, b), \\ \mathbf{DR2} &\{ \} \Rightarrow \bigcirc \neg SP_{r_{lock}}(t0, v0, b), \end{aligned}$$

and the superiority relation $\mathbf{DR2} \preceq \mathbf{DR1}$, where the liquid l is not beer(b). The antecedent of the rule $\mathbf{DR1}$ is a fact, which is defined as just sensed information and the antecedent of the rule $\mathbf{DR2}$ is empty and always satisfied. Therefore, we consider only \mathbf{DFD} as being defeasible deontic reasoning in this case. Moreover, as neither the consequent $\neg \bigcirc \neg SPr_{lock}(t0, v0, b)$ nor $\bigcirc \neg SPr_{lock}(t0, v0, b)$ appears as the consequent of any other strict rules, those consequents cannot be strictly provable. Thus, the defeasible deontic rules $\mathbf{DR1}$ and $\mathbf{DR2}$ with the superiority relation are simply translated to instances of EVALPSN clauses Eqn. (17) and Eqn. (18) in the next paragraph.

Now, we formalize all the safety properties \mathbf{SPr} , \mathbf{Val} and \mathbf{Pr} in EVALPSN.

\mathbf{SPr}

The property can be intuitively interpreted as derivation rules of forbiddance such as “if the sub-process from valve/tank i to valve/tank j is locked with one sort of liquid, it is forbidden that the sub-process is locked with different sorts of liquid simultaneously”. Thus, we have the following EVALPSN clauses,

$$SPr(i, j, l_2):[\mathbf{f}, \beta] \leftarrow SPr(i, j, l_1):[1, \alpha] \wedge \sim Eql(l_1, l_2):[\mathbf{sa}, \alpha], \quad (17)$$

where $l_1, l_2 \in \{b, cw, ww, hw, na, cs, 0\}$. Moreover, in order to derive permission for locking sub-processes, we need the following EVALPSN clauses,

$$SPr(i, j, l):[\mathbf{f}, \gamma] \leftarrow \sim SPr(i, j, l):[\mathbf{f}, \beta], \quad (18)$$

where $l \in \{b, cw, ww, hw, na, cs, 0\}$.

\mathbf{Val}

The safety property can be intuitively interpreted as derivation rules of the forbiddance from controlling valves. We have to consider two cases: one is for deriving the forbiddance from changing the control state of valves, and another one is for deriving the forbiddance from mixing different sorts of liquid without changing the control state of valves.

Case 1

If a valve is separately controlled, it is forbidden for it to be a controlled mix; conversely, if a valve is a controlled mixture, it is forbidden for the valve to be separately controlled. Thus, we have the following EVALPSN clauses:

$$\begin{aligned} Val(i, l_n, l_c):[\mathbf{cs}, \beta] &\leftarrow Val(i, l_n, l_c):[\mathbf{cs}, \alpha] \wedge \sim Eql(l_n, 0):[\mathbf{sa}, \alpha] \\ &\wedge \sim Eql(l_c, 0):[\mathbf{sa}, \alpha], \end{aligned} \quad (19)$$

$$\begin{aligned} Val(i, l_n, l_c):[\mathbf{cm}, \beta] &\leftarrow Val(i, l_n, l_c):[\mathbf{cm}, \alpha] \wedge \sim Eql(l_n, 0):[\mathbf{sa}, \alpha] \\ &\wedge \sim Eql(l_c, 0):[\mathbf{sa}, \alpha], \end{aligned} \quad (20)$$

where $l_n, l_c \in \{b, cw, ww, hw, na, cs, 0\}$.

Case 2

Next, we consider the other forbiddance derivation case in which different sorts of liquid are mixed even if the valve control state is not changed. We have the following EVALPSN clauses:

$$\begin{aligned} Val(i, l_{n_2}, l_{c_2}) : [\mathbf{cm}, \beta] \leftarrow & Val(i, l_{n_1}, l_{c_1}) : [\mathbf{cs}, \alpha] \wedge \sim Eql(l_{n_1}, l_{n_2}) : [\mathbf{sa}, \alpha] \\ & \wedge \sim Eql(l_{n_1}, 0) : [\mathbf{sa}, \alpha], \end{aligned} \quad (21)$$

$$\begin{aligned} Val(i, l_{n_2}, l_{c_2}) : [\mathbf{cm}, \beta] \leftarrow & Val(i, l_{n_1}, l_{c_1}) : [\mathbf{cs}, \alpha] \wedge \sim Eql(l_{c_1}, l_{c_2}) : [\mathbf{sa}, \alpha] \\ & \wedge \sim Eql(l_{c_1}, 0) : [\mathbf{sa}, \alpha], \end{aligned} \quad (22)$$

$$Val(i, l_{n_2}, l_{c_2}) : [\mathbf{cs}, \beta] \leftarrow Val(i, l_{n_1}, l_{c_1}) : [\mathbf{cm}, \alpha] \wedge \sim Eql(l_{n_1}, l_{n_2}) : [\mathbf{sa}, \alpha], \quad (23)$$

$$Val(i, l_{n_2}, l_{c_2}) : [\mathbf{cs}, \beta] \leftarrow Val(i, l_{n_1}, l_{c_1}) : [\mathbf{cm}, \alpha] \wedge \sim Eql(l_{c_1}, l_{c_2}) : [\mathbf{sa}, \alpha], \quad (24)$$

where $l_{n_1}, l_{c_1}, l_{n_2}, l_{c_2} \in \{b, cw, ww, hw, na, cs, 0\}$.

Note that the EVALPSN clause $\sim Eql(l_n, 0) : [\mathbf{sa}, \alpha]$ represents “there does not exist information such that the normal direction with the liquid l_n in the valve is free (not controlled either)”.

As well as sub-processes, in order to derive permission for controlling valves, we need the following EVALPSN clauses:

$$Val(i, l_n, l_c) : [\mathbf{cm}, \gamma] \leftarrow \sim Val(i, l_n, l_c) : [\mathbf{cm}, \beta], \quad (25)$$

$$Val(i, l_n, l_c) : [\mathbf{cs}, \gamma] \leftarrow \sim Val(i, l_n, l_c) : [\mathbf{cs}, \beta], \quad (26)$$

where $l_n, l_c \in \{b, cw, ww, hw, na, cs, 0\}$.

Pr

The property can be intuitively interpreted as derivation rules of the permission for setting processes and directly translated into EVALPSN clauses as a rule “if all the components of the process can be locked or controlled consistently, then the process can be set”. For example, if the beer process Pr_0 consists of the sub-process from tank T_0 to valve V_0 , then valve V_0 with separately controlled for beer in the normal direction, and sub-process from valve V_0 to tank T_1 , then we have the following EVALP clause for obtaining permission for setting process Pr_0 ,

$$\begin{aligned} \text{Process } Pr_0 : \\ Pr(p0, b) : [\mathbf{xs}, \gamma] \leftarrow & SPr(t0, v0, b) : [\mathbf{f}, \gamma] \wedge SPr(v0, t1, b) : [\mathbf{f}, \gamma] \wedge \\ & Val(v0, b, l) : [\mathbf{cm}, \gamma] \wedge Tan(t0, b) : [\mathbf{fu}, \alpha] \wedge \\ & Tan(t1, 0) : [\mathbf{em}, \alpha], \end{aligned} \quad (27)$$

where $l \in \{b, cw, ww, hw, na, cs, 0\}$.

We also have the following EVALP clauses for setting the other processes:

Process Pr_1 :

$$\begin{aligned}
 Pr(p1, na) : [\mathbf{xs}, \gamma] \leftarrow & SPr(t3, v1, na) : [\mathbf{f}, \gamma] \wedge SPr(v1, v0, na) : [\mathbf{f}, \gamma] \wedge \\
 & SPr(v0, t2, na) : [\mathbf{f}, \gamma] \wedge Val(v0, l, na) : [\mathbf{cm}, \gamma] \wedge \\
 & Val(v1, na, 0) : [\mathbf{cs}, \gamma] \wedge Tan(t3, na) : [\mathbf{fu}, \alpha] \wedge \\
 & Tan(t2, 0) : [\mathbf{em}, \alpha],
 \end{aligned} \tag{28}$$

Process Pr_2 :

$$\begin{aligned}
 Pr(p2, cw) : [\mathbf{xs}, \gamma] \leftarrow & SPr(t3, v1, cw) : [\mathbf{f}, \gamma] \wedge SPr(v1, v0, cw) : [\mathbf{f}, \gamma] \wedge \\
 & SPr(v0, t2, cw) : [\mathbf{f}, \gamma] \wedge Val(v0, l, cw) : [\mathbf{cm}, \gamma] \wedge \\
 & Val(v1, cw, 0) : [\mathbf{cs}, \gamma] \wedge Tan(t3, cw) : [\mathbf{fu}, \alpha] \wedge \\
 & Tan(t2, 0) : [\mathbf{em}, \alpha],
 \end{aligned} \tag{29}$$

Process Pr_3 :

$$\begin{aligned}
 Pr(p3, b) : [\mathbf{xs}, \gamma] \leftarrow & SPr(t0, v0, b) : [\mathbf{f}, \gamma] \wedge SPr(t3, v1, b) : [\mathbf{f}, \gamma] \wedge \\
 & SPr(v0, t1, b) : [\mathbf{f}, \gamma] \wedge SPr(v0, t2, b) : [\mathbf{f}, \gamma] \wedge \\
 & SPr(v1, v0, b) : [\mathbf{f}, \gamma] \wedge Val(v0, b, b) : [\mathbf{cs}, \gamma] \wedge \\
 & Val(v1, b, 0) : [\mathbf{cs}, \gamma] \wedge Tan(t0, b) : [\mathbf{fu}, \alpha] \wedge \\
 & Tan(t1, 0) : [\mathbf{em}, \alpha] \wedge Tan(t3, b) : [\mathbf{fu}, \alpha] \wedge \\
 & Tan(t2, 0) : [\mathbf{em}, \alpha],
 \end{aligned} \tag{30}$$

where $l \in \{b, cw, ww, hw, na, cs, 0\}$.

Therefore, we have the safety verification EVALPSN,

$$P_{sc} = \{(17), \dots, (30)\}. \tag{31}$$

Process Release Control

If a process has finished, we need to release all entities in the process from ‘locked’ states to ‘free’ states. Then, we need process release conditions to be verified at the end of each process in EVALPSN.

Example 7

Suppose that the process Pr_i that transfers liquid l in tank T_j to tank T_k has finished. Then, we may have the process release condition expressed by the following **if...then** rule:

- if** the liquid l in tank T_j has been transferred into tank T_k
in process Pr_i after process Pr_i starts,
and the finish signal $Fin(pi)$ for process Pr_i has been issued;
then process Pr_i itself is allowed to be unset,
and each logical entity of process Pr_i is also allowed to be free.

The release condition for the process Pr_i may be translated to the EVALP clauses,

$$Pr(pi, b):[s, \gamma] \leftarrow Str(pi):[is, \alpha] \wedge Tan(tj, b):[em, \alpha] \wedge \\ Tan(tk, b):[fu, \alpha] \wedge Fin(pi):[is, \alpha].$$

As well as **Example 7**, we can formalize the release conditions for the processes $Pr_{0,1,2,3}$ in EVALPSN.

Process Pr_0 :

$$Pr(p0, b):[s, \gamma] \leftarrow Str(p0):[is, \alpha] \wedge Tan(t0, b):[em, \alpha] \wedge \\ Tan(t1, b):[fu, \alpha] \wedge Fin(p0):[is, \alpha], \quad (32)$$

$$SPR(t0, v0, 0):[1, \gamma] \leftarrow Pr(p0, b):[s, \gamma], \quad (33)$$

$$SPR(v0, t1, 0):[1, \gamma] \leftarrow Pr(p0, b):[s, \gamma], \quad (34)$$

$$Val(v0, 0, l):[cm, \gamma] \leftarrow Pr(p0, b):[s, \gamma], \quad (35)$$

Process Pr_1 :

$$Pr(p1, na):[s, \gamma] \leftarrow Str(p1):[is, \alpha] \wedge Tan(t3, na):[em, \alpha] \wedge \\ Tan(t2, na):[fu, \alpha] \wedge Fin(p1):[is, \alpha], \quad (36)$$

$$SPR(v0, t2, 0):[1, \gamma] \leftarrow Pr(p1, na):[s, \gamma], \quad (37)$$

$$SPR(v1, v0, 0):[1, \gamma] \leftarrow Pr(p1, na):[s, \gamma], \quad (38)$$

$$SPR(t3, v1, 0):[1, \gamma] \leftarrow Pr(p1, na):[s, \gamma], \quad (39)$$

$$Val(v0, l, 0):[cm, \gamma] \leftarrow Pr(p1, na):[s, \gamma], \quad (40)$$

$$Val(v1, 0, 0):[cm, \gamma] \leftarrow Pr(p1, na):[s, \gamma], \quad (41)$$

Process Pr_2 :

$$Pr(p2, cw):[s, \gamma] \leftarrow Str(p2):[is, \alpha] \wedge Tan(t3, cw):[em, \alpha] \wedge \\ Tan(t2, cw):[fu, \alpha] \wedge Fin(p2):[is, \alpha], \quad (42)$$

$$SPR(v0, t2, 0):[1, \gamma] \leftarrow Pr(p2, cw):[s, \gamma], \quad (43)$$

$$SPr(v1, v0, 0):[1, \gamma] \leftarrow Pr(p2, cw):[s, \gamma], \quad (44)$$

$$SPr(t3, v1, 0):[1, \gamma] \leftarrow Pr(p2, cw):[s, \gamma], \quad (45)$$

$$Val(v0, l, 0):[cm, \gamma] \leftarrow Pr(p2, cw):[s, \gamma], \quad (46)$$

$$Val(v1, 0, 0):[cm, \gamma] \leftarrow Pr(p2, cw):[s, \gamma], \quad (47)$$

Process Pr_3 :

$$\begin{aligned} Pr(p3, b):[s, \gamma] \leftarrow & Str(p3):[is, \alpha] \wedge Tan(t0, b):[em, \alpha] \wedge \\ & Tan(t3, b):[em, \alpha] \wedge Tan(t1, b):[fu, \alpha] \wedge \\ & Tan(t2, b):[fu, \alpha] \wedge Fin(p3):[is, \alpha], \end{aligned} \quad (48)$$

$$SPr(t0, v0, 0):[1, \gamma] \leftarrow Pr(p3, b):[s, \gamma], \quad (49)$$

$$SPr(v0, t1, 0):[1, \gamma] \leftarrow Pr(p3, b):[s, \gamma], \quad (50)$$

$$SPr(v0, t2, 0):[1, \gamma] \leftarrow Pr(p3, b):[s, \gamma], \quad (51)$$

$$SPr(v1, v0, 0):[1, \gamma] \leftarrow Pr(p3, b):[s, \gamma], \quad (52)$$

$$SPr(t3, v1, 0):[1, \gamma] \leftarrow Pr(p3, b):[s, \gamma], \quad (53)$$

$$Val(v0, l, 0):[cm, \gamma] \leftarrow Pr(p3, b):[s, \gamma], \quad (54)$$

$$Val(v1, 0, 0):[cm, \gamma] \leftarrow Pr(p3, b):[s, \gamma], \quad (55)$$

where $l = \{b, cw, ww, hw, na, cs, 0\}$.

Therefore, we have the EVALP clause,

$$P_{pr} = \{(32), \dots, (55)\} \quad (56)$$

as the process release safety control.

Example

Now we show an example of EVALPSN safety verification for setting and releasing processes $Pr_{0,1,2,3}$ according to the process schedule chart in Fig. 7.

Initial Stage All the sub-processes and valves are free (unlocked), and no process has already started at this stage. In order to verify the safety of all the processes $Pr_{0,1,2,3}$, the following fact EVALP clauses (detected information) are input to EVALPSN P_{sc} Eqn. (31):

$$\begin{aligned} P_{input}^1 = \{ & SPr(t0, v0, 0):[f, \alpha], & Val(v0, 0, 0):[cs, \alpha], \\ & SPr(v0, t1, 0):[f, \alpha], & Val(v1, 0, 0):[cs, \alpha], \\ & SPr(v0, t2, 0):[f, \alpha], \\ & SPr(v1, v0, 0):[f, \alpha], \\ & SPr(t3, v1, 0):[f, \alpha], \\ & Tan(t0, b):[fu, \alpha], & Tan(t1, 0):[em, \alpha], \\ & Tan(t2, 0):[em, \alpha], & Tan(t3, na):[fu, \alpha] \}. \end{aligned} \quad (57)$$

$$(58)$$

Then all the sub-processes and valves are permitted to be locked or controlled. However, the tank conditions Eqn. (57) and Eqn. (58) do not permit the processes Pr_2 and Pr_3 to be set. The beer process Pr_0 can be verified to be set as follows:

- we have neither forbiddance from locking the sub-processes SPr_0 and SPr_1 , nor forbiddance from controlling valve V_0 separate with beer in the normal direction, by EVALPSN clauses Eqns. (17), (20), (21) and (22) in EVALPSN P_{sc} and the input EVALP P_{input}^1 ;
- then, we have permission for locking sub-processes SPr_0 and SPr_1 , and controlling valve V_0 separately with beer in the normal direction and any sort of liquid in the cross direction,

$$\begin{aligned} SPr(t0, v0, b) &: [\mathbf{f}, \gamma], & Val(v0, b, l) &: [\mathbf{cm}, \gamma], \\ SPr(v0, t1, b) &: [\mathbf{f}, \gamma], \end{aligned}$$

where $l \in \{b, cw, ww, hw, na, cs, 0\}$, by the EVALPSN clauses Eqn. (18) and Eqn. (25);

- moreover, we have other tank conditions,

$$Tan(t0, b) : [\mathbf{fu}, \alpha], \quad Tan(t1, 0) : [\mathbf{em}, \alpha];$$

- thus, we have permission for process Pr_0 to be set,

$$Pr(p0, b) : [\mathbf{xs}, \gamma],$$

by EVALP clause Eqn. (27).

According to the process schedule in Fig. 7, if process Pr_0 has started first, then the next process Pr_1 has to have its safety verified at the next stage.

2nd Stage Beer process Pr_0 has already started but not yet finished at this stage. Then in order to verify the safety of the other three processes $Pr_{1,2,3}$, the following fact EVALP clauses are input to EVALPSN P_{sc} Eqn. (31):

$$\begin{aligned} P_{input}^2 = \{ & SPr(t0, v0, b) : [\mathbf{1}, \alpha], & Val(v0, b, 0) &: [\mathbf{cs}, \alpha], \\ & SPr(v0, t1, b) : [\mathbf{1}, \alpha], & Val(v1, 0, 0) &: [\mathbf{cs}, \alpha], \\ & SPr(v0, t2, 0) : [\mathbf{f}, \alpha], \\ & SPr(v1, v0, 0) : [\mathbf{f}, \alpha], \\ & SPr(t3, v1, 0) : [\mathbf{f}, \alpha], \\ & Tan(t2, 0) : [\mathbf{em}, \alpha], & & (59) \\ & Tan(t3, na) : [\mathbf{fu}, \alpha] \}. & & (60) \end{aligned}$$

Since tank conditions Eqns. (59) and (60) permit the setting of neither process Pr_2 nor process Pr_3 at this stage, only process Pr_1 is assured of its safety, as follows:

- we have neither forbiddance from locking the three sub-processes $SPr_{2,3,4}$, forbiddance from controlling the valves V_0 separate with any sort of liquid in the normal direction and nitric acid in the cross direction, nor forbiddance from controlling the valve V_1 mix(open) with nitric acid in the normal direction and no liquid in the cross direction, by the EVALPSN clauses Eqns. (17) – (24) and the input EVALP P_{input}^2 ;
- therefore, we have permission for locking the three sub-processes $SPr_{2,3,4}$, and controlling valves V_0 and V_1 as described previously,

$$\begin{aligned} SPr(v0, t2, na) : [f, \gamma], & \quad Val(v0, b, na) : [cm, \gamma], \\ SPr(v1, v0, na) : [f, \gamma], & \quad Val(v1, na, 0) : [cs, \gamma], \\ SPr(t3, v1, na) : [f, \gamma], & \end{aligned}$$

by EVALP clauses Eqns. (18), (25) and (26);

- moreover, we have the tank conditions,

$$Tan(t3, na) : [fu, \alpha], \quad Tan(t2, 0) : [em, \alpha] ;$$

- thus, it is permitted to set process Pr_1 ,

$$Pr(p1, na) : [xs, \gamma],$$

by EVALPSN clause Eqn. (28).

Both processes Pr_0 and Pr_1 have started, then the other processes Pr_2 and Pr_3 need to have their safety verified at the next stage.

3rd Stage In order to verify the safety of processes Pr_2 and Pr_3 , the following fact EVALP clauses are input to EVALPSN P_{sc} :

$$\begin{aligned} P_{input}^3 = \{ & SPr(t0, v0, b) : [1, \alpha], & Val(v0, b, na) : [cs, \alpha], \\ & SPr(v0, t1, b) : [1, \alpha], & Val(v1, na, 0) : [cm, \alpha], \\ & SPr(v0, t2, na) : [1, \alpha], \\ & SPr(v1, v0, na) : [1, \alpha], \\ & SPr(t3, v1, na) : [1, \alpha] \}. \end{aligned}$$

Apparently, neither process Pr_2 nor Pr_3 is permitted to be set, since there is no tank condition in the input EVALP P_{input}^3 to be satisfied. We show the failed safety verification for the process Pr_2 :

- we have forbiddance from locking the three sub-processes $SPr_{2,3,4}$, forbiddance from controlling valve V_0 separately with beer in the normal direction and cold water in the cross direction, as well as forbiddance from

controlling valve V_1 mix with cold water in the normal direction and no liquid in the cross direction,

$$\begin{aligned} SPr(v0, t2, cw) &: [\mathbf{f}, \beta], & Val(v0, b, cw) &: [\mathbf{cm}, \beta], \\ SPr(v1, v0, cw) &: [\mathbf{f}, \beta], & Val(v1, cw, 0) &: [\mathbf{cs}, \beta], \\ SPr(t3, v1, cw) &: [\mathbf{f}, \beta], \end{aligned}$$

by EVALPSN clauses Eqns. (17), (22), and (23), and the input EVALP P_{input}^3 .

The finish condition for the nitric acid process Pr_1 is that tank T_2 is fully filled with nitric acid and tank T_3 is empty. If the nitric acid process Pr_1 has finished and its finish conditions have been satisfied, process Pr_1 is permitted to be released (unset) by EVALP P_{pr} Eqn. (56). It is also supposed that tank T_3 is filled with cold water and that tank T_2 is empty, as preparation for the cold water process Pr_2 immediately after process Pr_1 finishes. Then, cold water process Pr_2 has to be verified and started according to the process schedule. We show the safety verification of process Pr_2 during the next stage.

4th Stage If the nitric acid process Pr_1 has finished and its finishing conditions have been satisfied, the three sub-processes $SPr_{2,3,4}$, valve V_1 , and the cross direction of valve V_0 are permitted to be released by EVALP clauses Eqns. (36) – (41). Since only Pr_0 is an ongoing process, the other three processes $Pr_{1,2,3}$ have to be verified. In order to do that, the following input EVALP P_{input}^4 is input to EVALPSN P_{sc} :

$$\begin{aligned} P_{input}^4 = \{ & SPr(t0, v0, b) : [1, \alpha], & Val(v0, b, 0) &: [\mathbf{cs}, \alpha], \\ & SPr(v0, t1, b) : [1, \alpha], & Val(v1, 0, 0) &: [\mathbf{cs}, \alpha], \\ & SPr(v0, t2, 0) : [\mathbf{f}, \alpha], \\ & SPr(v1, v0, 0) : [\mathbf{f}, \alpha], \\ & SPr(t3, v1, 0) : [\mathbf{f}, \alpha], \\ & Tan(t2, 0) : [\mathbf{em}, \alpha], & Tan(t3, cw) &: [\mathbf{fu}, \alpha] \}. \end{aligned}$$

Since process Pr_0 is still ongoing, neither Pr_1 nor Pr_3 are permitted to be set, and only process Pr_2 is permitted to be set – as well as process Pr_1 – during the 2nd stage. Therefore, we have permission for setting process Pr_2 ,

$$Pr(p2, cw) : [\mathbf{xs}, \gamma],$$

by EVALP clause Eqn. (29).

Now, both processes Pr_0 and Pr_2 are ongoing. Then, apparently any other processes are not permitted to be set. Moreover, even if one of the processes

Pr_0 and Pr_2 has finished, process Pr_3 is not permitted to be set until both Pr_0 and Pr_2 finish. We show the safety verification for the last process Pr_3 during the next stages.

5th Stage If neither process Pr_0 nor Pr_2 has finished yet, we have to verify the safety of processes Pr_1 and Pr_3 . The following input EVALP P_{input}^5 is input to EVALPSN P_{sc} Eqn. (31):

$$P_{input}^5 = \{ \begin{array}{ll} SPr(t0, v0, b) : [1, \alpha], & Val(v0, b, cw) : [cs, \alpha], \\ SPr(v0, t1, b) : [1, \alpha], & Val(v1, cw, 0) : [cm, \alpha], \\ SPr(v0, t2, cw) : [1, \alpha], & \\ SPr(v1, v0, cw) : [1, \alpha], & \\ SPr(t3, v1, cw) : [1, \alpha] \}. \end{array}$$

Then, since all sub-processes and valves are locked and controlled, neither processes Pr_1 nor Pr_3 is permitted to be set. It is shown that process Pr_3 is not permitted to be set as follows:

- we have forbiddance from locking the three sub-processes $SPr_{2,3,4}$ in process Pr_3 and controlling the two valves $V_{0,1}$,

$$\begin{array}{ll} SPr(v0, t2, b) : [f, \beta], & Val(v0, b, b) : [cs, \beta], \\ SPr(t3, v1, b) : [f, \beta], & Val(v1, b, 0) : [cs, \beta], \\ SPr(v1, v0, b) : [f, \beta], & \end{array}$$

by the EVALPSN clauses Eqn. (17) – (23) and the input EVALP P_{input}^5 ;

- therefore, we cannot have permission for setting process Pr_3 .

The finish conditions for process Pr_2 are that tank T_2 is fully filled with cold water and tank T_3 is empty. If process Pr_2 has finished and its finish conditions have been satisfied, then process Pr_2 is permitted to be released (unset) by EVALP P_{pr} Eqn. (56). It is also supposed that tank T_3 is filled with beer and that tank T_2 is empty as preparation for the beer process Pr_3 immediately after process Pr_2 finishes. Then, process Pr_3 has to be verified as safe and started according to the process schedule, but process Pr_3 cannot be permitted to be set. We show the safety verification for process Pr_3 during the next stage.

6th Stage If process Pr_2 has finished and its finish conditions have been satisfied, the three sub-processes $SPr_{2,3,4}$, valve V_1 , and the cross direction of valve V_0 are permitted to be released by EVALP clauses Eqns. (36) – (41) in the EVALP P_{pr} . Since only process Pr_0 is ongoing, the other three processes

$Pr_{1,2,3}$ need to have their safety verified. In order to do so, the following input EVALP P_{input}^6 is input to EVALPSN P_{sc} Eqn. (31):

$$P_{input}^6 = \{ \begin{array}{ll} SPr(t0, v0, b):[1, \alpha], & Val(v0, b, 0):[cs, \alpha], \\ SPr(v0, t1, b):[1, \alpha], & Val(v1, 0, 0):[cs, \alpha], \\ SPr(v0, t2, 0):[f, \alpha], & \\ SPr(v1, v0, 0):[f, \alpha], & \\ SPr(t3, v1, 0):[f, \alpha], & \\ Tan(t2, 0):[em, \alpha], & Tan(t3, b):[fu, \alpha] \end{array} \}.$$

Since process Pr_0 is still being processed, process Pr_3 does not have its safety verified due to the tank conditions and the safety property **Val** for valve V_0 . Safety verification for process Pr_3 is carried out as follows:

- we have the forbiddance from controlling the valve V_0 mix,

$$Val(v0, b, b):[cs, \beta],$$

by EVALPSN clause Eqn. (19);

- therefore, we cannot have permission for setting process Pr_3 then. On the other hand, even if process Pr_0 has finished while process Pr_2 is still being processed, process Pr_3 is not permitted to be set. If both processes Pr_0 and Pr_2 have finished, then process Pr_3 is assured its safety and set. Then, process Pr_3 starts according to the process schedule.

Safety verification of subsequent stages is executed in a similar manner to the preceding ones.

4 Before-after EVALPSN

We have introduced EVALPSN and its application to the safety verification for pipeline valve control. This was for verifying the safety of each individual process; however, we still need to consider another safety verification for process *order*. For example, recall the brewery pipeline processes of the previous Section. Dangerous liquids such as nitric acid and caustic soda are used for cleaning the pipelines in the same pipeline. If those liquids are processed continuously and mixed, explosion by neutralization could result. In order to avoid such a dangerous event, the safety of process order should be strictly verified. It is not so appropriate for EVALPSN safety verification to deal with process order (before-after (bf) relations between processes). In order to deal with process order in the same EVALPSN safety verification framework, we provide a new interpretation of vector annotations in EVALPSN.

4.1 Before-after Relation in EVALPSN

First of all, we introduce a special literal $R(pi, pj, t)$ whose vector annotation represents the before-after relation between processes $Pr_i(pi)$ and $Pr_j(pj)$ at the time t , and the literal $R(pi, pj, t)$ is called a *bf-literal*.⁴

Definition 36 (bf-EVALPSN)

An extended vector annotated literal $R(pi, pj, t) : [\mu_1, \mu_2]$ is called a *bf-EVALP* literal, where μ_1 is a vector annotation and $\mu_2 \in \{\alpha, \beta, \gamma\}$. If an EVALPSN clause contains bf-EVALP literals, it is called a *bf-EVALPSN clause* or just a *bf-EVALP clause* if it contains no strong negation. A bf-EVALPSN is a finite set of bf-EVALPSN clauses.

We provide some paraconsistent interpretations of vector annotations for representing bf-relations, which are called *bf-annotations*. Strictly speaking, bf-relations between processes are classified into 15 sorts according to bf-relations between start/finish times of two processes. Suppose that there are two processes, Pr_i with start time x_s and finish time x_f , and Pr_j with start time y_s and finish time y_f . Then 15 sorts of bf-annotations are defined.

Before (be)/After (af)

Firstly, we define basic bf-relations *before/after* between two processes according to the bf-relation between the start times of the two processes, which are represented by the bf-annotations **be/af**, respectively. If one process has started before/after another, then the bf-relations between those processes are defined as ‘before(**be**)/after(**af**)’, respectively. The bf-relations also are described in the process time chart Fig. 9 with the condition that process Pr_i has started before process Pr_j starts. The bf-relation between their start/finish times is denoted by the inequality $\{x_s < y_s\}$.⁵ For example, the fact that at time t “process Pr_i has started before process Pr_j starts” can be represented by the bf-EVALP clause,

$$R(pi, pj, t) : [\mathbf{be}, \alpha].$$

The bf-relations before/after do not care when the two processes finish.

Disjoint Before (db)/After (da)

The bf-relations *disjoint before/after* between processes Pr_i and Pr_j are represented by the bf-annotations **db/da**, respectively. The expressions ‘disjoint

⁴ Hereafter in this Chapter the phrase ‘before-after’ is abbreviated as simply ‘bf’.

⁵ If time t_1 is earlier than time t_2 , we conveniently denote the relation by the inequality $t_1 < t_2$ in this Section.

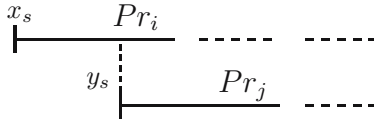


Fig. 9. Bf-relations, Before/After



Fig. 10. Bf-relations disjoint before/after

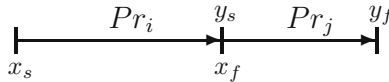


Fig. 11. Bf-relations, Immediate Before/After

before/after’ imply that there is a time lag between the earlier process finish time and the later process start time. They also are described in the process time chart Fig.10 with the condition that process Pr_i has finished before process Pr_j starts. The bf-relation between their start/finish times is denoted by the inequality $\{x_f < y_s\}$. For example, an obligation at time t that “process Pr_i must start after process Pr_j finishes” can be represented by the bf-EVALP clause,

$$R(pi, pj, t) : [da, \beta].$$

Immediate Before (mb)/After (ma)

The bf-relations *immediate before/after* between processes Pr_i and Pr_j are represented by the bf-annotations **mb/ma**, respectively. The expressions ‘immediate before/after’ imply that there is no time lag between the earlier process finish time and the later process start time. The bf-relations also are described in the process time chart Fig.11 with the condition that process Pr_i has finished immediately before process Pr_j starts. The bf-relation between their start/finish times is denoted by the equality $\{x_f = y_s\}$. For example, the fact that at time t “process Pr_i has finished immediately before process Pr_j starts” can be represented by the bf-EVALP clause,

$$R(pi, pj, t) : [mb, \alpha].$$

Joint Before (jb)/After (ja)

The bf-relations, *joint before/after* between processes Pr_i and Pr_j are represented by the bf-annotations **jb/ja**, respectively. The expressions ‘joint before/after’ imply that the two processes overlap and the earlier process has

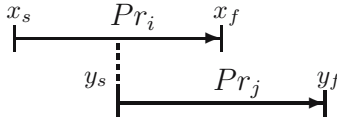


Fig. 12. Bf-relations joint before/after

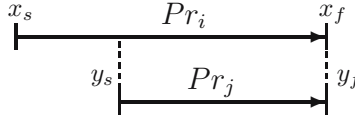


Fig. 13. Bf-relations s-included before/after

finished before the later one finishes. The bf-relations also are described in the process time chart Fig. 12 with the condition that process Pr_i has started before process Pr_j starts, and that process Pr_i has finished before process Pr_j finishes. The bf-relation between their start/finish times is denoted by the inequalities $\{x_s < y_s < x_f < y_f\}$. For example, the fact at time t “process Pr_i has started before process Pr_j starts, and finishes before process Pr_j finishes” can be represented by the bf-EVALP clause,

$$R(pi, pj, t): [jb, \alpha].$$

S-included Before (sb)/After (sa)

The bf-relations *s-included before/after* between processes Pr_i and Pr_j are represented by the bf-annotations **sb/sa**, respectively. The expressions ‘s-included before/after’ imply that one process has started before the other, and that they finish at the same time. The bf-relations also are described in the process time chart Fig. 13 with the condition that process Pr_i has started before process Pr_j starts, and they finish at the same time. The bf-relation between their start/finish times is denoted by the equality and inequalities $\{x_s < y_s < x_f = y_f\}$. For example, the fact that at time t “process Pr_i has started before process Pr_j starts, and they finish at the same time” can be represented by the bf-EVALP clause,

$$R(pi, pj, t): [sb, \alpha].$$

Included Before (ib)/After (ia)

The bf-relations *included before/after* between processes Pr_i and Pr_j are represented by the bf-annotations **ib/ia**, respectively. The expressions ‘included before/after’ imply that one process has started/finished before/after another one starts/finishes, respectively. The bf-relations also are described in the process time chart Fig. 14 with the condition that process Pr_i has started

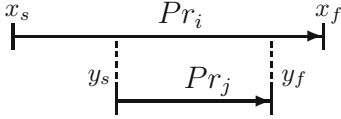


Fig. 14. Bf-relations included before/after

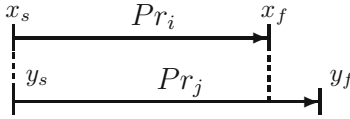


Fig. 15. Bf-relations f-included before/after

before process Pr_j starts, and finishes after process Pr_j finishes. The bf-relation between their start/finish times is denoted by the inequalities $\{x_s < y_s, y_f < x_f\}$. For example, an obligation at time t that “process Pr_i must start before process Pr_j starts, and finish after process Pr_j finishes” can be represented by the bf-EVALP clause,

$$R(pi, pj, t) : [ib, \beta].$$

F-included Before (fb)/After (fa)

The bf-relations *f-included before/after* between processes Pr_i and Pr_j are represented by the bf-annotations **fb/fa**, respectively. The expressions ‘f-included before/after’ imply that the two processes have started at the same time, and that one process finishes before the other. The bf-relations also are described in the process time chart Fig. 15 with the condition that processes Pr_i and Pr_j start at the same time, and that process Pr_i finishes after process Pr_j . The bf-relation between their start/finish times is denoted by the equality and inequality $\{x_s = y_s, y_f < x_f\}$. For example, the fact at time t “processes Pr_i and Pr_j start at the same time, and that process Pr_i finishes after process Pr_j ” can be represented by the bf-EVALP clause,

$$R(pi, pj, t) : [fa, \alpha].$$

Paraconsistent Before-after (pba)

The bf-relation *paraconsistent before-after* between processes Pr_i and Pr_j is represented by the bf-annotation **pba**. The expression ‘paraconsistent before-after’ implies that the two processes start at the same time and also finish at the same time. The bf-relation is also described in the process time chart Fig. 16 with the condition that processes Pr_i and Pr_j not only start at the

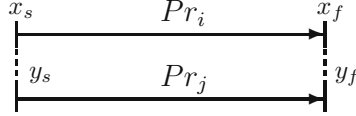


Fig. 16. Bf-relation paraconsistent before-after

same time but also finish at the same time. The bf-relation between their start/finish times is denoted by the equalities $\{x_s = y_s, y_f = x_f\}$. For example, an obligation at time t “processes Pr_i and Pr_j must not only start but also finish at the same time” can be represented by the bf-EVALP clause

$$R(pi, pj, t) : [pba, \beta].$$

Here we define the epistemic negation \neg_1 that maps bf-annotations to themselves.

If we consider the before-after measure over the 15 bf-annotations, obviously there exists a partial order ($<_h$) based on the before-after measure, where $\mu_1 <_h \mu_2$ is intuitively interpreted that the bf-annotation μ_1 denotes a more “before” degree than the bf-annotation μ_2 , and $\mu_1, \mu_2 \in \{be, af, db, da, mb, ma, jb, ja, ib, ia, sb, sa, fb, fa, pba\}$. If the bf-annotations μ_1 and μ_2 have the same before-after degree, we denote it $\mu_1 \equiv_h \mu_2$. Then we have the following ordering :

$$\begin{aligned} db <_h mb <_h jb <_h sb <_h ib <_h fb <_h pba <_h \\ fa <_h ia <_h sa <_h ja <_h ma <_h da \\ \text{and} \\ jb \equiv_h be <_h af \equiv_h ja. \end{aligned}$$

Definition 37 (Epistemic Negation \neg_1 for Bf-annotations)

The epistemic negation \neg_1 over the bf-annotations

$$\{be, af, da, db, ma, mb, ja, jb, sa, sb, ia, ib, fa, fb, pba\}$$

is obviously defined as the following mappings :

$$\begin{aligned} \neg_1(af) &= be, & \neg_1(be) &= af, \\ \neg_1(da) &= db, & \neg_1(db) &= da, \\ \neg_1(ma) &= mb, & \neg_1(mb) &= ma, \\ \neg_1(ja) &= jb, & \neg_1(jb) &= ja, \\ \neg_1(sa) &= sb, & \neg_1(sb) &= sa, \\ \neg_1(ia) &= ib, & \neg_1(ib) &= ia, \\ \neg_1(fa) &= fb, & \neg_1(fb) &= fa, \\ \neg_1(pba) &= pba. \end{aligned}$$

On the other hand, if we take the before-after knowledge(information) amount of each bf-relation into account as another measure, obviously there also exists another partial order($<_v$) in terms of the knowledge amount, where $\mu_1 <_v \mu_2$ is intuitively interpreted that the bf-annotation μ_1 has less knowledge amount in terms of before-after relation than the bf-annotation μ_2 . If the annotations μ_1 and μ_2 have the same before-after knowledge amount, we denote it $\mu_1 \equiv_v \mu_2$. Then we have the following ordering:

$$\begin{aligned} \text{be} <_v \mu_1, \quad \mu_1 \in \{ \text{db, mb, jb, sb, ib} \}, \\ \text{af} <_v \mu_2, \quad \mu_2 \in \{ \text{da, ma, ja, sa, ia} \}, \\ \text{db} \equiv_v \text{mb} \equiv_v \text{jb} \equiv_v \text{sb} \equiv_v \text{ib} \equiv_v \text{fb} \equiv_v \text{pba} \equiv_v \\ \text{fa} \equiv_v \text{ia} \equiv_v \text{sa} \equiv_v \text{ja} \equiv_v \text{ma} \equiv_v \text{da} \\ \text{and} \\ \text{be} \equiv_v \text{af}. \end{aligned}$$

If we regard the before-after measure as the horizontal one and the before-after knowledge amount as the vertical one, we obtain the complete bi-lattice $\mathcal{T}_v(12)_{bf}$ of bf-annotations.

$$\begin{aligned} \mathcal{T}_v(12)_{bf} = \{ \perp_{12}(0, 0), \dots, \text{be}(0, 8), \dots, \text{db}(0, 12), \dots, \text{mb}(1, 11), \dots, \\ \text{jb}(2, 10), \dots, \text{sb}(3, 9), \dots, \text{ib}(4, 8), \dots, \text{fb}(5, 7), \dots, \\ \text{be}(6, 0), \dots, \text{pba}(6, 6), \dots, \text{fa}(7, 5), \dots, \text{af}(8, 0), \dots, \\ \text{ia}(8, 4), \dots, \text{sa}(9, 3), \dots, \text{ja}(10, 2), \dots, \text{ma}(11, 1), \dots, \\ \text{da}(12, 0), \dots, \top_{12}(12, 12) \}, \end{aligned}$$

which is described as the Hasse diagram in Fig. 17.

We note that a bf-EVALP literal,

$$\begin{aligned} R(pi, pj, t) : [\mu_1(m, n), \mu_2], \\ \text{where } \mu_2 \in \{ \alpha, \beta, \gamma \} \text{ and} \\ \mu_1 \in \{ \text{be, db, mb, jb, sb, ib, fb, pba, fa, ia, sa, jb, ma, da, af} \}, \end{aligned}$$

is not well annotated if $m \neq 0$ and $n \neq 0$, however, since the bf-EVALP literal is equivalent to the following two well annotated bf-EVALP literals:

$$R(pi, pj) : [(m, 0), \mu] \quad \text{and} \quad R(pi, pj) : [(0, n), \mu].$$

Therefore, such non-well annotated bf-EVALP literals can be dealt with as the conjunction of two well annotated EVALP literals. Take, for example, the non-well annotated bf-EVALP clause,

$$R(pi, pj, t_1) : [(k, l), \mu_1] \leftarrow R(pi, pj, t_2) : [(m, n), \mu_2],$$

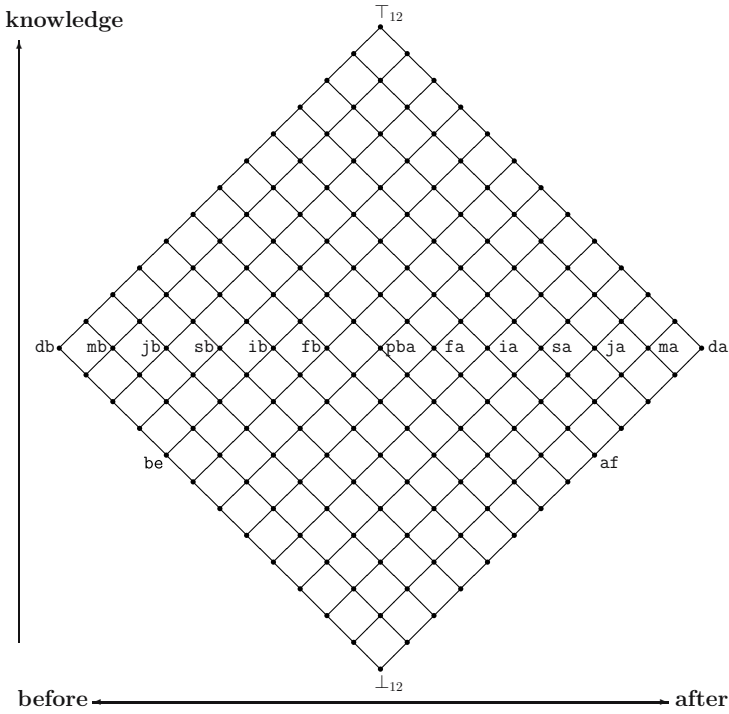


Fig. 17. The complete bi-lattice $\mathcal{T}_v(12)_{bf}$ of bf-annotations

where $k \neq 0$, $l \neq 0$, $m \neq 0$ and $n \neq 0$. It can be equivalently transformed into the two well annotated bf-EVALP clauses,

$$\begin{aligned}
 R(pi, pj, t_1):[(k, 0), \mu_1] \leftarrow R(pi, pj, t_2):[(m, 0), \mu_2] \wedge R(pi, pj, t_2):[(0, n), \mu_2], \\
 R(pi, pj, t_1):[(0, l), \mu_1] \leftarrow R(pi, pj, t_2):[(m, 0), \mu_2] \wedge R(pi, pj, t_2):[(0, n), \mu_2].
 \end{aligned}$$

4.2 Implementation of bf-EVALPSN

We now explain how to implement bf-EVALPSN in process order safety verification systems. For simplicity, we assume that one process does not start/finish with another at the same time, however, the process order control system can deal with immediately before/after relations between two processes. Then, we do not have to take the bf-annotations, **sb/sa**, **fb/fa** and **pba** into account, but only the ten bf-annotations corresponding to the following vector annotations:

$$\begin{aligned}
 \text{before}(\mathbf{be})/\text{after}(\mathbf{af}), & \quad (0, 4)/(4, 0), \\
 \text{discrete before}(\mathbf{db})/\text{after}(\mathbf{da}), & \quad (0, 7)/(7, 0), \\
 \text{immediate before}(\mathbf{mb})/\text{after}(\mathbf{ma}), & \quad (1, 6)/(6, 1),
 \end{aligned}$$

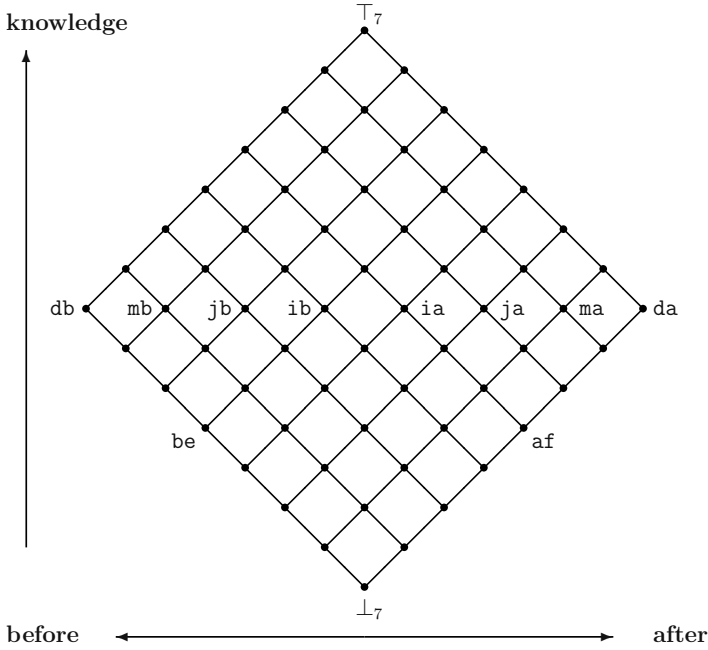


Fig. 18. The complete bi-lattice $\mathcal{T}_v(7)_{bf}$ of Bf-annotations

$$\begin{aligned} \text{joint before}(\mathbf{jb})/\text{after}(\mathbf{ja}), & \quad (2, 5)/(5, 2), \\ \text{included before}(\mathbf{ib})/\text{after}(\mathbf{ia}). & \quad (3, 4)/(4, 3). \end{aligned}$$

The complete bi-lattice $\mathcal{T}_v(7)_{bf}$ of those bf-annotations are described in Fig. 18.

Example 8

Suppose that there are three processes $Pr_1(\text{id } p1)$, $Pr_2(\text{id } p2)$ and $Pr_3(\text{id } p3)$ appearing, and the next process $Pr_4(\text{id } p4)$ not appearing in the process time chart of Fig. 19.

Those processes are supposed to be processed according to the process schedule in Fig. 19. Then, we consider three bf-relations represented by the three bf-EVALP clauses Eqns.(61) – (63) that are inferred by each process start/finish information based on the time sequence t_0, \dots, t_7 in Fig. 19:

$$R(p1, p2, t):[(i_1, j_1), \alpha], \tag{61}$$

$$R(p2, p3, t):[(i_2, j_2), \alpha], \tag{62}$$

$$R(p3, p4, t):[(i_3, j_3), \alpha]. \tag{63}$$

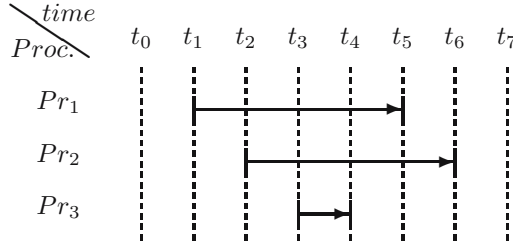


Fig. 19. Process time chart

At time (t_0): no process has started yet. Thus, we have no knowledge in terms of each bf-relation. Therefore, we have the bf-EVALP clauses,

$$\begin{aligned}
 R(p1, p2, t_0) &: [(0, 0), \alpha], \\
 R(p2, p3, t_0) &: [(0, 0), \alpha], \\
 R(p3, p4, t_0) &: [(0, 0), \alpha].
 \end{aligned}$$

At time (t_1): only process Pr_1 has started before process Pr_2 starts, and the bf-literal $R(p1, p2, t_1)$ has the bf-annotation $\mathbf{be}(0, 4)$. Because the four bf-relations ‘disjoint before’ \mathbf{db} (0, 7), ‘immediately before’ \mathbf{mb} (1, 6), ‘joint before’ \mathbf{jb} (2, 5) or ‘included before’ \mathbf{ib} (3, 4) could be the final bf-relation between processes Pr_1 and Pr_2 , thus the greatest lower bound $\mathbf{be}(0, 4)$ of those vector annotations (0, 7), (1, 6), (2, 5) and (3, 4) becomes the vector annotation of the bf-literal $R(p1, p2, t_1)$. Other bf-literals have the bottom vector annotation (0, 0). Therefore, we have the bf-EVALP clauses,

$$\begin{aligned}
 R(p1, p2, t_1) &: [(0, 4), \alpha], \\
 R(p2, p3, t_1) &: [(0, 0), \alpha], \\
 R(p3, p4, t_1) &: [(0, 0), \alpha].
 \end{aligned}$$

At time (t_2): the second process Pr_2 also has started before process Pr_1 finishes. Then, the two bf-relations ‘joint before’ \mathbf{jb} (2, 5) or ‘included before’ \mathbf{ib} (3, 4) could be the final bf-relation between processes Pr_1 and Pr_2 at this time. Thus, the greatest lower bound (2, 4) of the vector annotations (2, 5) and (3, 4) has to become the vector annotation of the bf-literal $R(p1, p2, t_2)$. In addition, the bf-literal $R(p2, p3, t_2)$ has the vector annotation (0, 4) as well as the bf-literal $R(p1, p2, t_1)$, since process Pr_2 has also started before process Pr_3 starts. The bf-literal $R(p3, p4, t_2)$ has the bottom vector annotation (0, 0), since process Pr_3 has not started yet. Therefore, we have the bf-EVALP clauses,

$$\begin{aligned}
 R(p1, p2, t_2) &: [(2, 4), \alpha], \\
 R(p2, p3, t_2) &: [(0, 4), \alpha], \\
 R(p3, p4, t_2) &: [(0, 0), \alpha].
 \end{aligned}$$

At time (t_3): process Pr_3 has started before both processes Pr_1 and Pr_2 finish. Then, the bf-literals $R(p1, p2, t_3)$ and $R(p2, p3, t_3)$ have the same vector annotation $(2, 4)$ as well as the bf-literal $R(p1, p2, t_2)$. Moreover, the bf-literal $R(p3, p4, t_3)$ has the vector annotation $(0, 4)$ as well as the bf-literal $R(p1, p2, t_1)$. Therefore, we have the bf-EVALP clauses,

$$\begin{aligned} R(p1, p2, t_3) &: [(2, 4), \alpha], \\ R(p2, p3, t_3) &: [(2, 4), \alpha], \\ R(p3, p4, t_3) &: [(0, 4), \alpha]. \end{aligned}$$

At time (t_4): process Pr_3 has finished before both processes Pr_1 and Pr_2 finish. Then, the bf-literal $R(p1, p2, t_4)$ still should have the same vector annotation $(2, 4)$ as well as the previous time t_3 . In addition, the bf-literal $R(p2, p3, t_4)$ has its final bf-annotation $\mathbf{ib}(3, 4)$. There are still two alternatives for the final bf-relation between processes Pr_3 and Pr_4 : (1) if process Pr_4 starts immediately after process Pr_3 finishes, the bf-literal $R(p3, p4, t_4)$ has the bf-annotation $\mathbf{mb}(1, 6)$; (2) if process Pr_4 does not start immediately after process Pr_3 finishes, the bf-literal $R(p3, p4, t_4)$ has the bf-annotation $\mathbf{db}(0, 7)$. Either way, we have only the information that process Pr_3 has just finished at time t_4 , which can be represented by the vector annotation $(0, 6)$ that is the greatest lower bound of the vector annotations $(1, 6)$ and $(0, 7)$. Therefore, we have the bf-EVALP clauses,

$$\begin{aligned} R(p1, p2, t_4) &: [(2, 4), \alpha], \\ R(p2, p3, t_4) &: [(3, 4), \alpha], \\ R(p3, p4, t_4) &: [(0, 6), \alpha]. \end{aligned}$$

At time (t_5): process Pr_1 has finished before process Pr_2 finishes. Then, the bf-literal $R(p1, p2, t_5)$ has its final bf-annotation $\mathbf{jb}(2, 5)$, and the bf-literal $R(p3, p4, t_5)$ also has its final bf-annotation $\mathbf{jb}(0, 7)$, because process Pr_4 has not started yet. Therefore, we have the bf-EVALP clauses,

$$\begin{aligned} R(p1, p2, t_5) &: [\mathbf{jb}(2, 5), \alpha], \\ R(p2, p3, t_5) &: [\mathbf{ib}(3, 4), \alpha], \\ R(p3, p4, t_5) &: [\mathbf{db}(0, 7), \alpha], \end{aligned}$$

and all three bf-relations have been determined at time t_5 before process Pr_2 finishes and process Pr_4 starts.

In **Example 8**, the detected process start/finish information was input to the EVALPSN process order safety verification systembf-EVALPSN process order safety verification system to calculate and update the vector annotations of the bf-EVALP literals at each time $t_i (i = 0, 1, 2, \dots, 7)$. In the bf-EVALPSN verification system, the safety of process order is verified before each process starts. For example, if there is a safety property rule “process Pr_i must finish

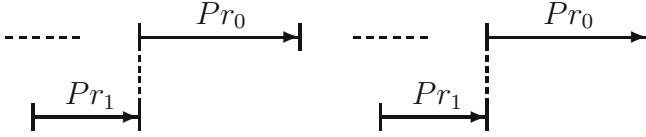


Fig. 20. Bf-EVALP safety verification example

before process Pr_j starts”, then the safety of the process order should be assured before process Pr_j starts. Details of the bf-EVALP safety verification for process order will be introduced with a simple example in the following Section.

4.3 Safety Verification in bf-EVALPSN

We introduce the basic idea of bf-EVALPSN safety verification for process order. Suppose that two processes Pr_0 and Pr_1 are processed repeatedly, and that process Pr_1 must be processed immediately before process Pr_0 starts, as shown in Fig. 20.

In order to verify the safety of the process order, we may have the following safety properties **SP-0** and **SP-1** of processes Pr_0 and Pr_1 :

SP-0 process Pr_0 must start immediately after process Pr_1 finishes,

SP-1 process Pr_1 must start in a while after (disjoint after) process Pr_0 finishes.

Then, safety properties **SP-0** and **SP-1** need to be verified immediately before processes Pr_0 and Pr_1 start, respectively.

In order to verify the bf-relation ‘immediate after’ in the safety property **SP-0**, it should be verified whether process Pr_1 has finished immediate before process Pr_0 starts or not, and the safety verification should be carried out immediately after process Pr_1 finishes. Then the bf-literal $R(p_0, p_1, t)$ must have the vector annotation $(6, 0)$, which denotes that process Pr_1 has finished but that process Pr_0 has not yet started. Therefore, the safety property **SP-0** is translated to the bf-EVALPSN clauses,

SP-0

$$Start(p_0, t) : [(0, 1), \gamma] \leftarrow R(p_0, p_1, t) : [(6, 0), \alpha] \wedge \sim R(p_0, p_1, t) : [da(7, 0), \alpha], \tag{64}$$

$$Start(p_0, t) : [(0, 1), \beta] \leftarrow \sim Start(p_0, t) : [(0, 1), \gamma], \tag{65}$$

where the literal $Start(pi, t)$ represents “process Pr_i starts at time t ”.

On the other hand, in order to verify the bf-relation ‘disjoint after’ in safety property **SP-1**, it needs to be verified whether or not there is a time lag between the process Pr_0 finish time and the process Pr_1 start time. Then,

the bf-literal $R(p1, p0, t)$ must have the vector annotation $\mathbf{da}(7, 0)$. Therefore, the safety property **SP-1** is translated to the bf-EVALPSN clauses,

SP-1

$$Start(p1, t) : [(0, 1), \gamma] \leftarrow R(p1, p0, t) : [\mathbf{da}(7, 0), \alpha], \quad (66)$$

$$Start(p1, t) : [(0, 1), \beta] \leftarrow \sim Start(p1, t) : [(0, 1), \gamma]. \quad (67)$$

Now we show how to use the safety properties **SP-0** and **SP-1** in the bf-EVALPSN clauses Eqns. (64) – (67) for verifying the process order safety. In order to verify the safety for the process order of Fig. 20, the following safety verification cycle is applied repeatedly:

[Safety Verification Cycle]

1st Step (safety verification before process Pr_1 starts):

Suppose that process Pr_1 has not yet started at time t_1 . If process Pr_0 has already finished at time t_1 , we have the bf-EVALP clause,

$$R(p1, p0, t_1) : [(7, 0), \alpha]. \quad (68)$$

On the other hand, if process Pr_0 has just finished at time t_1 , we have the bf-EVALP clause

$$R(p1, p0, t_1) : [(6, 0), \alpha]. \quad (69)$$

If the bf-EVALP clause (68) is input to the safety property **SP-1** Eqns. (66), (67), we obtain the EVALP clause $Start(p1, t_1) : [(0, 1), \gamma]$, and the safety of process Pr_1 starting is assured. On the other hand, if the bf-EVALP clause Eqn. (69) is input to the same safety property **SP-1**, we obtain the EVALP clause $Start(p1, t_1) : [(0, 1), \beta]$, and the safety of process Pr_1 is not assured.

2nd Step (safety verification for process Pr_0 starting):

Suppose that process Pr_0 has not yet started at time t_2 . If process Pr_1 has just finished at time t_2 , we have the bf-EVALP clause,

$$R(p0, p1, t_2) : [(6, 0), \alpha]. \quad (70)$$

On the other hand, if the process Pr_1 has not finished yet at the time t_2 , we have the bf-EVALP clause,

$$R(p0, p1, t_2) : [\mathbf{af}(4, 0), \alpha]. \quad (71)$$

If the EVALP clause Eqn. (70) is input to the safety property **SP-0** Eqns. (64), (65), we obtain the EVALP clause $Start(p0, t_2) : [(0, 1), \gamma]$, and the safety of process Pr_0 is assured. On the other hand, if the bf-EVALP clause Eqn. (71) is input to the same safety property **SP-0**, we obtain the EVALP clause $Start(p1, t) : [(0, 1), \beta]$, and the safety of process Pr_0 is not assured.

Example 9

In this example we show a more practical bf-EVALPSN safety verification for process order. Let us remind ourselves that a more complicated schedule for pipeline processes already appeared in Fig. 7. We propose the following safety properties for the process schedule:

- SPR-0** process Pr_0 must start before any other process,
- SPR-1** process Pr_1 must start immediately after process Pr_0 starts,
- SPR-2** process Pr_2 must start immediately after process Pr_1 finishes,
- SPR-3** process Pr_3 must start immediately after both processes Pr_0 and Pr_2 finish.

Then the safety property **SPR-0** is translated to the bf-EVALPSN clauses,

SPR-0

$$\begin{aligned}
 Start(p0, t) : [(0, 1), \gamma] &\leftarrow \sim R(p0, p1, t) : [(4, 0), \alpha] \wedge \\
 &\quad \sim R(p0, p2, t) : [(4, 0), \alpha] \wedge \\
 &\quad \sim R(p0, p3, t) : [(4, 0), \alpha], \\
 Start(p0, t) : [(0, 1), \beta] &\leftarrow \sim Start(p0, t) : [(0, 1), \gamma]. \quad (72)
 \end{aligned}$$

As well as the safety property **SPR-0**, the other safety properties **SPR-1**, **SPR-2** and **SPR-3** are also translated to the bf-EVALPSN clauses,

SPR-1

$$\begin{aligned}
 Start(p1, t) : [(0, 1), \gamma] &\leftarrow R(p1, p0, t) : [(4, 0), \alpha], \\
 Start(p1, t) : [(0, 1), \beta] &\leftarrow \sim Start(p1, t) : [(0, 1), \gamma], \quad (73)
 \end{aligned}$$

SPR-2

$$\begin{aligned}
 Start(p2, t) : [(0, 1), \gamma] &\leftarrow R(p2, p1, t) : [(6, 0), \alpha] \wedge \\
 &\quad \sim R(p2, p1, t) : [(7, 0), \alpha], \\
 Start(p2, t) : [(0, 1), \beta] &\leftarrow \sim Start(p2, t) : [(0, 1), \gamma], \quad (74)
 \end{aligned}$$

SPR-3

$$\begin{aligned}
 Start(p3, t) : [(0, 1), \gamma] &\leftarrow R(p3, p0, t) : [(6, 0), \alpha] \wedge \\
 &\quad R(p3, p2, t) : [(6, 0), \alpha] \wedge \\
 &\quad \sim R(p3, p2, t) : [(7, 0), \alpha], \\
 Start(p3, t) : [(0, 1), \gamma] &\leftarrow R(p3, p0, t) : [(6, 0), \alpha] \wedge \\
 &\quad R(p3, p2, t) : [(6, 0), \alpha] \wedge \\
 &\quad \sim R(p3, p0, t) : [(7, 0), \alpha], \\
 Start(p3, t) : [(0, 1), \beta] &\leftarrow \sim Start(p3, t) : [(0, 1), \gamma]. \quad (75)
 \end{aligned}$$

Now, we describe the process order safety verification based on the process schedule of Fig. 7.

[Safety Verification Process]

Initial Stage (time t_1): If no process has started at time t_1 , we have no information in terms of all bf-relations between processes Pr_0, Pr_1, Pr_2 and Pr_3 , thus, we have the bf-EVALP clauses,

$$R(p0, p1, t_1) : [(0, 0), \alpha], \quad (76)$$

$$R(p0, p2, t_1) : [(0, 0), \alpha], \quad (77)$$

$$R(p0, p3, t_1) : [(0, 0), \alpha]. \quad (78)$$

In order to verify the safety for starting the first process Pr_0 , the bf-EVALP clauses Eqns. (76) – (78) are input to the safety property **SPR-0** Eqn. (72) in EVALPSN. Then we obtain the EVALP clause $Start(p0, t_1) : [(0, 1), \gamma]$, and the safety for the process Pr_0 start is assured at time t_1 ; otherwise, it is not assured.

2nd Stage (time t_2): Suppose that only process Pr_0 has already started at time t_2 . Then, we have the bf-EVALP clauses,

$$R(p1, p0, t_2) : [(4, 0), \alpha]. \quad (79)$$

In order to verify the safety of starting the second process Pr_1 , the bf-EVALP clause Eqn. (79) is input to the safety property **SPR-1** Eqn. (73) in bf-EVALPSN. Then, we obtain the EVALP clause $Start(p1, t_2) : [(0, 1), \gamma]$, and the safety of process Pr_1 starting is assured at time t_2 ; otherwise, it is not assured.

3rd Stage (time t_3): Suppose that processes Pr_0 and Pr_1 have already started, and neither of them has finished yet at time t_3 . Then, we have the bf-EVALP clauses,

$$R(p2, p0, t_3) : [(4, 0), \alpha], \quad (80)$$

$$R(p2, p1, t_3) : [(4, 0), \alpha]. \quad (81)$$

$$R(p1, p3, t_3) : [(0, 4), \alpha]. \quad (82)$$

In order to verify the safety of starting the third process Pr_2 , if the EVALP clause Eqn. (81) is input to the safety property **SPR-2** Eqn. (74) in bf-EVALPSN, then, we obtain the EVALP clause $Start(p2, t_3) : [(0, 1), \beta]$, and the safety of process Pr_2 starting is not assured at time t_3 . On the other hand, if process Pr_1 has just finished at time t_3 , then we have the bf-EVALP clause

$$R(p2, p1, t_3) : [(6, 0), \alpha]. \quad (83)$$

If the bf-EVALP clause (83) is input to the safety property **SPR-2** Eqn. (74) in EVALPSN, then we obtain the EVALP clause $Start(p2, t_3) : [(0, 1), \gamma]$, and the safety of process Pr_2 starting is assured.

4th Stage (time t_4): Suppose that processes Pr_0 , Pr_1 and Pr_2 have already started, that processes Pr_0 and Pr_1 have already finished, and only process Pr_3 has not yet started at time t_4 . Then, we have the bf-EVALP clauses

$$R(p3, p0, t_4) : [(7, 0), \alpha], \quad (84)$$

$$R(p3, p1, t_4) : [(7, 0), \alpha], \quad (85)$$

$$R(p3, p2, t_4) : [(4, 0), \alpha]. \quad (86)$$

In order to verify the safety of starting the last process Pr_3 , if the bf-EVALP clauses Eqn. (84) and Eqn. (86) are input to the safety property **SPR-3** Eqn. (75) in bf-EVALPSN, then we obtain the EVALP clause $Start(p3, t_4) : [(0, 1), \beta]$, and the safety of process Pr_3 starting is not assured at time t_4 . On the other hand, if process Pr_2 has just finished at time t_4 , then we have the bf-EVALP clause

$$R(p3, p2, t_4) : [(6, 0), \alpha]. \quad (87)$$

If the bf-EVALP clause Eqn. (87) is input to the safety property **SPR-3** Eqn. (75) in bf-EVALPSN, then we obtain the EVALP clause $Start(p3, t_4) : [(0, 1), \gamma]$, and the safety of process Pr_3 starting is assured.

5 Conclusion and Future Work

In this Chapter, we have introduced a paraconsistent annotated logic program called EVALPSN that can deal with defeasible deontic reasoning and contradiction as knowledge, and applied it to the formal safety verification for pipeline valve control. Moreover we have proposed a new EVALPSN called bf-EVALPSN that can deal with before-after relations between two processes as paraconsistent notions represented in vector annotations, and applied it to process order safety verification for pipeline processes.

Our formal safety verification methods based on EVALPSN/bf-EVALPSN have the following advantages and disadvantages:

Advantages

If the safety verification EVALPSN is locally stratified, it can be easily implemented in Prolog, C language, PLC (Programmable Logic Controller) ladder program, and so forth.

It has been proved that the method can be implemented as electronic circuits on micro chips [41]. Therefore, if real-time processing is required in the system, the method might be very useful.

Our bf-EVALP safety verification method for process order control is an essential application of paraconsistent annotated logic in the sense of treating before-after relations between processes in paraconsistent vector annotations.

Safety verification methods for both process control and process order control can be implemented under the same environment.

Disadvantages

Since EVALPSN itself is basically not a tool for formal safety verification, it includes complicated and redundant expressions to construct safety verification systems. Therefore, it would be better to develop a safety verification oriented tool or programming language based on EVALPSN if EVALPSN is applied to formal safety verification.

It is difficult to understand how to fully utilize EVALPSN for practical implementations due to paraconsistent annotated logic.

In the future, we are considering developing (i) a more efficient bf-EVALPSN process order safety verification system for real-time processing, and (ii) EVALPSN to deal with temporal reasoning and/or tense notion (such as past and future) toward more practical applications.

Acknowledgements

First of all I would like to thank the Editors – Prof. John Fulcher and Prof. Lakhmi C. Jain – for giving me the opportunity to contribute to this Handbook. Throughout the preparation of this chapter, I have benefited from the constructive suggestions and comments of various people. Prof. Fulcher and Dr. Patrick T. Dougherty kindly assisted me in proofreading and correcting my English; my sincere thanks go to them both.

References

1. Apt K, et al. (1988) Towards a theory of declarative knowledge. In: Minker J (ed) *Foundation of Deductive Database and Logic Programs*. Morgan Kaufmann, San Mateo, CA: 89–148.
2. Belnap ND (1977) A useful four-valued logic. In: Reidel D (ed) *Modern Uses of Many-Valued Logic*: 8–37.
3. Billington D (1993) Defeasible logic is Stable, *J.Logic and Computation*, (3): 379–400.
4. Billington D (1997) Conflicting literals and defeasible logic. In: Nayak A, Pagnucco M (eds) *Proc. 2nd Australian Workshop Commonsense Reasoning*, 1 December, Perth, Western Australia, The Australian Computer Society: 1–15.
5. Blair HA, Subrahmanian VS (1989) Paraconsistent logic programming. *Theoretical Computer Science*, 68: 135–154.

6. da Costa NCA, Subrahmanian VS, Vago C (1989) The paraconsistent logics *PT*. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 37: 139–148.
7. Dressler O (1988) An extended basic ATMS. In: Reinfrank M, et al. (eds) *Proc. 2nd Intl. Workshop Non-monotonic Reasoning*, 13–15 June, Grassau, Germany. Lecture Notes in Computer Science LNCS 346, Springer-Verlag, Berlin: 143–154.
8. Fitting M (1991) Bilattice and the semantics of logic programming. *J. Logic Programming*, 11: 91–116.
9. Geerts P, Laenens F, Vermeir D (1998) Defeasible logics In: Gabbay DM, Smets PH (eds) *Handbook of Defeasible Reasoning and Uncertainty Management Systems 2*. Kluwer Academic Publishers, The Netherlands: 175–210.
10. Gelfond M, Lifschitz V (1989) The stable model semantics for logic programming. In: Kowalsky RA, Bowen KA (eds) *Proc. 5th Intl. Conf. and Symp. Logic Programming (ICLP/SLPSS)*, 15–19 August, Seattle, WA, MIT Press, Cambridge, MA: 1070–1080.
11. Gelder AV, Ross KA, Schlipf JS (1991) The well-founded semantics for general logic programs. *J. ACM*, 38: 620–650.
12. Gordon MJC, Melham TF (1993) *Introduction to HOL*. Cambridge University Press, UK.
13. Jaskowski S (1948) Propositional calculus for contradictory deductive system (English translation of the original Polish paper). *Studia Logica*, 24: 143–157.
14. Kifer M, Subrahmanian VS (1992) Theory of generalized annotated logic programming and its applications. *J. Logic Programming*, 12: 335–368.
15. Lloyd JW (1987) *Foundations of Logic Programming (2nd ed)*. Springer-Verlag, Berlin.
16. Meyer J-JC, Wiering RJ (eds) *Deontic Logic in Computer Science*. Wiley, Chichester, UK.
17. McNamara P, Prakken H (eds) (1999) *Norms, Logics and Information Systems (New Studies in Deontic Logic and Computer Science, Frontiers in Artificial Intelligence and Applications 49)*. IOS Press, The Netherlands.
18. Moore R (1985) Semantical considerations on non-monotonic logic. *Artificial Intelligence*, 25: 75–94.
19. Morley JM (1996) Safety Assurance in Interlocking Design. *PhD Thesis*, School of Informatics, University of Edinburgh, UK.
20. Nakamatsu K, Suzuki A (1994) Annotated semantics for default reasoning. In: Dai R (ed) *Proc. 3rd Pacific Rim Intl. Conf. Artificial Intelligence (PRICAI94)*, 15–18 August, Beijing, China. International Academic Publishers, China: 180–186.
21. Nakamatsu K, Suzuki A (1998) A nonmonotonic ATMS based on annotated logic programs. In: Wobcke W, et al. (eds) *Agents and Multi-Agents Systems*, Lecture Notes in Artificial Intelligence LNAI 1441. Springer-Verlag, Berlin: 79–93.
22. Nakamatsu K, Abe JM (1999) Reasonings based on vector annotated logic programs. In: Mohammadian M (ed) *Computational Intelligence for Modelling, Control & Automation (CIMCA99) (Concurrent Systems Engineering Series 55)*. IOS Press, The Netherlands: 396–403.
23. Nakamatsu K, Abe JM, Suzuki A (1999) Defeasible reasoning between conflicting agents based on VALPSN. In: Tessier C, Chaudron L (eds) *Proc. AAAI Workshop Agents Conflicts*. 18 July, Orlando, FL. AAAI Press, Menlo Park, CA: 20–27.

24. Nakamatsu K, Abe JM, Suzuki A (1999) Defeasible reasoning based on VALPSN and its application. In: Nayak A, Pagnucco M (eds) *Proc. 3rd Australian Commonsense Reasoning Workshop*, 7 December, Sydney, Australia, University of Newcastle: 114–130.
25. Nakamatsu K (2001) On the relation between vector annotated logic programs and defeasible theories. *Logic and Logical Philosophy*, 8: 181–205.
26. Nakamatsu K, Abe JM, Suzuki A (2001) A defeasible deontic reasoning system based on annotated logic programming. In: Dubois DM (ed) *Proc. 4th Intl. Conf. Computing Anticipatory Systems (CASYS2000)*(AIP Conference Proceedings 573), 7–12 August, Liege, Belgium. American Institute of Physics, New York, NY: 609–620.
27. Nakamatsu K, Abe JM, Suzuki A (2001) Annotated semantics for defeasible deontic reasoning. In: Ziarko W, Yao Y (eds) *Proc. 2nd Intl. Conf. Rough Sets and Current Trends in Computing (RSCTC2000)*, 16–19 October, Banff, Canada, (Lecture Notes in Artificial Intelligence LNAI 2005). Springer-Verlag, Berlin: 432–440.
28. Nakamatsu K, Abe JM, Suzuki A (2002) Extended vector annotated logic program and its application to robot action control and safety verification. In: Abraham A, et al. (eds) *Hybrid Information Systems (Advances in Soft Computing Series)*. Physica-Verlag, Heidelberg, Germany: 665–680.
29. Nakamatsu K, Suito H, Abe JM, Suzuki A (2002) Paraconsistent logic program based safety verification for air traffic control. In: El Kamel A, et al. (eds) *Proc. IEEE Intl. Conf. System, Man and Cybernetics (SMC02)*, 6–9 October, Hammamet, Tunisia. IEEE SMC (CD-ROM).
30. Nakamatsu K, Abe JM, Suzuki A (2002) A railway interlocking safety verification system based on abductive paraconsistent logic programming. In: Abraham A, et al. (eds) *Soft Computing Systems (HIS02) – Frontiers in Artificial Intelligence and Applications 87*. IOS Press, The Netherlands: 775–784.
31. Nakamatsu K, Abe JM, Suzuki A (2002) Defeasible deontic robot control based on extended vector annotated logic programming. In: Dubois DM (ed) *Proc. 5th Intl. Conf. Computing Anticipatory Systems (CASYS2001)*(AIP Conference Proceedings 627), 13–18 August, Liege, Belgium. American Institute of Physics, New York, NY: 490–500.
32. Nakamatsu K, Mita Y, Shibata T (2003) Defeasible deontic action control based on paraconsistent logic program and its hardware application. In: Mohammedian M (ed) *Proc. Intl. Conf. Computational Intelligence for Modelling Control and Automation (CIMCA2003)*, 12–14 February, Vienna, Austria. IOS Press, The Netherlands (CD-ROM).
33. Nakamatsu K, Seno T, Abe JM, Suzuki A (2003) Intelligent real-time traffic signal control based on a paraconsistent logic program EVALPSN. In: Wang G, et al. (eds) *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGr2003)*, 26–29 May, Chongqing, China. Lecture Notes in Artificial Intelligence LNAI 2639. Springer-Verlag, Berlin: 719–723.
34. Nakamatsu K, Komaba H, Suzuki A (2004) Defeasible deontic control for discrete events based on EVALPSN. In: Tsumoto S, et al. (eds) *Proc. 4th Intl. Conf. Rough Sets and Current Trends in Computing (RSCTC2004)*, 1–5 June, Uppsala, Sweden, Lecture Notes in Artificial Intelligence LNAI 3066. Springer-Verlag, Berlin: 310–315.

35. Nakamatsu K, Ishikawa R, Suzuki A (2004) A paraconsistent based control for a discrete event cat and mouse. In: Negoita MG, et al. (eds) *Proc. 8th Intl. Conf. Knowledge-Based Intelligent Information and Engineering Systems (KES2004)*, 20–25 September, Wellington, NZ. Lecture Notes in Artificial Intelligence LNAI 3214, Springer-Verlag, Berlin: 954–960.
36. Nakamatsu K, Chung S-L, Komaba H, Suzuki A (2005) A discrete event control based on EVALPSN stable model. In: Slezak D, et al. (eds) *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC2005)*, 31 August – 3 September, Regina, Canada. Lecture Notes in Artificial Intelligence LNAI 3641. Springer-Verlag, Berlin: 671–681.
37. Nakamatsu K, Abe JM, Akama S (2005) An intelligent safety verification based on a paraconsistent logic program. In: Khosla R, et al. (eds) *Proc. 9th Intl. Conf. Knowledge-Based Intelligent Information and Engineering Systems (KES2005)*, 14–16 September, Melbourne, Australia. Lecture Notes in Artificial Intelligence LNAI 3682, Springer-Verlag, Berlin: 708–715.
38. Nakamatsu K, Kawasumi K, Suzuki A (2005) Intelligent verification for pipeline based on EVALPSN. In: Nakamatsu K, Abe JM (eds) *Advances in Logic Based Intelligent Systems*, Frontiers in Artificial Intelligence and Applications 132. IOS Press, The Netherlands: 63–70.
39. Nakamatsu K, Suzuki A (2005) Autoepistemic theory and paraconsistent logic program. In: Nakamatsu K, Abe JM (eds) *Advances in Logic Based Intelligent Systems*, Frontiers in Artificial Intelligence and Applications 132. IOS Press, The Netherlands: 177–184.
40. Nakamatsu K, Suzuki A (2005) Annotated semantics for non-monotonic reasonings in artificial intelligence – I, II, III, IV. In: Nakamatsu K, Abe JM (eds) *Advances in Logic Based Intelligent Systems*, Frontiers in Artificial Intelligence and Applications 132. IOS Press, The Netherlands: 185–215.
41. Nakamatsu K, Mita Y, Shibata T (2007) An intelligent action control system based on extended vector annotated logic program and its hardware implementation. *J. Intelligent Automation and Soft Computing*, 13(3): 289–304.
42. Nute D (1987) Defeasible reasoning. In: Stohr EA, et al. (eds) *Proc. 20th Hawaii Intl. Conf. System Science (HICSS87)*, 6–9 January, Kailua-Kona, Hawaii. University of Hawaii, Hawaii: 470–477.
43. Nute D (1992) Basic defeasible logics. In: del Cerro LF, Penttonen M (eds) *Intensional Logics for Programming*. Oxford University Press, UK: 125–154.
44. Nute D (1994) Defeasible logic In: Gabbay DM, et al. (eds) *Handbook of Logic in Artificial Intelligence and Logic Programming 3*. Oxford University Press, UK: 353–396.
45. Nute D (1997) Apparent obligatory. In: Nute D (ed) *Defeasible Deontic Logic*, Synthese Library 263, Kluwer Academic Publishers, The Netherlands: 287–316.
46. Prakken H (1997) *Logical tools for modelling legal argument*. Law and Philosophy Library 32. Kluwer Academic Publishers, The Netherlands.
47. Przymusiński TC (1988) On the declarative semantics of deductive databases and logic programs. In: Minker J (ed) *Foundation of Deductive Database and Logic Programs* Morgan Kaufmann: 193–216.
48. Reiter R (1980) A logic for default reasoning. *Artificial Intelligence*, 13: 81–123.
49. Shepherdson JC (1998) Negation as failure, completion and stratification. In: Gabbay DM, et al. (eds) *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, Oxford University Press, UK: 356–419.

50. Subrahmanian VS (1994) Amalgamating knowledge bases. *ACM Trans. Database Systems*, 19: 291–331.
51. Subrahmanian VS (1987) On the semantics of qualitative logic programs. In: *Proc. 4th IEEE Symp. Logic Programming (SLP87)*, 31 August – 4 September, IEEE Computer Society Press, San Francisco, CA: 178–182.
52. Visser A (1987) Four valued semantics and the liar. *J. Philosophical Logic*, 13: 99–112.

Resources

1 Logic Programming

Lloyd JW (1987) *Foundations of Logic Programming (2nd ed)*. Springer-Verlag, Berlin.

Gabbay DM et al. (eds) (1998) *Logic Programming – Handbook of Logic in Artificial Intelligence and Logic Programming Vol.5*. Oxford University Press, UK.

2 Paraconsistent Annotated Logic

da Costa NCA, Subrahmanian VS, Vago C (1989) The paraconsistent logics *PT*. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 37: 139–148.

Blair HA, Subrahmanian VS (1989) Paraconsistent logic programming. *Theoretical Computer Science*, 68: 135–154.

3 Defeasible Logic

Nute D (1992) Basic defeasible logics. In: del Cerro LF, Penttonen M (eds) *Intensional Logics for Programming* Oxford University Press, UK: 125–154.

Geerts P, Laenens F, Vermeir D (1998) Defeasible logics. In: Gabbay DM, Smets PH (eds) *Handbook of Defeasible Reasoning and Uncertainty Management Systems 2* Kluwer Academic Publishers, The Netherlands: 175–210.

4 Defeasible Deontic Logic

Nute D (ed) *Defeasible Deontic Logic* Synthese Library 263, Kluwer Academic Publishers, The Netherlands.

5 ALPSN, VALPSN, EVALPSN

Nakamatsu K, Suzuki A (1994) Annotated semantics for default reasoning. In: Dai R (ed) *Proc. 3rd Pacific Rim Intl. Conf. Artificial Intelligence (PRICAI94)*, 15–18 August, Beijing, China, International Academic Publishers, China: 180–186.

Nakamatsu K (2001) On the relation between vector annotated logic programs and defeasible theories. *Logic and Logical Philosophy*, 8: 181–205.

Nakamatsu K, Abe JM, Suzuki A (2001) Annotated semantics for defeasible deontic reasoning. In: Ziarko W, Yao Y (eds) *Proc. 2nd Intl. Conf. Rough Sets and Current Trends in Computing (RSCTC2000)*, Banff, 16–19 October, Banff, Canada. Lecture Notes in Artificial Intelligence LNAI 2005, Springer-Verlag, Berlin: 432–440.

6 EVALPSN Safety Verification

Nakamatsu K (2004) Intelligent information systems based on paraconsistent logic programs. In: Abraham A, Jain LC (eds) *Innovations in Intelligent Systems and Applications (Studies in Fuzziness and Soft Computing Series 140)*. Springer-Verlag, Berlin: 257–283.

Nakamatsu K (2006) Pipeline valve control based on EVALPSN safety verification. *J. Advanced Computational Intelligence and Intelligent Informatics*, 10: 647–656.

The Data-Oriented Parsing Approach: Theory and Application

Rens Bod

School of Computer Science, University of St. Andrews, Scotland,
rens.bod@gmail.com

1 Introduction

Parsing models have many applications in AI, ranging from natural language processing (NLP) and computational music analysis to logic programming and computational learning. Broadly conceived, a parsing model seeks to uncover the underlying structure of an input, that is, the various ways in which elements of the input combine to form phrases or constituents and how those phrases recursively combine to form a tree structure for the whole input. During the last fifteen years, a major shift has taken place from rule-based, deterministic parsing to corpus-based, probabilistic parsing. A quick glance over the NLP literature from the last ten years, for example, indicates that virtually all natural language parsing systems are currently probabilistic. The same development can be observed in (stochastic) logic programming and (statistical) relational learning. This trend towards probabilistic parsing is not surprising: the increasing availability of very large collections of text, music, images and the like allow for inducing statistically motivated parsing systems from actual data.

A corpus-based parsing approach that has been quite successful in various fields of AI, is known as Data-Oriented Parsing or DOP. DOP was originally developed as an NLP technique but has been generalized to music analysis, problem-solving and unsupervised structure learning [7, 8, 14, 81]. The distinctive feature of the DOP approach, when it was first presented, was to model sentence structures on the basis of previously observed frequencies of sentence-structure fragments, without imposing any constraints on the *size* of these fragments. Fragments include, for instance, subtrees of depth 1 (corresponding to context-free rules), as well as entire trees.

The DOP model was different from all other statistical parsing models at the time. Other models typically started off with a predefined grammar and used a corpus only for estimating the rule probabilities [5, 6, 27, 49, 80]. The DOP model, on the other hand, proposed not to train a predefined

grammar on a corpus, but to directly use corpus fragments as a grammar. This approach has now gained wide usage, as exemplified by the work of [29, 30, 32, 36, 37, 55, 66 and many others].

The other innovation of DOP was to take all corpus fragments, of any size, rather than a small subset. During the last few years, we can observe a shift towards using more and larger corpus fragments with fewer restrictions in parsing models: while the models of [35] and [46] still restricted the fragments to the locality of head-words, later models showed the importance of including context from higher nodes in the tree [29, 60]. The importance of including nonhead-words has also become accepted [for example, 30, 37]. Moreover, [38] argues for “keeping track of counts of arbitrary fragments within parse trees”, which has indeed been carried out in [39] and others, who use exactly the same set of (all) subtrees from a parsed corpus as already proposed in [7].

To date, one of the most robust empirical results in natural language parsing is that the parse accuracy increases if larger subtrees are included in the grammar [for instance, 11, 14, 40, 56, 84]. Although the use of all subtrees was for a long time deemed too costly, efficient algorithms have now been developed, ranging from compact Probabilistic Context-Free Grammar (PCFG) reductions of DOP [53] to tree kernels for all-subtrees models [40]. Consequently, the DOP model has been employed to boost a number of concrete applications, such as dialog processing [9], speech understanding [84] and machine translation [55, 57].

In the meantime, the DOP approach has been generalized to other modalities, including music analysis and problem solving. It has turned out that probabilistic corpus-based parsing outperforms deterministic rule-based processing not only for language but also for melodic analysis [12, 13] and problem solving [19, 20]. Our goal for this Chapter is therefore to present the DOP approach from a multi-modal perspective. But in order to do, it is convenient to first explain DOP for language processing, after which we discuss an integrated DOP model that unifies the different modalities. We will go into the various computational issues and show how the model can be tested against hand-annotated corpora. Finally, we will discuss shortcomings of this supervised approach, and present some results of recent work that extends DOP towards unsupervised learning.

2 A DOP Model for Language: Combining Likelihood and Simplicity

The main motivation behind DOP is to integrate rule-based and exemplar-based aspects of natural language. DOP is rule-based in that it proposes a generative system of productive units; it is exemplar-based in that its productive units are concrete fragments from representations of previous input. An

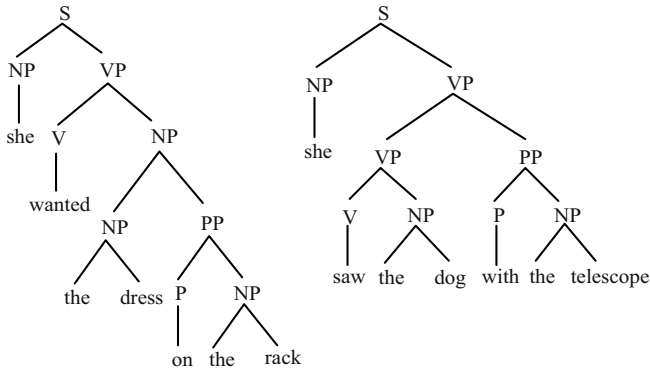


Fig. 1. A small training set of two tree structures

example from language may illustrate the approach. Suppose that we start with a very small corpus of only two sentences with their phrase-structure trees that are labeled by traditional lexical-syntactic categories, as shown in Fig. 1. Here NP = noun phrase, VP = verb phrase, and PP = prepositional phrase. (Note that actual corpora like the **Penn Treebank** contain tens of thousands of trees – see [74]).

To dispel dogmatic slumbers it is good to realize that a corpus of annotated sentences need not be produced by means of a separate grammar or parser. Instead, most existing annotated corpora or ‘treebanks’ are created by human annotators who are only given an annotation guideline with some example analyses of sentences. One could claim that human annotators use an internalized grammar to annotate the sentences, but recent work by [63] and [17, 18] has shown that a contrasting position is just as viable: humans learn to understand and produce new sentences entirely in a statistical, item-based way (see [87] for a psycholinguistic motivation). We will go into the unsupervised learning of tree structures in Sect. 9, but for the moment we will start out from corpora that are already annotated.

Turning back to the corpus in Fig. 1, a new sentence can be derived by combining subtrees from the trees in the corpus. The combination operation between subtrees used by DOP is called *label substitution*, indicated as ‘ \circ ’. The substitution operation identifies the leftmost nonterminal leaf node of one subtree with the root node of a second subtree – that is, the second subtree is substituted on the leftmost nonterminal leaf node of the first subtree, provided that their categories match. Starting out with the corpus of Fig. 1, for instance, the sentence “She saw the dress with the telescope” may be derived as shown in Fig. 2.

In Fig. 2, the sentence “She saw the dress with the telescope” is interpreted analogously to the corpus sentence “She saw the dog with the telescope”: both sentences receive the same phrase structure where the prepositional phrase

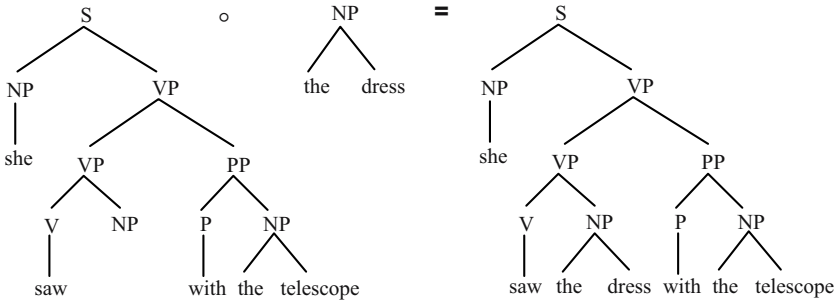


Fig. 2. Analyzing a new sentence by combining subtrees from Fig. 1

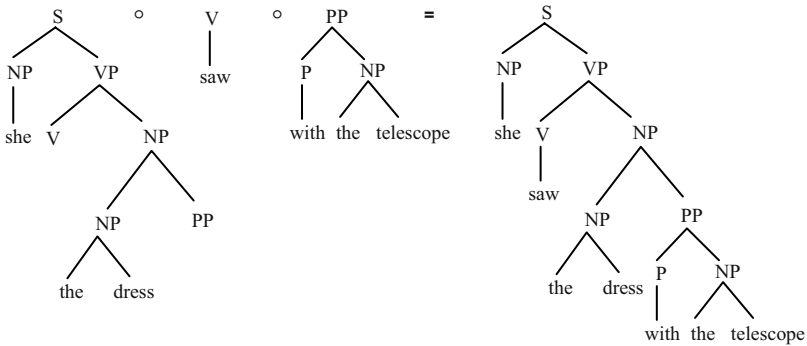


Fig. 3. A different derivation for “She saw the dress with the telescope”

“with the telescope” is attached to “the VP saw the dress”. We can also derive an alternative phrase structure for the test sentence, namely by combining three (rather than two) subtrees from Fig. 1, as shown in Fig. 3. We will write $(t \circ u) \circ v$ as $t \circ u \circ v$ with the convention that \circ is left-associative. In Fig. 3, the sentence “She saw the dress with the telescope” is analyzed in a different way where the PP with the telescope is attached to the NP the dress, corresponding to a different meaning than the tree in Fig. 2. Thus the sentence is ambiguous in that it can be derived in (at least) two different ways which is analogous either to the first or second tree in Fig. 1. Both analyses can in principle be perceived by humans although the first analysis in Fig. 2 is generally seen as the most plausible one.

Note that subtrees can be of arbitrary size: they can range from just one categorized word to entire sentence-analyses. This allows DOP to take into account both the rule-based nature of linguistic productivity and the exemplar-based nature of idiomatic phrases, multi-word units and other idiosyncrasies in language [52]. Also note that an unlimited number of other sentences can be derived by combining subtrees from the corpus, such as “She saw the dress on the rack with the telescope” and “She saw the dress with the dog on the rack

with the telescope”, and so forth. Thus we can get infinite productivity from a finite corpus of representations (see [21] for further examples and how this argues against [34]). Most of these sentences are highly ambiguous: many different analyses can be assigned to each of them due to a combinatorial explosion of different prepositional-phrase attachments. Yet, most of these analyses are not plausible: they do not correspond to the structures humans come up with. The phenomenon that the same input can have many different structural organizations while only one of them tends to be perceived is known as the ‘ambiguity problem’. This problem is one of the hardest problems in artificial intelligence and cognitive science. [34:37] estimates that almost every sentence from the Wall Street Journal has many – often more than one million – possible phrase-structure trees!

How can we select from the possible trees of a sentence the ‘best’ tree as assigned by humans? DOP basically employs a statistical methodology: it computes the best tree from the relative frequencies of partial trees in a large corpus of previous trees (of sentences). Results from psycholinguistics indeed support the idea that the frequency of occurrence of a structure is a very important factor in language comprehension. In particular, frequencies play a key role in disambiguation and well-formedness judgments of sentences (refer to [62] or [72] for overview papers). [50] even argue that “knowledge of grammar includes knowledge of probabilities of syntactic structures”.

Of course, the frequency of occurrence of a structure is not the only factor in syntactic disambiguation. Discourse context and semantics also play an important role. In [8, 9], we have shown how discourse and semantics can be incorporated into DOP if we have corpora containing discourse structure and semantic annotations. The other main disambiguation factor that we will go into here, and that appears to be essentially different from frequency, is the notion of ‘simplicity’ of a structure. It has often been claimed that there is strong preference in favor of the simplest analysis consisting of the smallest number of derivation steps (see [31, 48]). The preference for simplicity may be in competition with the preference for likelihood. Although some accounts propose that likelihood and simplicity are parts of the same coin [31], we showed in [12] that the two principles appear to play a rather distinct role in language processing.

If we indeed take the ‘simplest’ parse tree as the one that can be derived by the smallest number of steps, which in our case is the smallest number of subtrees, then the simplest analysis corresponds to the shortest derivation, which differs almost always from the most likely analysis [12]. An interesting property of the shortest derivation of a sentence is that it corresponds to the parse tree produced by the largest possible subtrees from the corpus. This means that the preference for simplicity can also be seen as a preference for maximizing the structural similarity or analogy between a sentence and the corpus, regardless of the frequencies of the subtrees. On the other hand, the principle of likelihood favors the parse tree that can be constructed out

of the likeliest subtrees, by computing the total likelihood of the parse tree from the relative frequencies of the subtrees – regardless of the size of these subtrees.

In [12], we investigated a number of ways to balance these two principles. We came up with a model that selected the tree generated by the shortest derivation from among the top of the distribution of most likely trees for a certain input. A drawback of this approach is that by selecting the shortest derivation from the most likely trees, we first have to compute (the top of) the probability distribution of trees for an input string. While this is feasible for large corpora such as the linguistic **Penn Treebank** [74] or the musical **Essen Folksong Collection** [82], containing tens of thousands of analyses, it is not so for the much smaller corpora for deductive explanations or proof trees used in modeling problem-solving (see Sect. 6). As a consequence, the probability distribution of trees for a new input is almost flat for such corpora, resulting in poor predictions for the ‘best’ tree. Therefore the model in [12] does not suffice as a general disambiguation technique.

In [16, 20], we proposed an alternative combination of simplicity and likelihood within the DOP framework which seems to be a better candidate for a modality-independent integration of the two principles. According to this combination, the best tree is the one that is generated by the shortest derivation consisting of the fewest subtrees, and in case the shortest derivation is not unique we select the most probable tree from among the shortest derivations. Since in almost all real-world situations the shortest derivation is indeed not unique, this approach *de facto* takes into account both simplicity and likelihood, and can still be described by an overall probabilistic model (see below).

We will illustrate this integration with the linguistic example given above. We start with the criterion of simplicity, in other words the shortest derivation. According to this criterion the tree structure in Fig. 2 would be preferred because it can be generated by just two subtrees from the training set. Any other tree structure, such as in Fig. 3, would need at least three subtrees from the training set in Fig. 1. Note that the tree generated by the shortest derivation indeed tends to be structurally more similar to the corpus (that is, having a larger overlap with one of the corpus trees) than the tree generated by the longer derivation.

Had we restricted the subtrees to smaller sizes – for example to depth-1 subtrees, which makes DOP equivalent to a (stochastic) context-free grammar – the shortest derivation would not be able to distinguish between the two trees in Figs. 2 and 3 as they would both be generated by 9 rewrite rules. The same is true if we used subtrees of maximal depth 2 or 3. It seems that only if we do not restrict the subtree depth can we take into account arbitrarily far-ranging dependencies and model new sentences as closely as possible on previous sentence-analyses by the shortest derivation. It is of course an

empirical question as to how far the inclusion of large subtrees also leads to better predictions for the tree structure as assigned by humans (Sect. 8).

When the shortest derivation is not unique, our model selects from the remaining derivations the one that generates the most probable tree (as computed from the relative frequencies of the subtrees – see below). Why don't we do this the other way round – more specifically, why don't we first compute the most probable tree(s) of a sentence and next select the shortest derivation? We already mentioned above that such an approach obtains poor predictions of the best tree in domains with highly specific labels for which it is difficult to gather meaningful statistics. Instead, by first computing the shortest derivations we constrain the set of candidates for the best tree to those that are structurally most similar to trees in the corpus, on which next statistical computations are applied.

We refer to this instantiation of the DOP framework as DOP+ [20] which we will now define more formally. Let us first give the definition of an analysis tree of an input string (see Definition 1).

Definition 1 *Given a corpus C of trees T_1, T_2, \dots, T_n , and a label substitution operation \circ , then an analysis tree of an input string W with respect to C is a tree T such that (i) there are subtrees t_1, t_2, \dots, t_k in T_1, T_2, \dots, T_n for which $t_1 \circ t_2 \circ \dots \circ t_k = T$, (ii) the root of T is equal to the distinguished symbol S and (iii) the yield of T is equal to W .*

The tree generated by the shortest derivation T_{sd} according to DOP+ is given by Definition 2.

Definition 2 *Let $L(d)$ be the length of derivation d in terms of its number of subtrees, that is if $d = t_1 \circ t_2 \circ \dots \circ t_k$ then $L(d) = k$. Let d_T be a derivation which results in tree T . Then T_{sd} is the tree which is produced by a derivation of minimal length: $T_{sd} = \operatorname{argmin}_T L(d_T)$.*

If T_{sd} is not unique, we select from among the shortest derivations the tree with highest probability. The probability of a tree is defined in terms of the probabilities of the derivations that generate it, which are in turn defined in terms of the probabilities of the subtrees these derivations consist of [8], as given by Definition 3.

Definition 3 *The probability of a subtree t , $P(t)$, is defined as the number of occurrences of t , $|t|$, divided by the total number of occurrences of treebank-subtrees that have the same root label as t . (For practical purposes this simple relative frequency may be adjusted if the frontier of t contains unknown words – see Sect. 8).*

Let $r(t)$ return the root label of t . Then we may write:

$$P(t) = \frac{|t|}{\sum_{t':r(t')=r(t)} |t'|} \quad (1)$$

Under the assumption that subtrees are stochastically independent, the probability of a derivation $t_1 \circ \dots \circ t_k$ is defined as the product of the probabilities of its subtrees t_i :

$$P(t_1 \circ \dots \circ t_k) = \prod_i P(t_i) \quad (2)$$

There may be different derivations that generate the same analysis tree. Assuming that the derivations partition the space of tree occurrences, the probability of a tree T is defined as the sum of the probabilities of its distinct derivations. Let t_{id} be the i -th subtree in the derivation d that produces tree T , then the probability of T is given by:

$$P(T) = \sum_d \prod_i P(t_{id}) \quad (3)$$

The best parse tree T_{best} maximizes the probability of T_{sd} given input string W :

$$T_{best} = \operatorname{argmax}_{T_{sd}} P(T_{sd} | W) \quad (4)$$

It should be emphasized that the DOP+ model deals exclusively with tree structures. Several richer models, based on more sophisticated linguistic representations, have been proposed, ranging from (Lexical-Functional Grammar) LFG-annotated corpora consisting of trees enriched with attribute-value matrices [22] to (Head-driven Phrase Structure Grammar) HPSG-annotated databases consisting of feature structures [79] and TAG-based DOP models that allow for richer combination operations [58]. But since these richer DOP models all use trees as their backbone, it is convenient to base our exposition of DOP on the use of trees. Moreover, since all international benchmarks for NLP (and other modalities) currently consist of phrase-structure trees, we will stick to tree-based DOP models for the scope of this review and refer to [24] for a general overview of linguistic DOP models.

Yet, even for tree-based DOP models there exist many different versions. For example, DOP+ uses a simple relative frequency estimator for assigning

weights to the subtrees. While this estimator obtains good results on the **Penn Treebank**, it does not maximize the likelihood of the training set (see [61]). In [18], a DOP model was developed which does maximize the likelihood of the corpus by using the relative frequency estimator only as an initial parameter which is next re-estimated by the well-known Expectation-Maximization (EM) algorithm [43]. This model, called ML-DOP, uses cross-validation to avoid overtraining. While ML-DOP does not improve over previous DOP models, the maximum likelihood estimator in ML-DOP does boost unsupervised versions of DOP. We will go into these unsupervised extensions in Sect. 9. For an overview of the various statistical estimators used in DOP and many probabilistic grammars, we refer the reader to [95] or [96].

3 A DOP Model for Music

It is rather straightforward to apply the DOP approach to melodic analysis of musical pieces. As in natural language, a listener segments a sequence of notes into groups or phrases that form a grouping structure for the whole piece [69]. For example, according to [68: 37], a listener hears the grouping structure in Fig. 4 for the first few bars of melody in the Mozart G Minor Symphony, K. 550.

Each group is represented by a slur beneath the musical notation. A slur enclosed within a slur means that a group is heard as part of a larger group. This hierarchical structure of melody can, without loss of generality, also be represented by a phrase structure tree, as illustrated in Fig. 5.

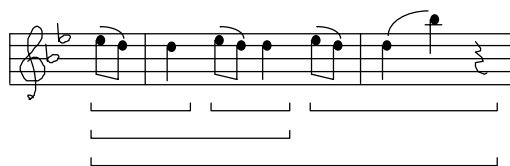


Fig. 4. Grouping structure for the opening theme of Mozart's G minor symphony

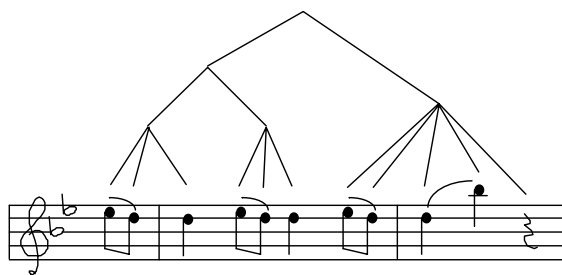


Fig. 5. Tree structure for the grouping structure in Fig. 4

Note the analogy with phrase structure trees in linguistics: a tree describes how parts of the input combine into constituents, how these constituents combine into larger constituents, and so on into a representation for the whole input. Apart from this analogy, there is also a difference: while the nodes in a linguistic tree structure are typically labeled with syntactic categories such as S, NP, VP and the like, musical tree structures are usually unlabeled. This is because in language there are syntactic constraints on how words can be combined into larger constituents (for instance, in English a determiner can be combined with a noun only if it precedes that noun), while in music there are no such restrictions: in principle any note may be combined with any other note. This makes the problem of ambiguity in music much harder than in language. [70] note that “Any given sequence of note values is in principle infinitely ambiguous, but this ambiguity is seldom apparent to the listener.” For example, the first few bars of Mozart’s G Minor Symphony could also be assigned the alternative grouping structure in Fig. 6 (among the many other possible structures).

While this alternative structure is possible in that it can be perceived, it does not correspond to the structure that is actually perceived by a human listener. As in natural language, there is thus an important question as to how to select the perceived tree structure from the set of possible tree structures of a musical input. Many systems attempt to disambiguate melodic structure in an entirely rule-based way. For example, [68] and [86] use preference rules that describe Gestalt-perceptions of the kind identified by [91]. However, similar to language, there are extremely many ‘multi-note units’ and other idiomatic phrases and musical clichés that melodic analysis has to deal with (see [12, 13]). Most parsing approaches to melodic analysis are nowadays probabilistic and exemplar-based (for instance, [12, 41, 47]). These approaches are trained on corpora such as the **Essen Folksong Collection** (EFC) which contains musical trees of over 6,000 folk songs [82].

Since the encoding of note sequences is not as straightforward as in natural language, let us briefly explain how the folk songs in the EFC are annotated (see [82] for the full annotation scheme). The Essen folk songs are represented

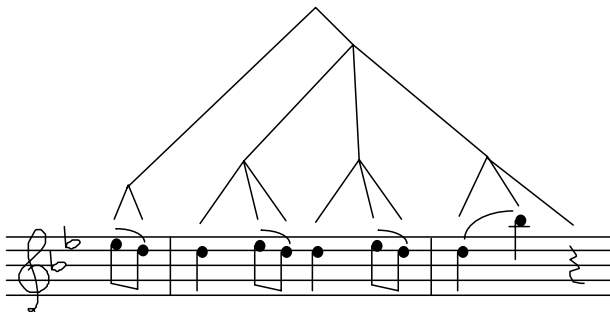


Fig. 6. Alternative grouping structure for Mozart’s opening theme

by the so-called Essen Associative Code (ESAC). The pitch encodings in ESAC resemble ‘solfege’: scale degree numbers are used to replace the movable syllables ‘do’, ‘re’, ‘mi’, and so forth. Thus 1 corresponds to ‘do’, 2 corresponds to ‘re’, and so on. Chromatic alterations are represented by adding either a ‘#’ or a ‘b’ after the number. The plus (+) and minus (−) signs are added before the number if a note falls respectively above or below the principle octave (thus −1, 1 and +1 all refer to ‘do’, but on different octaves). Duration is represented by adding a period or an underscore after the number. A period (‘.’) increases duration by 50% and an underscore (‘_’) increases duration by 100%; more than one underscore may be added after each number. If a number has no duration indicator, its duration corresponds to the smallest value. A pause is represented by ‘0’, possibly followed by duration indicators, and is also treated as an atomic symbol. No loudness or timbre indicators are used in ESAC. Phrase boundaries are annotated by hard returns in ESAC, which we automatically convert into bracket representations where ‘(’ indicates the start and ‘)’ the end of a phrase. These phrase boundaries were manually assigned on the basis of the pitch encodings only (the lyrics were not taken into account – see [82]). The phrases in the EFC are unlabeled.

However, to use the DOP approach for parsing the EFC, we first need to (automatically) add three basic labels to the phrase structures: *S* for the whole song, *P* for each phrase and *N* for each note. In this way, we obtain conventional tree structures that can directly be used by DOP+. To illustrate this, consider the simple corpus in Fig. 7 of two musical tree structures.

This corpus contains two very simple melodies, the first consisting of two phrases, (1 2) (2 3), and the second consisting of only one phrase, (1 2 3 1). If we take this corpus as our training set, then a new melody, such as ‘1 2 1 2’, can be parsed by DOP+ by combining subtrees from this corpus, again by means of the substitution operation \circ .

But this melody can also be parsed in a different way, resulting in a different parse tree, for example by combining the following subtrees from Fig. 7.

Remember that this free combination of subtrees only defines the set of possible structures of an input. To predict the best musical tree structure as

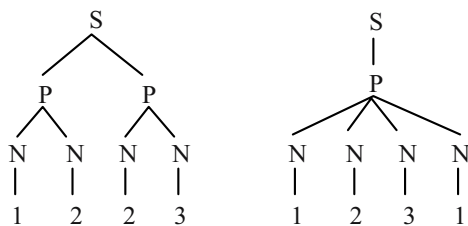


Fig. 7. A simple musical corpus

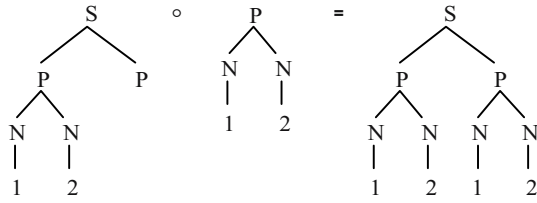


Fig. 8. Parsing a new melody by combining subtrees from Fig. 6

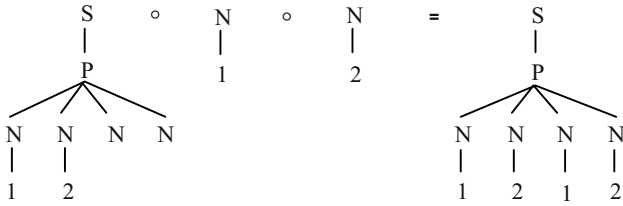


Fig. 9. Generating a different tree structure for the same melody

assigned by humans, DOP+ selects the resulting tree structure in Fig. 8, as it corresponds to the shortest derivation, and thereby recognizes that the phrase (1 2) can be parsed as one (previously observed) pattern. Had we restricted the subtrees to the smallest ones (which would lead to a probabilistic context-free grammar), the resulting tree in Fig. 9 would have corresponded to the shortest derivation since it only consists of 6 rules (or depth-1 subtrees), while the tree in Fig. 8 would have needed 7 rules. Thus large subtrees are important, and should not be restricted if we want to maximize similarity. In case there is more than one shortest derivation, the most probable tree is selected, just as with language (Definition 3).

Of course this musical example is exceedingly simple. The *Essen Folksong Collection* provides a much more challenging set of melodies, and will be used for our experiments in Sect. 8. For an in-depth discussion of the variety and complexity of the melodies in the *Essen Folksong Collection*, see [59].

4 A DOP Model for Problem Solving in Physics

What counts for syntactic and melodic analysis also counts for problem solving and reasoning: given a problem or theorem, there can be (extremely) many different possible solutions or derivations. As in language and music, a major challenge is to select the ‘best’ derivation among the many possibilities. As a case study, we will concentrate on derivations for physics problems.

Let us first discuss what problem solutions or ‘derivational explanations’ in physics look like. Physics textbooks provide many examples of problem solutions which are typically used to solve new problems. Although textbook

problem solutions may not reflect scientific practice, they are part of the training of every scientist and highly influence their reasoning. We will start with a simple, idealized example. In their Physics textbook, Alonso and Finn derive the Earth’s mass from the Earth-Moon system as follows [2: 247]:

Suppose that a satellite of mass m describes, with a period P , a circular orbit of radius r around a planet of mass M . The force of attraction between the planet and the satellite is $F = \frac{GMm}{r^2}$. This force must be equal to m times the centripetal acceleration $v^2/r = 4\pi^2r/P^2$ of the satellite. Thus,

$$4\pi^2mr/P^2 = GMm/r^2$$

Canceling the common factor m and solving for M gives

$$M = 4\pi^2r^3/GP^2$$

This rather textual derivation can be represented by means of the proof tree or derivation tree of Fig. 10. Proof trees are widely used data structures in automated reasoning and theorem proving [3] and form the main representations in Explanation-Based Learning [76], Inductive Logic Programming [42] and Statistical Relational Learning [77].

Thus the derivation tree in Fig. 10 represents the various steps from general laws to an equation for the mass M . In general, a derivation tree is a labeled tree where each node is annotated with a formula (the boxes are only convenient representations that have no additional meaning). The formulas at the top of each ‘vee’ (in other words, each pair of connected branches) in the

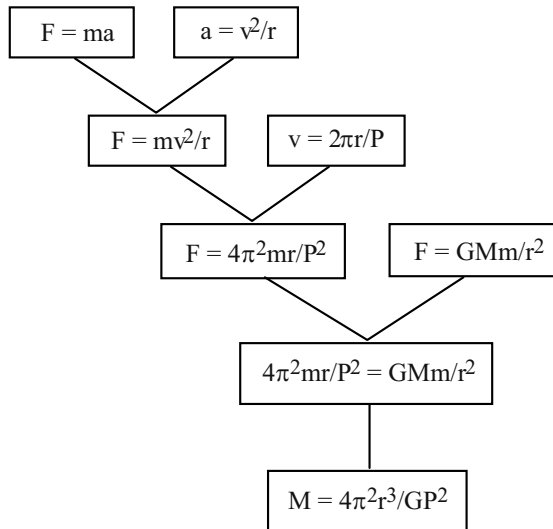


Fig. 10. Derivation tree for the derivation of the Earth’s mass

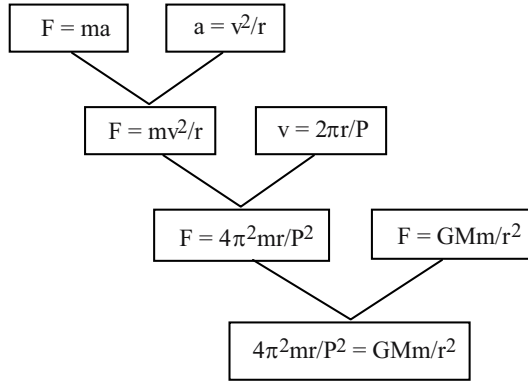


Fig. 11. A subtree from the tree in Fig. 10

tree can be viewed as premises, and the formula at the bottom of each ‘vee’ can be viewed as a conclusion which is arrived at by simple term substitution. The last derivation step in the tree is not formed by a ‘vee’ but consists of a unary branch which solves the directly preceding formula for a certain variable (in the tree above, for the mass M). Unary branches may also appear at other places in a derivation tree. In general, a unary branch refers to a mathematical derivation step, while a binary branch refers to a combination of laws (or conditions) by means of simple term substitution. Note that derivation trees are conventionally represented in an ‘upside-down’ manner with respect to phrase-structure trees in language and music (see [3]).

Suppose that the derivation tree in Fig. 10 constitutes our training corpus, then the regularity known as Kepler’s third law, which states that r^3/P^2 is constant, can be easily derived by using the subtree in Fig. 11 which is extracted from the tree in Fig. 10. The root of this subtree only needs to be solved for r^3/P^2 , which is accomplished by a mathematical derivational step added in Fig. 12.

We can thus already note a difference between derivations in language and music and derivations in physics: for the latter we need an additional mathematical component that can solve equations.

Of course, it is not the typical case that we can derive a new phenomenon by just one subtree. Often we need to combine several smaller subtrees, as is the case for instance in deriving the velocity of a satellite at a certain distance from a planet. This is accomplished by using the following two subtrees in Fig. 13 from the tree in Fig. 10, that are first combined by term substitution (represented by the operation ‘o’)¹ and then solved for the velocity v .

¹ As long as no confusion arises we will use the same symbol for label substitution in language and music and term substitution in derivational problem solving.

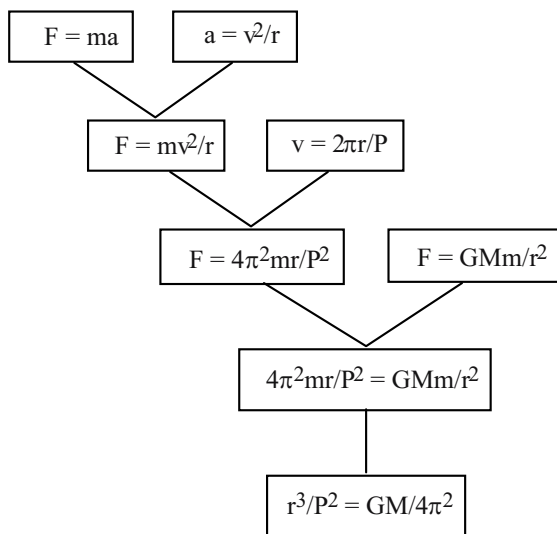


Fig. 12. Deriving Kepler's Law by the subtree in Fig. 11

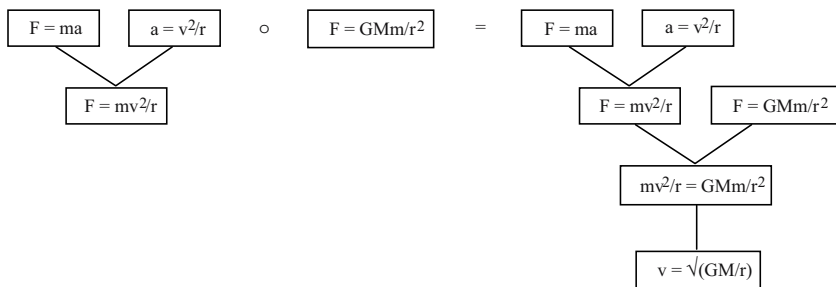


Fig. 13. Deriving the velocity of a satellite by combining two subtrees from Fig. 10

Note the analogy with linguistic and musical processing: new input can be derived by combining subtrees from previously derived input, where the subtrees may be of any size – from single laws to entire derivation trees. But there are also some differences. Apart from the additional mathematical component, there is a difference in combining subtrees: while in music and language the ‘combination operation’ between subtrees consists of simple (left-most) label substitution, the term ‘substitution operation’ in problem solving also expands the tree with a new root node (see Fig. 13). We will therefore specify this operation more explicitly as follows:

The term ‘substitution operation \circ ’ is a partial function on pairs of labeled trees, and its range is the set of labeled trees. The combination of tree t and tree u , written as $t \circ u$, is defined iff the equation at the root node of u can be substituted in the equation at the root node of t (that is, iff the lefthand side of the equation at the root node of u literally appears in the equation at

the root node of t). If $t \circ u$ is defined, it yields a tree that expands the root nodes of copies of t and u to a new root node where the righthand side of the equation at the root node of u is substituted in the equation at the root node of t . Note that the substitution operation can be iteratively applied to a sequence of trees, with the convention that \circ is left-associative.

The idea that new problems and phenomena can be solved by reusing parts of previous problem solutions is reminiscent of the notion of ‘exemplar’ in Thomas Kuhn’s account of normal science. According to [67], exemplars are “problem solutions that students encounter from the start of their scientific education”, and “Scientists solve puzzles by modeling them on previous puzzle-solutions” [67: 187–190]. In the following, we will use the terms ‘exemplary problem solution’ and ‘exemplar’ interchangeably. Our approach to problem solving is also congenial to Case-Based Reasoning (for example, [28]), where new problems are solved by modeling previous problems.

As noted above, there is a problem with derivational reasoning and problem solving which is analogous to linguistic and musical analysis: ambiguity. To illustrate this, it is convenient to enlarge our training corpus in Fig. 10 with one other example from Alonso and Finn’s textbook. This example again provides an exemplary problem solution for the Earth’s mass but this time using an alternative derivation (2: 246). Both solutions are used as exemplars on which other problems are modeled. This second exemplar computes the Earth’s mass from the acceleration of an object near the Earth’s surface and which, following the derivation steps in [2], can be represented by the derivation tree in Fig. 14 (where for the sake of conciseness the initial conditions $a = g$ and $r = R$ are represented by one label).

By substituting the values for g (the acceleration at the Earth’s surface), R (the Earth’s radius) and G (the gravitational constant), [2] obtain roughly the same value for the Earth’s mass as in the previous derivation in Fig. 10. They argue that this agreement is ‘a proof of the consistency of the theory’ [2: 247]. When we add this exemplar to our corpus, we get many different derivations for new phenomena or problems. For example, Kepler’s regularity can now

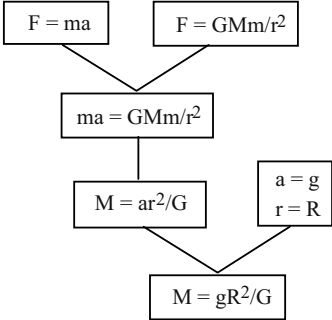


Fig. 14. An additional exemplar in the training corpus

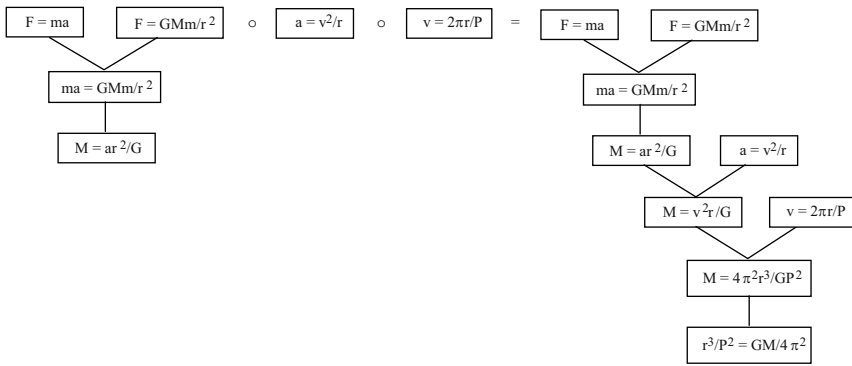


Fig. 15. An alternative derivation of Kepler’s regularity

also be derived by the following alternative derivation in Fig. 15, which uses a large subtree from Fig. 13 in combination with two small subtrees from Fig. 10.

There is nothing wrong with this alternative derivation: there are no spurious non-explanatory laws that are irrelevant to this derivation (as would be Hooke’s or Boyle’s law). The only difference is that the derivation in Fig. 15 is modeled on a different exemplar (that is, on a planet-particle model at rest) than the derivation in Fig. 12 (which is modeled on an orbiting planet-satellite exemplar). In fact, the alternative derivation in Fig. 15 is insightful as it expresses the conceptual equivalence between terrestrial and celestial mechanics in Newtonian dynamics. However, problem-solving experiments with physics students show that humans don’t come up with this alternative derivation (Sect. 6). Unfortunately, the ambiguity problem is much worse: by combining subtrees from the two exemplars in Figs. 10 and 14 in different ways, we get a combinatorial explosion of possible derivation trees of Kepler’s law.

5 Towards a Unifying Approach

As with linguistic and musical analysis, also for problem-solving we hypothesize that the best tree is the one that can be constructed by the shortest derivation and that in case there are more shortest derivations the most probable tree should be selected among them. For example, the derivation tree of Kepler’s regularity in Fig. 12 corresponds to the shortest derivation since it can be constructed by just one large subtree from an exemplar (modulo the mathematical derivation step), while the derivation tree for the same regularity in Fig. 15 needs at least three subtrees to be constructed. As a consequence, the tree in Fig. 12 is more structurally similar to an exemplar in the training corpus than is the tree in Fig. 15.

However, the use of the shortest derivation alone is not enough. It may occur that a phenomenon can be derived by two or more shortest derivations containing the same number of subtrees (and even the same number of labels)

but resulting in different derivation trees. In such a case we also take into account the relative frequencies of the subtrees in a representative corpus of exemplars, and compute the most probable tree from among the trees generated by the shortest derivations. A higher subtree frequency expresses a wider usability of the derivational pattern for deriving phenomena. (We already argued in Sect. 4 why the reverse order of first computing the most probable tree, and next selecting the shortest derivation, is problematic for sparse corpora with highly specific labels like problem solving. We will support this argument with computational experiments in Sect. 8.)

It should be noted that previous statistical approaches to reasoning and problem solving were mainly based on stochastic enrichments of context-free grammars (CFGs) or definite-clause grammars (DCUs). These approaches, known as Stochastic Logic Programs or Statistical Relational Learning [42, 44, 78] cannot cover all possible dependencies in a derivation tree. We have shown in [19] that, as with language and music, there may be arbitrarily distant dependencies in problem solving, both structurally and sequentially. The examples we discussed in [19] came from the field of fluid mechanics. But distant dependencies already occur in derivations for much simpler systems, such as Galileo's pendulum (between leaf nodes and formulas later in the derivation tree). Entire subtrees must be preserved, otherwise they lose the particular dependency. It is well-known that students of physics typically have to go through various example-derivations before they can successfully solve new problems by themselves, usually by modeling the new problem on similar, previously solved problems [51, 67].

Thus it appears that DOP+ can also be employed for (equational) reasoning and problem solving. We only need to slightly modify the DOP+ definitions in Sect. 4. That is, we need to change 'label substitution' into 'term substitution' (which we already defined in Sect. 4). Second, the root node of a derivation tree must correspond to a mathematical description of a phenomenon, and the leaf nodes must be laws, conditions or any knowledge that cannot be derived from other equations (such as empirical corrections and normalizations). This brings us to Definition 4 for a DOP+ model of problem solving and reasoning.

Definition 4 *Given a corpus C of trees T_1, T_2, \dots, T_n representing derivations of phenomena, and a term substitution operation \circ , a derivation tree of a phenomenon P with respect to C is a tree T such that (i) there are subtrees t_1, t_2, \dots, t_k in T_1, T_2, \dots, T_n for which $t_1 \circ t_2 \circ \dots \circ t_k = T$, (ii) the root of T is mathematically equivalent to P and (iii) the yield of T consists of either laws or antecedent conditions or any other equations that cannot be derived from higher-level equations.*

Definitions 2 and 3 in Sect. 4 – for the tree generated by the shortest derivation T_{sd} and the best tree T_{best} , respectively – remain the same (provided that we substitute the word string W by the phenomenon P in the definition of T_{best}). Given this commonality between problem solving and perceptual (linguistic and musical) processing, DOP+ may be a viable candidate for a general model of these modalities.

We can also try to further integrate Definitions 1 and 4 by referring to trees of natural phenomena, word strings and musical pieces as ‘exemplars’, and by referring to analysis trees and derivation trees as ‘DOP+ generated trees’. But we then need to abstract from the differences between label substitution in language and music and term substitution in problem solving, for example, by viewing label substitution as a special case of term substitution (in case the entire substitutable labels are exactly equivalent no node expansion is created). This results in Definition 5, where we use a generalized notion of ‘substitution operation’ which is left unspecified.

Definition 5 *Given a corpus C of trees T_1, T_2, \dots, T_n representing exemplars and a substitution operation \circ , a DOP+ generated tree with respect to C is a tree T such that there are subtrees t_1, t_2, \dots, t_k in T_1, T_2, \dots, T_n for which $t_1 \circ t_2 \circ \dots \circ t_k = T$. T is said to be a derivation tree of a phenomenon P iff the root of T is mathematically equivalent to P and the yield of T cannot be further derived. T is said to be an analysis tree of an input string W iff the root of T is equal to the distinguished symbol S and the yield of T is equal to W .*

Thus while there remain differences between problem solving in physics on the one hand and syntactic and melodic analysis on the other hand, there appears to be a common level of representation and computation. This is the level of tree structures (the common representation) that are decomposed and recomposed to analyze new input in the shortest and most probable way (the common computation). The labeling of the trees and details of the combination operation differ across the modalities, but the formula for the best tree of an input I is the same for all modalities: $T_{best} = \operatorname{argmax}_{T_{sd}} P(T_{sd} | I)$.

6 Test Corpora for DOP+

While annotated corpora are widely available for language [1, 74] and music [82], corpora of tree structures for physics problems are still very rare. Previous work that deals with students’ problem solutions does not formalize these solutions by means of trees [71, 88]. In [16, 20] we therefore developed a corpus

of physics problems whose solutions were directly converted to tree structures by students. The following briefly describes the construction of this ‘physics corpus’.

A total of 19 third-year physics students from the University of Amsterdam (academic year 2005–2006) were paid to construct both a test corpus and a training corpus. 10 students were involved in constructing the test corpus while the remaining 9 students constructed the training corpus. As to the test corpus, the 10 students were asked to solve 14 elementary problems from classical mechanics and 10 elementary problems from fluid mechanics. The students had previously followed courses in classical mechanics and more recently an extensive course in fluid mechanics. The 24 problems given to them consisted in deriving a phenomenon from law and initial conditions. Four of these problems are given below (for more details, see [20]):

Problem nr. 1

Show that the period of the Earth’s rotation for which an object at the equator would become weightless is given by $P = 2\pi\sqrt{R/g}$, where R is the Earth’s radius and g is the gravitational acceleration at the Earth’s surface.

Problem nr. 2

Show that the theoretical velocity which an object attains in free fall from height h is given by $v = \sqrt{2gh}$ where g is the gravitational acceleration at the Earth’s surface.

⋮

Problem nr. 23

When water flows through a right-angled V-notch, show that the discharge is given by $Q = KH^{5/2}$ in which K is a constant and H is the height of the surface of the water above the bottom of the notch.

Problem nr. 24

Show that the theoretical rate of flow through a rectangular notch is given by $Q = (2/3)B\sqrt{(2g)}H^{3/2}$, where B is the width of the notch and H is the height of the water level above the bottom of the notch.

After the students had solved the problems on paper, they were given a short, ten-minute tutorial on the concept of derivation tree, especially on the difference between binary branches in a tree (used for combining laws, conditions and similar), and unary branches (used for mathematical derivation steps of which the exact operations could be left implicit). The students were told that the exact order of the laws in a tree was not important as long as these laws could be properly combined by term substitution to solve the

problem. After this brief tutorial, the students were asked to draw derivation trees for their problem solutions.

There was a high agreement among the derivation trees constructed by the students: on average 95.4% (SD = 1.5) of the derivation trees per problem matched (modulo law order). In creating a gold standard, only the most voted tree was put in the corpus. In our case, the 24 most voted (that is, most frequently created) derivation trees for each problem constituted the test corpus.

As to the construction of the training corpus, the remaining 9 students were asked – after the same brief tutorial – to draw derivation trees for 33 problem solutions from classical and fluid mechanics that are used as exemplars in the textbooks by [2: Chaps. 9–11] and [45: Chap. 7]. The three examples in Figs. 9, 12 and 13 were among these exemplary solutions. The agreement among the constructed derivation trees for the exemplary solutions was very high: 98.0% (SD = 0.6). The most voted tree for each exemplary solution was put in the training corpus.

All 24 test problems could be solved by subtrees from the training corpus of 33 exemplars but this fact was not told to any of the students. Our total corpus of problem solutions thus consists of 57 trees including a training set of only 33 exemplars. This stands in strong contrast with the considerably larger linguistic and musical corpora. However, a representative corpus for elementary classical and fluid mechanics is necessarily much smaller than a representative corpus for language or music. It has for example been estimated by [51: 88] that the number of exemplars available to a physics expert lies around a few hundred. For undergraduates, who have only knowledge of elementary physics, this number is of course much lower, and the 33 exemplary problem solutions from [2] and [45] cover the typical exemplars from mechanics learned by undergraduate physics students (in other words, the planet-satellite model, the frictionless plane, the harmonic oscillator, the vena contracta, and the like). Thus our physics corpus is likely to correspond to the full set of exemplars learned by physics students in their curriculum.

7 Computing T_{best}

Before we can test DOP+ on the various corpora, we need to go into the problem of computing T_{best} for a given input. The computation of T_{best} is especially challenging for language and music where our corpora contain thousands of trees which correspond to millions of subtrees. We will therefore first go into linguistic and musical parsing and next come back to problem solving. The way DOP+ combines subtrees into new trees is formally equivalent to a Tree-Substitution Grammar or TSG [8]. Moreover, the way DOP+ defines the

best tree is covered by the notion of a Stochastic TSG or STSG. There are standard algorithms that compute the tree structures (a packed parse forest) of an input string given an STSG. These algorithms run in Gn^3 time, where G is the size of the grammar (the number of subtrees) and n is the length of the input string (the number of words or notes).

Existing parsing algorithms for context-free grammars or CFGs, such as the CKY algorithm [94], can be easily extended to TSGs by converting each subtree t into a context-free rewrite rule where the *root* of t is rewritten by its yield: $root(t) \rightarrow yield(t)$. Indices are used to link each rule to its original subtree. Next, T_{best} can be computed by a best-first beam search technique known as Viterbi optimization [73]. However, the direct application of these techniques to DOP(+) is infeasible mainly because the number of subtrees usually grows exponentially with the corpus size [83]. The relatively small **Air Travel Information System** (ATIS) corpus of 750 trees [74] contains over 40,000 subtrees, and the **Wall Street Journal** (WSJ) corpus of 50,000 trees contains more than 100 million subtrees.

To make parsing with DOP feasible, several heuristics have been proposed, ranging from randomly sampling subtrees [8] to restricting the subtrees on linguistic grounds [84]. DOP's ideal to parse with all, arbitrarily large subtrees might have died an early death as being computationally prohibitive, if it were not for an insight by [53, 54] that the unwieldy DOP grammar can be reduced to a set of eight Probabilistic Context-Free Grammars (PCFGs) which is linear rather than exponential in the number of nodes in the corpus.² Goodman's PCFG reduction was initially developed for the probabilistic version of DOP but it can also be applied to computing the shortest derivation. The following briefly summarizes the method.

The key idea is to re-label the nodes in the corpus trees. Every node in every tree is assigned a unique number which is called its 'address'. The notation $A@k$ denotes the node at address k where A is the nonterminal labeling of that node. A new nonterminal is created for each node in the training data. This nonterminal is called A_k . Nonterminals of this form are called 'interior', while the original nonterminals in the parse trees are called 'exterior'. Let a_j represent the number of subtrees headed by the node $A@j$. Let a represent the number of subtrees headed by nodes with nonterminal A , that is $a = \sum_j a_j$.

Goodman shows that there is a PCFG with the following property: for every subtree in the training corpus headed by A , the grammar will generate an isomorphic subderivation with probability $1/a$ [53]. The construction is as follows. For a node $(A@j(B@k, C@l))$, the following eight PCFG rules are generated, where the number in parentheses following a rule is its probability:

² [40] have used kernel methods to develop an efficient parsing algorithm for an all-subtrees representation.

$$\begin{array}{llll}
A_j & \rightarrow & BC(1/a_j) & A & \rightarrow & BC(1/a) \\
A_j & \rightarrow & B_k C(b_k/a_j) & A & \rightarrow & B_k C(b_k/a) \\
A_j & \rightarrow & BC_l(c_l/a_j) & A & \rightarrow & BC_l(c_l/a) \\
A_j & \rightarrow & B_k C_l(b_k c_l/a_j) & A & \rightarrow & B_k C_l(b_k c_l/a)
\end{array} \tag{5}$$

Goodman next shows by simple induction that subderivations headed by A with external nonterminals at the roots and leaves; internal nonterminals elsewhere have probability $1/a$ [53]. Further, subderivations headed by A_j with external nonterminals only at the leaves, and internal nonterminals elsewhere, have probability $1/a_j$. This can be easily demonstrated by multiplying the relevant probabilities of the rules, which brings Goodman to his main theorem, that his construction produces PCFG derivations isomorphic to DOP derivations with equal probability [53: 130–133].

Note that the PCFG reduction can also be used to compute the shortest derivation, since the most probable derivation is equal to the shortest derivation if each subtree is given equal probability. This can be seen as follows. Suppose we give each subtree a probability p , then the probability of a derivation involving n subtrees is equal to p^n , and since $0 < p < 1$, the derivation with the fewest subtrees has the greatest probability. For all our experiments with linguistic and musical corpora, we employ Goodman’s reduction in combination with a best-first CKY parsing algorithm [94] that computes the most probable parse tree from among the shortest derivations.

Let us now turn to computing T_{best} for problem solving. In practice the computation of T_{best} for a mathematical description of a phenomenon is less hard, simply because a corpus of exemplary problem solutions tends to be much smaller than corpora for language and music (as discussed in the previous section). The training set of our problem solving corpus contains only 33 trees, which correspond to 408 different subtrees. Although this number of subtrees is computationally tractable, the root of each subtree may be extended with mathematical derivation steps at any point in a derivation (as we have seen in Sect. 4). Thus there can be no *a priori* reduction of a problem-solving corpus into a compact PCFG, because we do not know beforehand which mathematical operations are needed at the subtree-roots.

In principle we could generate all possible mathematical extensions for all subtrees (in other words, all solutions for possible variables in the equations at the subtree-roots). But this would lead to a combinatorial explosion of possible subtree extensions. Fortunately, there are standard equational reasoning systems that can efficiently solve an equation given a set of other equations, such as **TK Solver** (<http://www.uts.com/>). In our experiments below, we first convert each derivation tree from the training corpus into its subtrees. Next, we extract the equations from the subtree-roots, which are indexed to remember the subtrees they are extracted from. This results in a list of 408 equations. For each test problem (namely, the equation to be solved), we use

TK Solver to derive a set of solutions given the list of 408 equations. It turns out that virtually all problems receive more than 60 different solutions, even *after* abstracting from the order of the equations used in the solution, which gives an idea of the ambiguity if we do not have any mechanism to break ties.

From the output of TK Solver we select the shortest solution(s) for each problem that use(s) the fewest equations. Next, the equations of the shortest solution(s) are converted back to their corresponding subtrees, which are combined into the tree corresponding to the shortest derivation, T_{sd} . In case T_{sd} is not unique we compute the probability for each T_{sd} and select the most probable tree, which yields T_{best} .

8 Experiments with DOP+

For language, we used the now standard division of the **Wall Street Journal** (WSJ) corpus in the **Penn Treebank**, of which Sects. 2 through 21 are used as training set (approximately 40,000 sentences), and of which Sect. 23 is used as test set (2,416 sentences \leq 100 words). As in other experiments with the WSJ, all trees were stripped of their semantic tags, co-reference information and quotation marks (see [73]). In case a word from a test sentence was unknown in the training set, we employed the unknown word model in [11], based on statistics on word-endings, hyphenation and capitalization. For music, we used the same (random) division of the **Essen Folksong Collection** (EFC) as in [13] into a training set of 5,251 trees and a test set of 1,000 trees. There were no unknown notes for this division. As explained in Sect. 3, the root of each EFC tree was labeled with the distinguished symbol ‘S’, the notes were labeled with ‘N’ and the internal nodes with ‘P’. For problem solving, we used the training set of 33 exemplary problem solutions and the test set of 24 problems, as described in Sect. 7.

The training set trees for each modality are used to extract the subtrees employed by DOP+, while the test data without the trees are used as input. The best trees predicted by DOP+ are compared with the trees in the respective test sets. The degree to which these best trees match the test set trees is a measure for the accuracy of the system. An evaluation metric which has become standard in the field of NLP, and which is also used in the field of music analysis, is the PARSEVAL metric of precision and recall [4]. This metric compares a so-called ‘proposed’ parse tree P (that is, our T_{best}) with the corresponding correct test set parse tree T as follows:

$$Precision = \frac{\# \text{ correct constituents in } P}{\# \text{ constituents in } P} \quad (6)$$

$$Recall = \frac{\# \text{ correct constituents in } P}{\# \text{ constituents in } T} \quad (7)$$

A constituent in P is said to be ‘correct’ if there exists a constituent in T of the same label that spans the same sequence of leaves. Since precision and recall can obtain rather different results (see [13]), they are typically balanced by a single measure of performance, known as the F -score:

$$Fscore = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (8)$$

We will use these definitions for evaluating T_{best} in language and music. However, they cannot be directly applied to evaluating T_{best} in problem solving. This is because the exact sequence of leaves is irrelevant here. While in language and music the sequence of leaves of a tree constitutes respectively the sentence and the musical piece, the leaves of a problem-solving tree constitute the laws and conditions in a derivation. Also, it does not matter whether we put a law as a premise at a left daughter node or at a right daughter node, as long as their combination results in the same conclusion (which we also explained to the students in creating the problem solving corpus – see Sect. 6). Thus we can only reasonably apply the metrics above to problem solving if we substitute ‘same sequence of leaves’ by ‘same leaves’ in the definition of ‘correct’ constituent above, such that we abstract from the order of the leaves.

It is one of the most essential features of DOP+ that arbitrarily large subtrees are taken into consideration. To test the usefulness of this feature, we performed a number of experiments where we restricted the training set subtrees to a certain maximum size. We define the size of a subtree by its depth, which is the length of a subtree’s longest path from root to leaf. In this way we can test a range of other models; for example, by restricting the maximum depth of the subtrees to one, DOP+ is equivalent to a stochastic context-free grammar. As pointed out in [54: 134], Goodman’s reduction method can still be applied to DOP when the training set subtrees are constrained to a certain depth. The following Table shows the results of our experiments where we give for increasing subtree depths the F -scores (in percentages) for respectively language (the WSJ corpus), music (the EFC corpus) and problem solving (the problem-solving corpus in Sect. 6). The maximum tree depth in the **Essen Folksong Collection** is 3, while the maximum tree depth in the problem solving corpus is 6.

Table 1 shows that there is a consistent increase in accuracy with increasing subtree depth for all modalities. Note that the maximum tree depth in the **Essen Folk song Collection** is 3, while the maximum tree depth in the Problem Solving corpus is 6. We have previously observed this phenomenon for language in [8, 11] where we called it ‘the DOP hypothesis’. This hypothesis has been corroborated for Dutch and English [8, 11, 85], for Chinese [55] and for Hebrew [85]. Moreover, the DOP hypothesis has been tested not only for tree-annotations but also for LFG-annotations [25], HPSG-annotations [79] and TAG-analyses [58]. Thus the hypothesis is robust, and seems to be

Table 1. F-scores of DOP+ for different subtree depths

Max. Subtree Depth	Language (<i>Wall St. Journal</i>)	Music (<i>Essen Folksongs</i>)	Problem-Solving (<i>Physics Corpus</i>)
1	68.5	58.4	45.8
2	77.2	77.3	62.5
3	80.9	88.9	75.0
4	82.1		83.3
6	86.0		87.5
8	88.2		
10	88.8		
unrestricted	91.1		

independent of the nature of the annotations. Table 1 shows that the DOP hypothesis also seems to hold for music and problem solving.

Our results are very competitive compared to other parsers for language and music. For the WSJ, DOP+ outperforms stochastic lexicalized grammars, such as in [36, 37] and [29, 30] – see [14] for a detailed quantitative comparison. Yet, there is more recent work which outperforms the DOP+ model, in particular [75], who extend their parser with discriminative self-training, achieving a 92.1% F -score on the same standard WSJ split, which is a 1% improvement over DOP+. It would be interesting to see how DOP+ performs if extended with self-training. Our scores on the Essen folk songs are higher than those reported by [86: 74], but unfortunately the results are not exactly comparable, since Temperley uses a smaller test set of only 65 folk songs [86].

There is an important question as to how other proposals for a unified DOP model perform. For this Chapter, we therefore accomplished an additional series of experiments with an alternative unifying DOP model which first computes the most probable tree and next selects the shortest derivation in case the most probable tree is not unique. Table 2 shows the results of these experiments for different subtree depths using the same training/test set divisions as for Table 1.

We again note that there is an increase in accuracy with increasing subtree size, but this time the best F -scores are considerably lower than in Table 1. For language and music the differences are a few percent only, but for problem solving the difference is nearly 40%: while DOP+ predicts for 21 out of 24 problems the correct derivation tree, the alternative DOP model only gets 12 out of 24 correct. Thus by first computing the most probable tree instead of the shortest derivation, the best score of the alternative DOP model is even worse than DOP+'s score at subtree-depth 2 for problem solving. We already explained why this may be the case: the most probable tree is a bad metric for small corpora, especially if such corpora have very specific labels. The shortest derivation, on the other hand, is a good metric in almost all cases, and by

Table 2. F-scores of an alternative DOP model for different subtree depths

Max. Subtree Depth	Language (<i>Wall St. Journal</i>)	Music (<i>Essen Folksongs</i>)	Problem-Solving (<i>Physics Corpus</i>)
1	70.4	62.7	20.8
2	78.3	76.9	25.0
3	80.1	86.5	37.5
4	82.6		45.8
6	84.4		50.0
8	85.6		
10	86.3		
unrestricted	88.7		

selecting among a few remaining shortest derivations (in case the shortest derivation is not unique), the differences in frequency apparently do work out well, also for the small problem-solving corpus.

To check whether these differences are statistically significant, we performed a series of experiments using a 10-fold division into random training and test sets for language and music with unrestricted subtree depth only. It turns out that the differences in the best accuracies between DOP+ and the alternative unifying DOP model are statistically significant, both for language ($p \leq 0.05$) and music ($p \leq 0.02$) using paired t -testing. We did not test on different training/test set divisions of the problem solving corpus, since the training set already consists of the *actual* exemplars used in the textbooks on classical and fluid dynamics. Moreover, these exemplars closely correspond to those in other textbooks (see [51] for a comparison between physics textbooks).

What should we learn from these experiments? While we have already qualitatively explained why large subtrees are important for language (Sect. 2), music (Sect. 3) and problem-solving (Sect. 4), our experiments show that this can also be quantitatively supported. Our results show that directly applying statistical computations is inferior to first computing the space of ‘most similar’ trees by means of the shortest derivations. The best model first maximizes similarity and next probability. The maximization of similarity may be reminiscent of Case-Based Reasoning [28, 90], but DOP+ additionally defines a probabilistic distribution over equally similar structures to break ties.

9 Current Developments: Unsupervised DOP

The DOP approach in this Chapter presents a fully supervised learning technique: it starts out from corpora of example-derivations for language, music and problem-solving. A drawback of supervised learning is that it is extremely costly to create such annotated corpora. Moreover, all supervised approaches

have reached an asymptote on annotated corpora. It has therefore become increasingly clear that the next major step consists of generalizing these supervised approaches to semi-supervised or even unsupervised learning since they can directly operate with unlabeled raw data, of which virtually unlimited quantities are available.

In particular in NLP, there has been considerable progress in unsupervised learning during the last few years. The performance of unsupervised parsers has gone up from around 40% unlabeled *F*-score on the ATIS corpus [33, 89] to around 78% *F*-score on the Wall Street Journal (WSJ) corpus [64]. Yet, all unsupervised parsing models proposed so far limit either the lexical or the structural context that is taken into account, or both. That is, these unsupervised models operate by statistically comparing contiguous subsequences of sentences: if substrings appear in similar lexical contexts they are likely to form a constituent of the same category [33, 64, 65, 90]. However, for building accurate unsupervised parsers it is imperative to also take into account *non*-contiguous substrings. This may be illustrated by the comparative construction ‘more...than’ in the sentence “BA carried more people than cargo in 2004”. Furthermore, there exist many more lexical dependencies which may be separated by any number of other words and which can therefore not be described by contiguous substrings. Examples range from linguistic constructions such as ‘if...then’ to sentences such as “Companies in Vietnam are small-sized”, where the subject-verb agreement is non-contiguous (it is not Vietnam that is small-sized but Companies). What would be needed is an ‘all-subtrees’ approach to unsupervised parsing that statistically compares all possible subtrees rather than all possible substrings.

In [17], an unsupervised generalization of DOP was proposed, termed U-DOP. Instead of using all subtrees from a set of given parse trees, U-DOP initially assigns all possible (binary) trees to a large data-set of initial sentences and next uses the subtrees from these trees to compute the best parse trees for new sentences. The underlying methodology of U-DOP is similar to (supervised) DOP: since we do not know beforehand what kind of structures are appropriate, we should not *a priori* restrict the set of possible structures, but take them all and learn only those structures (and subtrees thereof) that are useful in analyzing new data. U-DOP thus allows initially for any partial non-contiguous string to form a syntactic group and is therefore richer than previous unsupervised parsing methods.

To give an illustration of this U-DOP model, consider the following part-of-speech (p-o-s) string NNS VBD JJ NNS from the Wall Street Journal which may correspond to the sentence “Investors suffered heavy losses”, (contrary to DOP, U-DOP currently works with p-o-s strings that are first tagged by a – possibly unsupervised – part-of-speech tagger). U-DOP starts by assigning all possible binary trees to this string, where each root node is labeled ‘S’ and each internal node is labeled ‘X’. Thus NNS VBD JJ NNS has a total of five binary trees as shown in Fig. 16 – where for readability we add words as well.

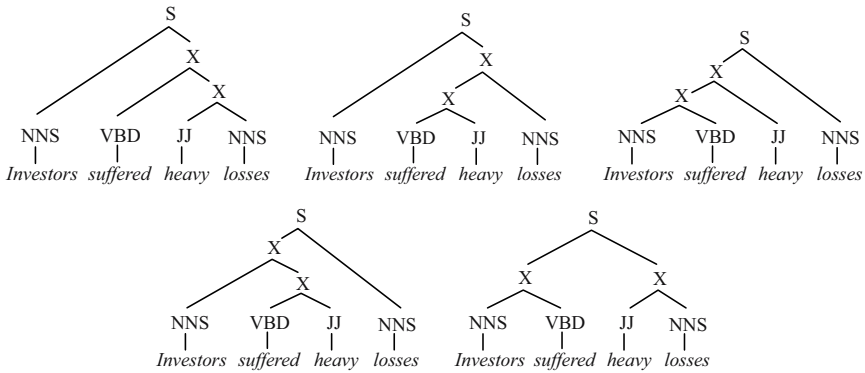


Fig. 16. All binary trees for “Investors suffered heavy losses”

New sentences can then be parsed by combining subtrees from all possible trees for given sentences, just as with the DOP+ model. We again let the DOP approach decide which trees – and subtrees thereof – are most useful in analyzing fresh data. Of course, if we only had the sentence “Investors suffered heavy losses” in our corpus, there would be no difference in probability or derivation length between the five parse trees in Fig. 16. However if we also have a different sentence where JJ NNS (heavy losses) appears in a different context, such as in “Heavy losses were reported”, its covering subtree gets a relatively higher frequency and the parse tree where ‘heavy losses’ occurs as a constituent gets a higher total probability.

While we can efficiently represent the set of all binary trees of a string by means of a chart, we need to unpack the chart if we want to extract subtrees from this set of binary trees. Also, since the total number of binary trees for the WSJ10 part (in other words, all WSJ sentences up to 10 words) is already 12 million, it is doubtful that we can apply the unrestricted U-DOP model to the WSJ in general. The U-DOP model in [17] therefore randomly samples a large subset from the total number of parse trees from the chart, and next computes the most probable parse trees for new sentences. In [18], U-DOP was extended with maximum likelihood training, using the Expectation-Maximization (EM) algorithm with cross-validation, called UML-DOP. It was shown that UML-DOP obtained the best reported results on inducing tree structures for three benchmarks: the English WSJ corpus, the German *Negra* corpus and the Chinese *Treebank for Mandarin* [18]. Moreover, we showed that UML-DOP even outperformed a well-known supervised parsing model, namely the *treebank grammar* from the WSJ corpus. This result was surprising since common wisdom had it that unsupervised approaches performed worse than supervised approaches. This result brought [18] to predict that the end of supervised parsing might be in sight.

While these recently developed unsupervised DOP models are thus very promising, there is still much work to be done: the UML-DOP model does not operate directly with word strings (due to data sparseness) and it neither induces syntactic categories or verb-argument structures. Moreover, unsupervised DOP models must still be developed for music and problem-solving. An overview paper on Unsupervised Data-Oriented Parsing must therefore await further research.

10 Conclusion

All state-of-the-art parsing systems are nowadays probabilistic and corpus-based. In this Chapter, we discussed the details of a well-known parsing approach, called DOP, which parses new data by probabilistically combining subtrees from a corpus of previously parsed data. DOP takes all subtrees and lets the statistics decide which subtrees contribute to the most probable parse trees. We showed how a particular instantiation of DOP, known as DOP+, integrates the notions of ‘simplicity’ and ‘likelihood’, and how it can be successfully applied to three different modalities: language, music and problem solving. We reported on experiments that show a consistent increase in accuracy if larger corpus subtrees are taken into account. Finally, we showed how the DOP approach can be extended to unsupervised learning by using the same underlying principle: assign all binary trees to all sentences and let the statistics decide which trees (and subtrees) are most useful in parsing new sentences.

Acknowledgements

We gratefully acknowledge support by way of the NWO Incentive Scheme Project ‘Towards a Unifying Model for Linguistic, Musical and Visual Processing’. This research was also partially supported by the EPSRC Advanced Research Fellowship ‘Combining Linguistic and Statistical Approaches to Spoken Language Processing’. The author is particularly grateful to Professor John Fulcher for his excellent help with editing this Chapter. As usual, all errors and inconsistencies remain the author’s responsibility.

References

1. Abeillé A (ed.) (2003) *Treebanks*. Kluwer Academic Publishers, Dordrecht, The Netherlands.
2. Alonso M, Finn E (1996) *Physics*. Addison Wesley, Reading, MA.
3. Baader F, Nipkow T (1998) *Term Rewriting and All That*. Cambridge University Press, UK.

4. Black E, Abney S, Flickinger D, Gnadiec C, Grishman R, Harrison P, Hindle D, Ingria R, Jelinek F, Klavans J, Liberman M, Marcus M, Roukos S, Santorini B, Strzalkowski T (1991) A Procedure for quantitatively comparing the syntactic coverage of English. In: *Proc. 5th DARPA Speech and Natural Language Workshop*, Pacific Grove, CA, Morgan Kaufmann, San Mateo, CA: 306–311.
5. Black E, Lafferty J, Roukos S (1992) Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. In: *Proc. 30th Association Computer Linguistics Conf. (ACL'92)*, Newark, DE, Association for Computer Linguistics, Stroudsburg, PA: 185–192.
6. Black E, Garside R, Leech G (1993) *Statistically-Driven Computer Grammars of English: The IBM/Lancaster Approach*. Rodopi, Amsterdam, The Netherlands.
7. Bod R (1992) Data-oriented parsing. In: *Proc. Computational Linguistics Conf. (COLING'92)*, Nantes, France, Association for Computer Linguistics, Stroudsburg, PA: 854–859.
8. Bod R (1998) *Beyond Grammar: An Experience-Based Theory of Language*. Stanford: CSLI Publications (Lecture Notes number 88), distributed by Cambridge University Press, Cambridge, UK.
9. Bod R (1999) Context-sensitive spoken dialogue processing with the DOP model. *Natural Language Engineering*, 5(4): 309–323.
10. Bod R (2000) Parsing with the shortest derivation. In: *Proc. 18th ACL Computational Linguistics Conf. (COLING'2000)*, Saarbrücken, Germany, Association for Computer Linguistics, Stroudsburg, PA: 69–75.
11. Bod R (2001) What is the minimal set of subtrees that achieves maximal parse accuracy? In: *Proc. 39th Association Computer Linguistics Conf. (ACL'2001)*, Toulouse, France, Association for Computer Linguistics, Stroudsburg, PA: 66–73.
12. Bod R (2002) A unified model of structural organization in language and music. *J. Artificial Intelligence Research*, 17: 289–308.
13. Bod R (2002) Memory-based models of melodic analysis: challenging the Gestalt principles. *J. New Music Research*, 31(1): 27–37.
14. Bod R (2003) An efficient implementation of a new DOP model. In: *Proc. 10th European Association Computer Linguistics Conf. (EACL'03)*, 12–17 April, Budapest, Hungary, Association for Computer Linguistics, Stroudsburg, PA: 19–26.
15. Bod R (2004) Exemplar-based explanation. In: *Proc. Computation and Philosophy Conf. (ECAP04)*, 3–5 June, Pavia, Italy.
16. Bod R (2005) Modeling scientific problem solving by DOP. In: *Proc. Cognitive Science Conf. (CogSci'05)*. Stresa, Italy: 103.
17. Bod R (2006) Unsupervised parsing with U-DOP. In: *Proc. 10th Computational Natural Language Learning Conf. (CONLL'2006)*, 8–9 June, New York, NY, Association for Computer Linguistics, Stroudsburg, PA: 85–92.
18. Bod R (2006) An all-subtrees approach to unsupervised parsing. In: *Proc. ACL Computational Linguistics Conf. (COLING'2006)*, Sydney, Australia, Association for Computer Linguistics, Stroudsburg, PA: 865–872.
19. Bod R (2006) Towards a general model of applying science. *Intl. Studies in the Philosophy of Science*, 20(1): 5–25.
20. Bod R (2006) Exemplar-based reasoning with the shortest derivation. In: Magnani L (ed.) *Model-Based Reasoning in Science and Engineering*. College Publications, London, UK: 119–140.

21. Bod R (2006) Exemplar-based syntax: how to get productivity from examples. *The Linguistic Review* (Special Issue on Exemplar-Based Models in Linguistics), 23(3): 289–318.
22. Bod R, Kaplan R (1998) A probabilistic corpus-driven model for lexical-functional analysis. In: *Proc. ACL Computational Linguistics Conf. (COLING'98)*, 10–14 August, Montreal, Canada, Association for Computer Linguistics, Stroudsburg, PA: 145–152.
23. Bod R, Hay J, Jannedy S (eds.) (2003) *Probabilistic Linguistics*. MIT Press, Cambridge, MA.
24. Bod R, Scha R, Sima'an K (eds.) (2003) *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL.
25. Bod R, Kaplan R (2003) A DOP model for lexical-functional grammar. In: Bod R, Scha R, Sima'an K (eds.) (2003) *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL.
26. Bonnema R, Bod R, Scha R (1997) A DOP model for semantic interpretation. In: *Proc. 4th European Association Computer Linguistics Conf. (EACL'97)*, Madrid, Spain, Association for Computer Linguistics, Stroudsburg, PA: 159–167.
27. Briscoe T, Waegner N (1992) Robust stochastic parsing using the inside-outside algorithm. In: *Proc. AAAI Workshop Statistically-Based Techniques in Natural Language Processing*, Menlo Park, CA, AAAI Press/MIT Press, Cambridge, MA: 39–53.
28. Carbonell J (1993) Derivational analogy: a theory of reconstructive problem solving and expertise acquisition. In: Michalski RS, Carbonell J, Mitchell T (eds.) *Machine Learning II*. Morgan Kaufmann, San Francisco, CA: 371–392.
29. Charniak E (1997) Statistical techniques for natural language parsing. *AI Magazine*, Winter: 32–43.
30. Charniak E (2000) A maximum-entropy-inspired parser. In: *Proc. 1st North American ACL Chapter Conf. (ANLP-NAACL'2000)*, Seattle, WA, Morgan Kaufmann, San Francisco, CA: 132–139.
31. Chater N (1999) The search for simplicity: a fundamental cognitive principle? *The Quarterly J. Experimental Psychology*, 52A(2): 273–302.
32. Chiang D (2000) Statistical parsing with an automatically extracted tree adjoining grammar. In: *Proc. 38th Association Computer Linguistics Conf. (ACL'2000)*, October, Hong Kong, China, Association for Computer Linguistics, Stroudsburg, PA: 456–463.
33. Clark A (2001) Unsupervised induction of stochastic context-free grammars using distributional clustering. In: *Proc. Computational Natural Language Learning Conf. (CoNLL'2001)*, July, Toulouse, France, Association for Computer Linguistics, Stroudsburg, PA: 97–104.
34. Chomsky N (1965) *Aspects of the Theory of Syntax*. MIT Press, Cambridge MA.
35. Collins M (1996) A new statistical parser based on Bigram lexical dependencies. In: *Proc. 34th Association Computer Linguistics Conf. (ACL'96)*, 23–28 June, Santa Cruz, CA, Association for Computer Linguistics, Stroudsburg, PA: 184–191.
36. Collins M (1997) Three generative lexicalised models for statistical parsing. In: *Proc. 35th Association Computer Linguistics Conf. (ACL'97)*, July, Madrid, Spain, Association for Computer Linguistics, Stroudsburg, PA: 16–23.
37. Collins M (1999) Head-Driven Statistical Models for Natural Language Parsing. *PhD Thesis*, University of Pennsylvania, PA.

38. Collins M (2000) Discriminative reranking for natural language parsing. In: *Proc. 17th Intl. Conf. Machine Learning (ICML-2000)*, Stanford, CA: 175–182.
39. Collins M, Duffy N (2001) Convolution kernels for natural language. In: Dietrich TG, Becker S, Ghahramani Z (eds.) *Advances in NIPS 14* (Proc. NIPS'2001), 3–8 December, Vancouver, Canada, MIT Press, Cambridge, MA: 617–624.
40. Collins M, Duffy N (2002) New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron. In: *Proc. 38th Association Computer Linguistics Conf. (ACL'2002)*, Philadelphia, PA, Association for Computer Linguistics, Stroudsburg, PA: 263–270.
41. Conklin D (2006) Melodic analysis with segment classes. *Machine Learning*, 65(2-3): 349–360.
42. Cussens J (2001) Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3): 245–271.
43. Dempster A, Laird N, Rubin D (1977) Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society*, 39: 1–38.
44. De Raedt L, Kersting K (2004) Probabilistic inductive logic programming. In: *Proc. Algorithmic Learning Theory (ALT) Conf.*, Lecture Notes in Computer Science 3244, Springer-Verlag, Berlin: 19–36.
45. Douglas J, Matthews R (1996) *Fluid Mechanics 1 (3rd ed.)*. Longman, Essex, UK.
46. Eisner J (1996) Three new probabilistic models for dependency parsing: an exploration. In: *Proc. 18th ACL Computational Linguistics Conf. (COLING'96)*, August, Copenhagen, Denmark, Association for Computer Linguistics, Stroudsburg, PA: 340–345.
47. Ferrand M, Nelson P, Wiggins G (2003) Unsupervised learning of melodic segmentation: a memory-based approach. In: *Proc. 5th European Society for the Cognitive Sciences of Music Conf. (ESCOM'2003)*, 8–13 September, Hanover, Germany.
48. Frazier L (1978) On Comprehending Sentences: Syntactic Parsing Strategies. *PhD Thesis*, University of Connecticut.
49. Fujisaki T, Jelinek F, Cocke J, Black E, Nishino T (1989) A probabilistic method for sentence disambiguation. In: *Proc. 1st Intl. Workshop Parsing Technologies*, 28–31 August, Pittsburgh, PA: 85–94.
50. Gahl S, Garnsey S (2004) Knowledge of grammar, knowledge of usage: syntactic probabilities affect pronunciation variation. *Language*, 80(4): 748–775.
51. Giere R (1988) *Explaining Science: A Cognitive Approach*. University of Chicago Press, Chicago, IL.
52. Goldberg A (2006) *Constructions at Work*. Oxford University Press, Oxford, UK.
53. Goodman J (1996) Efficient algorithms for parsing the DOP model. In: *Proc. Empirical Methods in Natural Language Processing*, Philadelphia, PA: 143–152.
54. Goodman J (2003) Efficient parsing of DOP with PCFG-reductions. In: Bod R, Scha R, Sima'an K (eds.) *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL.
55. Hearne M, Way A (2003) Seeing the wood for the trees: data-oriented translation. In: *Proc. Machine Translation Summit IX*, September, New Orleans, LO: 165–172.
56. Hearne M, Way A (2004) Data-oriented parsing and the Penn Chinese Treebank. In: *Proc. 1st Intl. Joint Conf. Natural Language Processing*, May, Hainan Island, China: 406–413.

57. Hearne M, Way A (2006) Disambiguation strategies for data-oriented translation. In: *Proc. 11th Intl. Conf. European Association for Machine Translation*, 19–20 June, Oslo, Norway.
58. Hoogweg L (2003) Extending DOP with insertion. In: Bod R, Scha R, Sima'an K (eds.) *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL.
59. Huron D (1996) The melodic arch in western folksongs. *Computing in Musicology*, 10: 2–23.
60. Johnson M (1998) PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4): 613–632.
61. Johnson M (2002) The DOP estimation method is biased and inconsistent. *Computational Linguistics*, 28(1): 71–76.
62. Jurafsky D (2003) Probabilistic modeling in psycholinguistics. In: Bod R, Scha R, Sima'an K (eds) *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL: 39–96.
63. Klein D (2005) The unsupervised learning of natural language structure. *PhD Thesis*, Department of Computer Science, Stanford University, CA.
64. Klein D, Manning C (2002) A general constituent-context model for improved grammar induction. In: *Proc. 40th Association Computer Linguistics Conf. (ACL'2002)*, July, Philadelphia, PA, Association for Computer Linguistics, Stroudsburg, PA: 128–135.
65. Klein D, Manning C (2004) Corpus-based induction of syntactic structure: models of dependency and constituency. *Proc. 42nd Association Computer Linguistics Conf. (ACL'2004)*, 21–26 July, Barcelona, Spain, Association for Computer Linguistics, Stroudsburg, PA: 438.
66. Kudo T, Suzuki J, Isozaki H (2005) Boosting-based parse reranking with subtree features. In: *Proc. 43rd Association Computer Linguistics Conf. (ACL'2005)*, June, Ann Arbor, MI, Association for Computer Linguistics, Stroudsburg, PA: 189–196.
67. Kuhn T (1970) *The Structure of Scientific Revolutions (2nd ed.)*. University of Chicago Press, Chicago, IL.
68. Lerdahl F, Jackendoff R (1983) *A Generative Theory of Tonal Music*. MIT Press, Cambridge, MA.
69. Longuet-Higgins H (1976) Perception of melodies. *Nature*, 263, October 21: 646–653.
70. Longuet-Higgins H, Lee C (1987) The rhythmic interpretation of monophonic music. In: Longuet-Higgins H (ed.) *Mental Processes: Studies in Cognitive Science*, MIT Press, Cambridge, MA.
71. Makatchev M, Jordan P, VanLehn K (2004) Abductive theorem proving for analyzing student explanations to guide feedback in intelligent tutoring systems. *J. Automated Reasoning*, (Special Issue: Automated Reasoning and Theorem Proving in Education), 32(3): 187–226.
72. Manning C (2003) Probabilistic syntax. In: Bod R, Hay J, Jannedy S (eds.) *Probabilistic Linguistics*. MIT Press, Cambridge, MA: 289–342.
73. Manning C, Schuetze H (1999) *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.
74. Marcus M, Santorini B, Marcinkiewicz M (1993) Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2): 313–330.

75. McClosky D, Charniak E, Johnson M (2006) Effective self-training for parsing. In: *Proc. North American Chapter of ACL Conf. Human Language Technology (NAACL-HLT 2006)*, June, New York, NY, Association for Computer Linguistics, Stroudsburg, PA: 152–159.
76. Mitchell T, Keller R, Kedar-Cabelli S (1986) Explanation-based learning: a unifying view. *Machine Learning*, 1: 47–80.
77. Mooney J, Zelle J (1994) Integrating ILP and EBL. *SIGART Bulletin*, 5(1): 12–21.
78. Muggleton S (1996) Stochastic logic programs. In: De Raed L (ed.) *Advances in Inductive Logic Programming* (Proc. 5th Intl. Workshop Inductive Logic Programming), IOS Press, Amsterdam, The Netherlands: 254–264.
79. Neumann G (2003) A data-oriented approach to HPSG. In: Bod R, Scha R, Sima'an K (eds) *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL.
80. Pereira F, Schabes Y (1992) Inside-outside reestimation from partially bracketed corpora. In: *Proc. 30th Association Computer Linguistics Conf. (ACL'92)*, Newark, DL, Association for Computer Linguistics, Stroudsburg, PA: 128–135.
81. Scha R (1990) Taaltheorie en taaltechnologie; competence en performance. In: de Kort Q, Leerdam G (eds) *Computertoepassingen in de Neerlandistiek*. Landelijke Vereniging van Neerlandici (LVVN-jaarboek), Almere, The Netherlands.
82. Schaffrath H (1995) The Essen Folksong Collection in the Humdrum Kern Format. In: Huron D (ed.) *Probabilistic Grammars for Music*. Center for Computer Assisted Research in the Humanities, Menlo Park, CA.
83. Sima'an K (1996) Computational complexity of probabilistic disambiguation by means of tree grammars. In: *Proc. 14th Computational Linguistics Conf. (COLING'96)*, 5–9 August, Copenhagen, Denmark, Association for Computer Linguistics, Stroudsburg, PA: 1175–1180.
84. Sima'an K (1999) Learning Efficient Disambiguation. *ILLC Dissertation Series 1999-02*, Utrecht University, The Netherlands.
85. Sima'an K, Itai A, Winter Y, Altman A, Nativ N (2001) Building a tree-bank of modern Hebrew text. *J. Traitement Automatique des Langues* (Special Issue on Natural Language Processing and Corpus Linguistics), 42(2): 347–380.
86. Temperley D (2001) *The Cognition of Basic Musical Structures*. MIT Press, Cambridge, MA.
87. Tomasello M (2003) *Constructing a Language*. Harvard University Press, Harvard, MA.
88. Van Lehn K (1998) Analogy events: how examples are used during problem solving. *Cognitive Science*, 22(3): 347–388.
89. van Zaanen M (2000) ABL: alignment-based learning. In: *Proc. 18th Computational Linguistics Conf. (COLING'2000)*, 31 July – 4 August, Saarbrücken, Germany, Association for Computer Linguistics, Stroudsburg, PA: 961–967.
90. van Zaanen M (2002) Bootstrapping Structure into Language. *PhD thesis*. School of Computing, University of Leeds, UK.
91. van Zaanen M, Bod R, Honing H (2003) A memory-based approach to meter induction. In: *Proc. 5th European Society for the Cognitive Sciences of Music Conf. (ESCOM5)*, September, Hanover, Germany: 250–253.
92. Veloso M, Carbonell J (1993) Derivational analogy in PRODIGY: automating case acquisition, storage, and utilization. *Machine Learning*, 10(3): 249–278.
93. Wertheimer M (1923) Untersuchungen zur lehre von der gestalt. *Psychologische Forschung*, 4: 301–350.

94. Younger D (1967) Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10(2): 189–208.
95. Zollmann A, Sima'an, K (2005) A consistent and efficient estimator for data-oriented parsing. *J. Automata, Languages and Combinatorics*, 10: 367–388.
96. Zuidema W (2006) What are the productive units of natural language grammar? A DOP approach to the automatic identification of constructions. In: *Proc. 10th Computational Natural Language Learning Conf. (CONLL'2006)*, 8–9 June, New York, NY, Association for Computer Linguistics, Stroudsburg, PA: 29–36.

Resources

1 Key Books

Cassell J, Sullivan J, Prevost S, Churchill E (eds.) (2000) *Embodied Conversational Agents*. MIT Press, Cambridge, MA.

Damasio A (1994) *Descartes Error*. Macmillian Publishers, London, UK.

Evans D (2001) *Emotion: the Science of Sentiment*. Oxford University Press, New York, NY.

LeDoux J (1996) *The Emotional Brain*. Simon and Schuster, New York, NY.

Oatley K, Jenkins JM (1996) *Understanding Emotions*. Blackwell Publishers, Oxford, UK.

Picard R (1997) *Affective Computing*. MIT Press, Cambridge, MA.

Plantec P (2004) *Virtual Humans*. Amacon, New York, NY.

Prendinger H, Ishizuka M (2004) *Life-Like Characters. Tools, Affective Functions, and Applications*. Springer-Verlag , Berlin.

Reeves B, Nass C (1996) *The Media Equation: How People Treat Computers, Televisions, and New Media Like Real People and Places*. Cambridge University Press, New York, NY.

2 Key Survey/Review Articles

Bates J (1994) The role of emotion in believable agents. *Communications ACM*, 37(7): 122–125.

Cowie R, et al. (2001) Emotion recognition in human-computer interaction. *IEEE Signal Processing Magazine*, 18(1): 32–80.

Dehn D, Van Mulken S (2000) The impact of animated interface agents: a review of empirical research. *Intl. J. Human-Computer Studies*, 52(1): 1–22.

Brave S, Nass C (2002) Emotion in human-computer interaction. In: Jacko JA, Sears A (eds.) *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Lawrence Erlbaum Associates, Mahwah, NJ: 81–96.

Picard RW, Klein J (2002) Computers that recognise and respond to user emotion: theoretical and practical implications. *Interacting with Computers*, 14(2): 141–169.

<http://emotion-research.net/deliverables> HUMAINE (Human-Machine Interaction Network on Emotion) deliverable Dxx: Proposed exemplar and work towards it:

- Scherer K, et al. (2005) _____D3e: Theory of Emotion.
- Douglas-Cowie E, et al. (2005) _____D5e: Data and Databases.
- Kollias S, et al. (2005) _____D4d: Signals and Signs of Emotion.
- Pelachaud C, et al. (2005) _____D6d: Emotion in Interaction.
- Canamero L, et al. (2005) _____D7d: Emotion in Cognition and Action.
- Stock O, et al. (2005) _____D8d: Communication and Emotions.
- Hook K, et al. (2005) _____D9d: Usability.
- Goldie P, et al. (2005) _____D10b: Interim report on ethical frameworks for emotion-oriented systems.

3 Organisations, Societies, Special Interest Groups

CHIL (Computers in the Human Loop)

<http://chil.server.de/serolet/is/101/>

COSY (Cognitive Systems for Cognitive Assistants)

<http://www.cognitivesystems.org/>

Design and Emotion Society

<http://www.designandemotion.org/>

Enactive Interfaces (EU Network of Excellence)

<http://www.reflex.lth.se/enactive/>

HUMAINE (Human-Machine Interaction Network on Emotion)

<http://emotion-research.net/>

International Society for Research on Emotion

<http://isre.org/prd/index.php>

SIMILAR (The European taskforce creating human-machine interfaces
SIMILAR to human-human interfaces)

<http://www.similar.cc/>

Virtual Human (Anthropomorphic Interaction Agents)

http://www.virtual-human.org/start_en.html

4 Research Groups

Affective Computing at MIT Media Lab

<http://affect.media.mit.edu/>

Cognition and Affect Project at University of Birmingham (UK)

<http://www.cs.bham.ac.uk/research/projects/cogaff>

Geneva Emotion Research Group

<http://www.unige.ch/fapse/emotion/>

LeDoux Lab, New York University

<http://www.cns.nyu.edu/home/ledoux/>

Relational Agents Group, Northeastern University

<http://www.ccs.neu.edu/research/rag/>

RITL (Center for Research of Innovative Technologies for Learning, Florida
State University)

<http://ritl.fsu.edu/>

Virtual Reality Lab, Swiss Federal Institute of Technology

<http://ligwww.epfl.ch/>

5 Discussion Groups, Forums

The Emotion Forum

<http://homepages.feis.herts.ac.uk/comqlc/emotion.html>

Emotional Intelligence Information Website

http://www.unh.edu/emotional_intelligence/

Facial Action Coding System (FACS) Manual

<http://face-and-emotion.com/dataface/facs/description.jsp>

Facial Expressions Resources Page

http://www.kasrl.org/facial_expression.html

Socially Intelligent Agents

<http://homepages.feis.herts.ac.uk/~comqkd/aaai-social.html>

Stanford University Persuasive Technology Lab

<http://captology.stanford.edu/>

Virtual Humans

<http://www.ordinarymagic.com/v-people/#>

6 Key International Conferences/Workshops

ACII 2005: 1st Intl. Conf. Affective Computing and Intelligent Interaction

<http://www.affectivecomputing.org/2005/>

ACE 2006: Agent Construction and Emotions: Modeling the Cognitive Antecedents and Consequences of Emotion

<http://www.ofai.at/~paolo.petta/conf/ace2006/>

Theories and Models of Emotion (HUMAINE Workshop – 2004)

<http://emotion-research.net/ws/wp3>

From Signals to Signs of Emotion and Vice Versa (HUMAINE Workshop – 2004)

<http://emotion-research.net/ws/wp4>

Data and Databases (HUMAINE Workshop – 2004)

<http://emotion-research.net/ws/wp5>

Emotion in Interaction (HUMAINE Workshop – 2005)

<http://emotion-research.net/ws/wp6/>

Emotion in Cognition and Action (HUMAINE Workshop – 2005)

<http://emotion-research.net/ws/wp7>

Emotion in Communication (HUMAINE Workshop – 2005)

<http://emotion-research.net/ws/wp8/proceedings-wswp8.pdf>

Innovative Approaches for Evaluating Affective Systems (HUMAINE Workshop – 2006)

<http://emotion-research.net/ws/wp9/>

7 (Open Source) Software

Croquet (Software for creating 3D collaborative multi-user online applications)

<http://www.opencroquet.org/>

Emofilt (Simulate emotional arousal with speech synthesis)

<http://felix.syntheticspeech.de/publications/emofiltInterspeech05.pdf>

FEELTRACE (Tool for rating the emotion expressed in audio-visual stimuli)

<http://emotion-research.net/download/Feeltrace%20Package.zip>

OpenAL (Cross Platform 3D Audio)

<http://www.openal.org/>

OpenGL (Graphics API)

<http://www.opengl.org/>

OpenMary (Open Source Emotional Text-to-Speech Synthesis System)

<http://mary.dfki.de>

TraceTools (Tools for tracing the presence of emotion)

<http://emotion-research.net/download/ECatPack.zip>

8 Data Bases

8.1 Multimodal Databases

Belfast Naturalistic Database

<http://www.idiap.ch/mmm/corpora/emotion-corpus>

ISLE project corpora

<http://isle.nis.sdu.dk/>

SMARTKOM

<http://www.phonetik.uni-muenchen.de/Bas/BasMultiModaleng.html#SmartKom>

SALAS

<http://www.image.ntua.gr/ermis/>

8.2 Face Databases

AR Face Database

http://cobweb.ecn.purdue.edu/~aleix/aleix_face_DB.html

CMU Facial Expression Database (Cohn-Kanade)

http://vasc.ri.cmu.edu/~idb/html/face/facial_expression/index.html

CMU PIE (Pose, Illumination and Expression) Database

http://www.ri.cmu.edu/projects/project_418.html

CVL Face Database

<http://www.lrv.fri.uni-lj.si/facedb.html>

Psychological Image Collection at Stirling

<http://pics.psych.stir.ac.uk/>

Japanese Female Facial Expression (JAFFE) Database

<http://www.kasrl.org/jaffe.html>

Yale Face Database

<http://cvc.yale.edu/projects/yalefaces/yalefaces.html>

Yale Face Database B

<http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html>

Ontology

Graph-Based Representation and Reasoning for Ontologies

Dan R. Corbett

Schafer Corporation, Washington, DC, USA,
dcorbett@schafertmd.com

1 Introduction

An ontology, in the Knowledge Engineering and Artificial Intelligence sense, is a framework for the domain knowledge of an intelligent system. An ontology structures the knowledge, and acts as a container for the knowledge. We define *knowledge conjunction* as one or more agents using multiple ontologies to perform tasks and understand the domain. Once a common ontology is agreed upon, the agents then have a common background in which to share knowledge. No current method exists that allows intelligent agents to agree on a common framework for sharing knowledge, although there has been some work in comparing semantic meanings within an ontology [44]. This means that agents are unable to use the knowledge of another agent, as the knowledge is meaningless if it isn't presented in a proper context or a common 'language'.

In this Chapter, we first give an overview of Conceptual Graph Theory, including what conceptual graphs are and how they work. We then take a different point-of-view for the representation of ontologies. Rather than constructing a CG to represent the ontology, we assert that the CG formalism is better exploited by using a combination of the concept type hierarchy, the canonical formation rules, the conformity relation and subsumption to act as the framework for the knowledge base. An unpopulated ontology (which is simply a framework for the knowledge) is represented by the type hierarchy without specific individuals, while the populated ontology (the framework, as well as the knowledge of the domain) is represented by a hierarchy and the specific conceptual graphs which instantiate individuals, constraints, situations or concepts.

Our definition of 'ontology' is a functional one. We can define what an ontology does more easily than we can define what it is. We use this definition to show how graphs represent knowledge, and are supported by the ontology. We then demonstrate the idea in a real-world knowledge domain.

2 Overview of Conceptual Graphs

It has been demonstrated many times that graphs are a powerful and efficient knowledge representation technique. [28] illustrate quite effectively why labeled graphs are useful for knowledge representation in general. Among the main advantages that they list are a solid grounding when it comes to combinatorial algorithms, and that a graph (as a mathematical object) allows a natural representation and therefore permits the construction of effective algorithms. Until recently, the major technique used in Computer Science for representing the semantic relationships between objects in a data structure was to use a graph technique known as Semantic Networks [22].

There have been many attempts to formalize and standardize these graphical knowledge representation schemes, but probably none has been as extensive and comprehensive in recent times as Conceptual Graphs. The major use of conceptual graphs is in representing the relationships between concepts in a system.

Conceptual Structures (or Conceptual Graphs, or ‘CGs’) are a knowledge representation scheme, inspired by the existential graphs of Charles Sanders Peirce and further extended and defined by John Sowa [34–36]. Informally, CGs can be thought of as a formalization and extension of Semantic Networks, although the origins are different. They are labeled graphs with two types of nodes: concepts (which represent objects, entities or ideas) and relation nodes, which represent relations between the concepts. As an example, Fig. 1 shows a conceptual graph which represents the knowledge that “The cat Felix is sitting on the mat which is known as mat 47”.

2.1 The Basics

Conceptual Graphs exist within a highly-structured, formal framework. This framework helps to guarantee a uniform semantics and that operations on the graphs are meaningful, sound and complete. Every concept or relation has an associated type. A concept may also have a specific referent or individual. A concept in a CG may represent a specific instance of that type (for instance, Felix is a specific instance, or individual, of type *cat*) or we may choose only to specify the type of the concept. That is to say that a concept may simply represent a generic concept for a type, such as *mammal* or *room*, or a concept may represent a specific object or idea, such as “my cat” or “the kitchen at the Smith’s house”. In the former case, the concepts in Fig. 1 would be shown

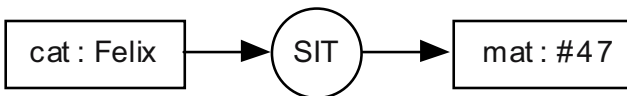


Fig. 1. A simple conceptual graph

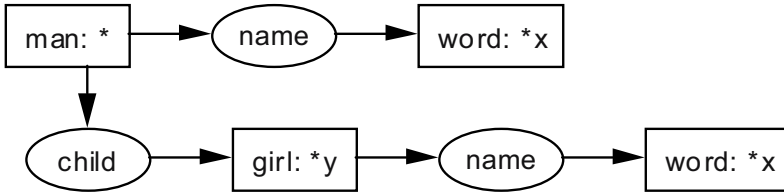


Fig. 2. Another example of a conceptual structure

as ‘*cat: **’ and ‘*mat: **’ indicating non-specified entities of types *cat* and *mat*. In the standard canonical formation rules for conceptual graphs, unbound concepts are existentially quantified.

Figure 2 shows another example conceptual graph. In this case, the graph represents the relationship between a man and his daughter. The graph can be read as “A man has a name, which is a word, and also has a child, which is a girl, who has the same name as the man.” Here, while the *** is used to represent any generic entity (a man or a girl - existentially quantified) the **x* is a named variable. The named variable is still generic and existentially quantified, but any value assigned to the variable is remembered (bound) to the variable name. This is how we can enforce a rule about a father and daughter sharing a common name.

A relation may have zero or one incoming arcs, and one or more outgoing arcs. The type of the relation determines the number of arcs allowed on the relation. The arcs always connect a concept to a relation. Arcs cannot exist between concepts, or between relations.

A canon in the sense discussed here is the set of all CGs which are well-formed, and meaningful in their domain. Canonical formation rules specify how CGs can be legally built and guarantee that the resulting CGs satisfy ‘sensitivity constraints’. Sensitivity constraints are rules in the domain which specify how a CG can be built, for example that the concept *eats* must have a theme which is *food*. Note that canonicity does not guarantee validity. A CG may be well-formed in the canonical formation rules for the domain, but still be false.

A type hierarchy is established for both the concepts and the relations within a canon. A type hierarchy is based on the intuition that some types subsume other types, for example, every instance of *cat* would also have all the properties of *mammal*. This hierarchy is expressed by a subsumption or generalization order on types. Many of these concepts are formalized later.

Conceptual Graphs are a useful and efficient knowledge representation tool. They can be used to represent the relations between complex objects in a system, and can represent multiple relations.

2.2 Fundamental Concepts

There are a few basic definitions that the student of Conceptual Graph Theory needs to be concerned with. These are the fundamental building blocks of conceptual graphs, and formally define how to build a domain theory in CGT. We will now visit the basic definitions in Conceptual Graph Theory (CGT). This Section draws on previous work on formal definitions for knowledge representation as defined by [51]. In our work, however, we follow the further formalized and refined versions of Sowa's original ideas for CGT presented by [46] and by [8], [28]. The following two definitions are adapted from their work, and from [12], [14].

We first define a background universe for our ontologies to exist in, which will give substance and order to the ontologies. A canon is the set of all valid expressions in the domain. The canon defines all the individuals that can exist, all possible relations between the individuals, and also imposes an ordering on the types of individuals and relations.

Definition 1 (canon)

A canon is a tuple $(T, I, \leq, ::, B)$, where

- T is the set of *types*; we will further assume that T contains two disjunctive subsets T_C and T_R containing types for concepts and relations.
- I is the set of *individuals*.
- $\leq \subseteq T \times T$ is the *subtype relation*. It is assumed to be a lattice (so there are types \top and \perp and operations \vee and \wedge).
- $:: \subseteq I \times T$ is the *conformity relation*. The conformity relation relates type labels to individual markers; this is essentially the relation which ensures that the typing of the concepts makes sense in the domain, and helps to enforce the type hierarchy.
- B is the *Canonical Basis function* (also called σ in the Conceptual Graphs literature). This function associates each relation type with the concept types that may be used with that relation; this helps to guarantee well-formed graphs.

It is common in the literature to write $c \in G$ instead of $c \in C$ when it is clear that c is a concept (similarly for relations $r \in G$).

Definition 2 (conceptual graph – CG)

A Conceptual Graph with respect to a canon is a tuple $G = (C, R, \text{type}, \text{referent}, \text{arg}_1, \dots, \text{arg}_m)$, where

- C is the set of *concepts*; $\text{type} : C \rightarrow T$ indicates the type of a concept, and $\text{referent} : C \rightarrow I$ indicates the referent marker of a concept.
- R is the set of conceptual *relations*, $\text{type} : R \rightarrow T$ indicates the type of relation, and each $\text{arg}_i : R \rightarrow C$ is a partial function where $\text{arg}_i(r)$ indicates the i -th argument of the relation r . The argument functions are partial as they are undefined for arguments higher than the relation's 'arity'. We adopt the convention that arg_0 indicates the (at most) one incoming arc. If there is no incoming arc to the relation, then arg_0 is undefined. We also define the function $\text{arity}(r)$ which returns an integer value representing the number of arguments that the relation r has.

2.3 Canonical Formation Rules

The following definitions are standard, classical definitions of CG formation, which date back to Sowa's original 1984 work on conceptual graphs [34], but which were formalized much more recently [29], [38]. We present here rules based on the work of [29].

Sowa (and others) also define a *copy* rule, which allows a new graph G' to be created as an exact duplicate of a graph G , and a *simplify* rule which allows the deletion of duplicate (and presumably redundant) relations. The *simplify* rule is just the equivalent of the internal join rule, but for relations.

2.4 Types and Inheritance

The discussion of type hierarchies presented here is adapted for conceptual graphs from [7], who discusses types and inheritance for Feature Structures. The set of types discussed in Definition 1 is arranged into a type hierarchy, ordered according to the specificity of each type. A type t is said to be more specific than a type s if t subsumes all of the information from s . We write $s \geq t$, and say that s *subsumes* t or is more general than t (or inversely, that t is *subsumed* by s , or is more specific than s). Equivalently to the above, one can write $t \leq s$. One may often read the expression that: “ s is a *supertype* of t ”, or “ t is a *subtype* of s ”. While these expressions are in common use, it confuses the issue of the difference between subsumption and inheritance.

A standard restriction on inheritance hierarchy specifications is that they do not contain inheritance loops [7]. It would simply be inconsistent (and even nonsensical) to be able to follow a chain of subtype links from a type back to itself.

Definition 3 (canonical graph)

A canonical graph is a conceptual graph which is in the closure of the conceptual graphs in its canonical basis under the following operations, called the canonical formation rules.

1. *External join.* Given two CGs $G = (C, R, type, referent, arg_1, \dots, arg_m)$ and $G' = (C', R', type', referent', arg'_1, \dots, arg'_m)$ (without loss of generality we assume C and C' to be disjoint) $\forall c \in C$, and $\forall c' \in C'$, where $c = c'$ (that is, they have identical types and referents), the external join of C and C' is the CG $G'' = (C(C' - \{c'\}), R \cup (R'_{c':=c}), type'', referent'', arg''_1, \dots, arg''_m)$. The subscript $c' := c$ denotes the replacement of every occurrence of c' by c . The functions *type* and *referent* are such that: $f''|_C \equiv f$ and $f''|_{C'} \equiv f'$.
2. *Internal join.* Given a CG $G = (C, R, type, referent, arg_1, \dots, arg_m)$ and two nodes $c, d \in C$ with identical types and referents, the internal join is the CG $G' = (C - \{d\}, (R_{d:=c}), type|_{C-\{d\}}, referent|_{C-\{d\}})$. The subscript $d := c$ denotes the replacement of every occurrence of d by c .
3. *Restrict type.* Given a CG $G = (C, R, type, referent, arg_1, \dots, arg_m)$ and a node $c \in C$ with type t which has a subtype $s \neq \perp$, the restrict type is the CG $G' = (C, R, type', referent, arg_1, \dots, arg_m)$ such that $type'(c) = s, \forall d \neq c : type'(d) = type(d)$.
4. *Restrict referent.* Given a CG $G = (C, R, type, referent, arg_1, \dots, arg_m)$ and a node $c \in C$ with $referent(c) = *$ and an individual marker $i \in I$, the restrict referent is the CG $G' = (C, R, type, referent', arg_1, \dots, arg_m)$ with $referent'(c) = i, \forall c : referent'(d) = referent(d)$, and $type(c) :: i$.

In early pioneering work on the unification of first-order terms, [32] used the natural lattice structure of first-order terms, which was a partial ordering based on subsumption of terms [16]. Many terms (or types in our case) are not in any subsumption relation, for example *cat* and *dog*, or *wood* and *mammal*. Unification corresponds to finding the greatest lower bound of two terms in the lattice [30]. The bottom of any lattice, which is represented with the symbol \perp , is the type to which all types can unify, and represents inconsistency. The top of the lattice, represented by \top , is the type to which all pairs of types can generalize, and is called the *universal* type. Every type is a subtype of \top . Subsumption type hierarchies can then be seen as lattices that admit unification and generalization [30].

2.5 Specialization, Projection and Subsumption

The common specialization of two conceptual graphs, s and t , is known as a *join*, and is represented as $s \vee t$. The common generalization of the two graphs is known as a *meet*, and is represented as $s \wedge t$.

Our definitions of *unification*, *consistency*, and *type subsumption* are based on formal concepts of projection and lower bounds. [7] defines each of these operators (for Feature Structures) as a *morphism*. We start by following Carpenter's definitions, and then modify them to work with the properties of conceptual graphs. A morphism is then a mapping from the set of nodes of one conceptual graph to the set of nodes of another that preserves the order of relation arguments and the values of those arguments. In a morphism, all of the connections and arguments are preserved. The following definition of *projection* is the standard definition used in recent conceptual graph literature ([7], [10], [11], [21], [28], [29], [46]).

Definition 4 (projection)

$G = (C, R, type, referent, arg_1, \dots, arg_m)$ is said to have a projection into $G' = (C', R', type', referent', arg'_1, \dots, arg'_m)$, $G \geq G'$, if and only if there is a pair of functions $h_C : C \rightarrow C'$ and $h_R : R \rightarrow R'$, called morphisms, such that:

$$\begin{aligned} \forall c \in C \text{ and } \forall c' \in C', h_C(c) = c' \text{ only if } type(c) \geq type'(c'), \text{ and} \\ referent(c) = * \text{ or } referent(c) = referent(c') \\ \forall r \in R \text{ and } \forall r' \in R', h_R(r) = r' \text{ only if } type(r) \geq type'(r') \\ \forall r \in R, arg'_i(h_R(r)) = h_C(arg_i(r)) \end{aligned}$$

Willems also includes the following non-emptiness condition in his definition of projection [46]: $\forall c \in C$ there is a concept $c' \in C'$, such that $h_C(c) = c'$.

This non-emptiness condition guarantees that all the concepts present in the more general graph are also present in the more specific graph, although they may be in a more specific state. Willems' definition allows for the more specific graph to have concepts of a more specific type, or for a generic referent to be replaced by a specific individual. The definition that we use also admits the non-emptiness condition.

Regarding the join and meet definitions, it is sometimes essential to obtain the most general common specialization for a given pair of Conceptual Graphs. In this case it is important to prove that not only is the graph obtained a consistent specialization of the two graphs being considered, but also that it is

the unique graph which is the most general of all possible common specializations. Such a graph is known as the Greatest Lower Bound, as it represents the highest join which falls under the two graphs in the specialization hierarchy.

Definition 5 (greatest lower bound)

The greatest lower bound (GLB) of two CGs is the most general common specialization of the two conceptual graphs. Let G'' be a specialization of G and G' . G'' is the GLB of G and G' if, for any conceptual graph U where $G \vee G' = U$, either $G'' \geq U$ or $G'' = U$.

The GLB of two graphs s and t is written as $s \sqcap t$. Conversely, the most specific common generalization, known as the least upper bound (LUB), of two graphs is written $s \sqcup t$. Note that it is not always possible to find a *unique* GLB. In these instances, it is often the case that a greedy algorithm is used which picks the first G'' which matches the constraints.

Definition 6 (subsumption)

We say that a conceptual graph G subsumes another conceptual graph G' , or $G \geq G'$, iff G' can be obtained by applying a finite number of canonical formation rules to G .

Note that Definition 6 is actually redundant, as subsumption is simply another form of projection. Since any application of the canonical formation rules to a graph s will always produce a graph t which is more specific than the original, s will necessarily have a projection into the new graph t . [28] formalize this idea, and demonstrate that $s \geq t$ iff there exists a projection from s to t .

While Definition 6 is presented here for completeness, and to give formal substance to any discussion of subsumption, the rest of our discussion will concern itself strictly with the use of projection. Any mention of subsumption from this point can be construed as meaning projection.

3 Projection as an Ontology Operator

We base our formal definition of ontology on the formal definition of canon, as defined earlier. We now treat canon in the sense of the set of all CGs which are well-formed, and meaningful in their domain. We treat canonical formation

rules as specifying how ontologies can be legally built and guarantee that the resulting graphs satisfy ‘sensitivity constraints’, (the Canonical Basis). The canonical basis is a set of rules in the domain which specifies how the relations can be legally used, for example that the concept *eats* must have a theme which is *food*.

The set of types discussed in Definition 2 is arranged into a type hierarchy, ordered according to the specificity of each type. Type hierarchies are established for both the concepts and the relations within a canon. A type hierarchy is based on the intuition that some types subsume other types, for example, every instance of *cat* would also have all the properties of *mammal*. This hierarchy is expressed by a subsumption or generalization order on types. A type t is said to be more specific than a type s if t specializes some of the concepts from s . As with all type hierarchies, the universal type is shown at the top of the hierarchy, and is represented by \top . The absurd type is shown at the bottom of the graph, and is represented by \perp . Type hierarchies are discussed and illustrated in detail in [13].

The definitions for type hierarchies and for the operations on those hierarchies inform our definition of an ontology. We want an ontology to provide a framework for the semantics of a domain. A canon, as defined above, provides the background for the representation, since we can use the definitions of relations, subsumption and conformity to support our definition of an ontology. We can now formally define an ontology as the particular set of hierarchies that are created for a given domain, along with all of the operations on a canon.

Definition 7 (ontology)

An ontology in a given domain M with respect to a canon is a tuple (T_{CM}, T_{RM}, I_M) , where

T_{CM} is the set of concept types for the domain M and T_{RM} is the set of relation types for the domain M .

I_M is the set of individuals for the domain M .

An ontology is then a collection of types and individuals, which forms a framework for the knowledge in a domain. The collection is arranged into a hierarchy based on the subtype relation \leq . The canon provides the basis for subsumption in the ontology and guarantees consistency among the relations and in the typing of individuals.

Note that this hierarchy is not necessarily a taxonomy, in that a type may have *multiple supertypes*. Further note that there is no point on the hierarchy where we must make a distinction between a type and an instance. Every

concept on the hierarchy is treated as a type. A type may have subtypes and supertypes, but there is no need to distinguish these from instances of the types.

This is distinct from the object-oriented objective of objects inheriting all the properties of a class of objects. The essential difference is in, for example, treating a kitchen as you would any generic room. The *type* room can be placed, occupy space, and have specific values for color and number of doors. A *class* of rooms will have attributes, but cannot be said to occupy a space or have specific dimensions, or have a specific count or placement of doors. The generic room can have constraints placed on its attributes, and finally can be specialized into a kitchen. Fundamentally, a generic room can take the place of a specialized room, unlike a class of objects.

The ontology (as a concept type hierarchy) acts as the framework, with conceptual graphs that conform to the hierarchy used to instantiate concepts in the domain. The ontology is populated by creating conceptual graphs which represent actions, ideas, situations or states in the domain. Recall, though, that a conceptual graph need not be a complete description, and will always be treated in the same manner as any other type.

The closest approach to demonstrating an equivalence between First-Order Logic (FOL) and Conceptual Graphs is due to [3]. They use a restrictive form of CGs, in which each concept type is allowed only one individual to represent it. Once the existential operator has been applied to a generic referent, all concepts of that type must use that one individual. Clearly, this makes it much easier to interpret Conceptual Graphs into FOL. Given that restriction, [3] show that graph derivation through projection is sound and complete. They discuss a method for graph deduction on these restricted graphs.

The real significance of the work by [3], and indeed of our own work, is the proof that deduction systems over Conceptual Graphs are not only possible, but also effective ways of handling knowledge merging, comparison and deduction.

4 Projection of Ontology Types

The definitions of consistency and type subsumption in this Chapter are based on formal concepts of projection and lower bounds from Conceptual Graph Theory [35]. Projection is the operation used to determine subsumption relations, and to find similarities between parts of the knowledge base. A more general type G is said to subsume a more specific type H if G has a projection into H . For example, the type *mammal* would have a projection into the type *cat*.

The following definitions of projection are modified from the standard definition used in recent Conceptual Graph literature ([12], [21], [28], [30], [46]).

Rather than defining projection from one graph into another, these definitions represent projection of types, and therefore define the subsumption operator on type hierarchies.

Definition 8 (concept projection)

Given two concept types, s and t , s is said to have a projection into t if and only if there is a morphism $h_C : C \rightarrow C'$, such that:

$\forall c \in s$ and $\forall c' \in t', h_C(c) = c'$ only if $type(c) \geq type'(c')$, and $referent(c) = *$ or $referent(c) = referent(c')$

C is the set of concepts, $type : C \rightarrow T$ indicates the type of a concept, and $referent : C \rightarrow I$ indicates the referent marker of a concept.

Definition 9 (relation projection)

Given two relation types, s and t , s is said to have a projection into t if and only if there is a morphism $h_R : R \rightarrow R'$, such that:

$\forall r \in R$ and $\forall r' \in R', h_R(r) = r'$ only if $type(r) \geq type'(r')$

R is the set of relations, and $type : R \rightarrow T$ indicates the type of a relation.

The definition of type subsumption is based on notions of graph projection. Projection and subsumption are defined for individual graphs to help determine their ordering in accordance with the type hierarchy, and to allow unification, deduction and combination of graphs. While we concern ourselves here with issues of type projection, the topic of graph projection and subsumption is covered in detail in [12], [14], [30].

This definition of projection then gives us a formal definition for subtype and supertype and for subsumption on the partial order of the types in the hierarchy. The operations of join, meet and unify are now simply applications of the projection operator. Finding types which are compatible (in other words, that can be unified) is now a matter of finding a common subtype (or *join*) between the two types. If the only common subtype is \perp , then there can be no comparison.

Consistency and validity of conceptual graphs are guaranteed by adhering to strict rules regarding the formation of new graphs. A domain is defined by a set of basic graphs, the canonical basis and by its type hierarchies. All other graphs must be derived from the canonical basis by the use of the canonical

formation rules. Like the rest of the CG field, these rules are still evolving, but they all involve the same basic ideas, as expressed here.

This definition of projection then gives us a formal definition for subtype and supertype and for subsumption on the partial order of the types in the hierarchy. All of these operations are now simply applications of the projection operator. Finding types which are compatible is now a matter of finding a common subtype (join) of the two types. If the only common subtype is \perp , then there can be no unification.

5 Knowledge Conjunction

5.1 Ontology Comparison and Conjunction

As an operator for ontology comparison, the use of the projection operator becomes obvious. When comparing two ontologies, one need only determine whether the two concepts under consideration (one from each ontology) are in a subtype-supertype relation. This means that there needs to be a way for specifying which two types to compare. As discussed for CG unification in [15], the user will need to specify a starting node, or in this case a starting type in each ontology. This may simply be \top , but can be any node that the user wants to specify.

Many terms (or types in our case) are not in any subsumption relation, for example *cat* and *dog*, or *wood* and *mammal*. Inheritance hierarchies can be seen as lattices that admit unification and generalization [30]. So, in our case, combining two ontologies is the process of finding the common points in the two ontologies (represented as lattices) and merging the rest of the structures together.

An example for such an approach is Login [2], where first-order terms are replaced by feature terms. In the ψ -terms of ([1], [2]), subterms are labeled symbolically, rather than by argument position, and there is no fixed arity. The novel contribution of ψ -terms is in the use of type inheritance information. At-Kaci's view of unification was as a filter for matching partial structures, using functions and variables as the 'filters'. Then, his unification technique uses information from a taxonomic hierarchy to achieve a more gradual filtering.

An example of Ait-Kaci's ideas from [30] illustrates this gradual filtering technique. Assume that we have the following inheritance information, as illustrated in Fig. 3: *birds* and *fish* are *animals*; a *fish-eater* is an *animal*; a *trout* is a *fish*; and a *pelican* is both a *bird* and a *fish-eater*. Then unifying the following ψ -terms:

$$\begin{aligned} & \textit{fish eater} \textit{ (likes} \rightarrow \textit{trout)} \\ & \textit{bird} \textit{ (color} \rightarrow \textit{brown; likes} \rightarrow \textit{fish)} \end{aligned}$$

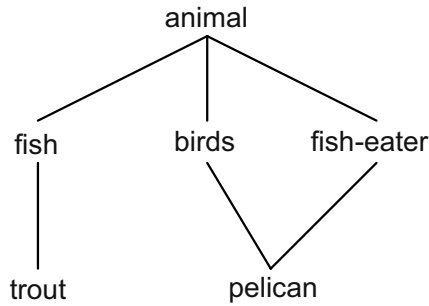


Fig. 3. A type hierarchy

will yield the new ψ -term:

$$pelican \text{ (color} \rightarrow \text{brown; likes} \rightarrow \text{trout)}$$

Unification does not fail on comparing *fish-eater* to *bird*, or *trout* to *fish*. Instead, the conflict is resolved by finding the greatest lower bound on each of the two pairs of items in the taxonomic hierarchy, in this case *pelican* and *trout*, respectively. In this manner, Ait-Kaci's system naturally extends the information merging (or *knowledge conjunction*) nature of unification.

The merging of two ontologies is somewhat more complicated, and also more interesting and useful than merely an extension of the projection operation. A unification of two graphs contains neither more nor less information than the two graphs being unified. This is the idea behind knowledge conjunction. The merging of two ontologies is a matter of finding a common starting point on the two hierarchies (usually with the assistance of the user) and then continuing outward from that point in a depth-first manner to find other matching points.

The main thrust of previous research has been the unification of CGs in terms of conjoining the knowledge contained in two different graphs [11], [15]. In our case, these pieces of partial information are represented by Conceptual Graphs. However, our current work involves combining the knowledge of two entire domains. We want to be able to combine the expert knowledge of two systems, or even combine knowledge from different sources, not merely gather additional information.

When an ontology is represented by a type hierarchy constructed in this way, subsumption can be used to combine, refine and reuse the knowledge contained in the graphs. This further allows us to perform reasoning over the knowledge in the graphs as concepts. Reasoning is not limited to objects, classes or libraries, but can also be applied to generic concepts in the knowledge.

5.2 Unification, Constraints and Conceptual Graphs

The standard method for representing and validating constraints has been to use type subsumption to specify which concept types (or subsumed subtypes) are valid in a system. One could constrain values in a knowledge representation system by forcing the concepts to conform to a specified type, or else to be subsumed by that type. A similar method applies to relations. To extend a previous example, the concept *eats* is specified to occur only between an agent which is an *animal* and a theme which is a *food*. Any individual used in the *animal* concept must conform to the *animal* type, which means that it must either be *animal*, or be subsumed by *animal*, such as *cat* or *reptile* or “Henry the Meerkat that lives at Adelaide Zoo.”

Unification is related to constraint processing in that constraints are now used to play the role in theorem proving and logic programming that unification of terms once played in Constraint Satisfaction Problems (CSP) [4]. The unification-based approach computes projections and general unifiers and applies them to the terms under consideration. The constraint-based approach uses constraints to determine which instances are valid. Constraints can be seen as a filter that prohibits instantiations of the variables not satisfying this constraint [4]. Further concepts in Constraint Satisfaction, Constraint Logic Programming and related areas are discussed in detail in the next Section.

This Section discusses the current methods for unification and knowledge combination of Conceptual Graphs. In the methods described in the current CG literature, constraints are sometimes handled during the unification process, but again not as a standard CG technique. In order to make a CG programming language feasible and usable, it is essential for the user to be able to validate a set of constraints over a system. It is also essential to be able to combine knowledge in a way which is sensitive to the domain, and the knowledge being represented. Unification is the likely method for doing this.

The usual abstract definition given for the unification problem is to find an object z that fits both of the descriptions of two objects x and y [30]. We discuss unification of Conceptual Graphs in terms of combining the knowledge contained in two different graphs. While this may involve term substitution and constraint solving, we are more concerned with knowledge combination as discussed by [7]. Carpenter defines unification as a system in which two pieces of partial information can be combined into a single unified whole. In our case, these pieces of partial information are represented by Conceptual Graphs. [8] refers to this idea as information conjunction, but in our work, it is knowledge conjunction that is more important to us. Unification here is the combining of pieces of knowledge, represented as Conceptual Graphs, in a domain. Where information is simply a gathering and processing of data, knowledge is the intelligent application of information in a domain. Knowledge conjunction uses unification to combine partial or incomplete knowledge into a single result.

5.3 Knowledge Structures, Partialness and Unification

The Relationship Between Unification and Knowledge Structures

Unification is related to Knowledge Structures through the concept of partialness. Structures which are not completely specified can be merged together through unification. In this Section, we discuss the idea of knowledge conjunction in terms of partially-specified structures being unified. The purpose, need and intent of unification of ontologies is clarified.

Partialness

In work on unification, *partialness* means that a structure need not contain all information that is implied about it by its structure and types. A partial representation is used here as a generalized, or higher-level description of an object in the domain. Whether a structure is partial or not depends on the context of the knowledge, and the domain. In domain terms, a model might be partial against one set of knowledge but complete with respect to a subset of the knowledge. To take an example from the Architectural Design domain, if our current domain knowledge of a building is limited to its spatial organization, a complete model of it would assign functions to physical spaces. Such a model would be partial with respect to a larger set of knowledge, containing for example, knowledge of how to construct the building.

The main thrust of the research described here is the unification of Conceptual Graphs in terms of conjoining the knowledge contained in two different graphs. While this may involve term substitution (or the Conceptual Graphs equivalent – instantiation, subsumption, variable binding, and the like) and constraint solving, our research is more concerned with knowledge conjunction as discussed in [7]. Unification then becomes the combining of pieces of knowledge in a domain, represented as Conceptual Graphs. We define unification as an operation that simultaneously determines the consistency of two pieces of partial or incomplete knowledge, and if they are consistent, combines them into a single result.

When an ontology is represented by the use of Conceptual Graphs constructed in this way, subsumption can be used to combine, refine and reuse the knowledge contained in the graphs. This further allows us to perform reasoning over the knowledge in the graphs as concepts. Reasoning is not limited to objects, classes or libraries, but can also be applied to generic concepts in the knowledge. We demonstrate reasoning over generic concepts in the next Section.

One major advantage that Conceptual Graphs have over other representation schemes is that they contain existentially quantified concepts that can still be unified. In Feature Structures theory [7] for example, it is important to know whether one is attempting to unify the *intensions* or the *extensions*

of two Feature Structures (FS). Essentially, the intension of a Feature Structure is all of the attributes (or properties, or features) of a construct. The extension of a Feature Structure is the actual object being represented, with the attributes specified, even if only partially. In Feature Structures theory, one must decide whether the Feature Structures being unified are of the same *intensional* type, or the same *extensional* type, and then seek to identify the two FSs under that type. The unification of two FSs under their extensional type is simply the identification of all their values for their features (similar to type labels and individual markers for the concepts in CGs). There is no way to derive identities of intensional types of two Feature Structures, as there are no values to be compared.

Essentially, the intension of a knowledge structure is all of the attributes (or properties, or features) of a construct. The extension is the actual object being represented, with the attributes specified, even if only partially ([39], [40]).

The significance of intensionality in a representation scheme is the simple fact that two structures can be identical in all aspects yet remain distinct objects. In an intensional representation scheme, two structures which represent the same structure must be explicitly identified as being the same. One major advantage that Conceptual Graphs have is that graphs which contain existentially quantified concepts can still be unified.

Intensionality, Join and Unify

It is essential to clarify the difference between the ‘join’ operator, introduced earlier, and the general concept of unification. The difference between these two operators can be illustrated in the following way. In the standard canonical formation rules for Conceptual Graphs, unbound concepts are existentially quantified.

We take for our example the two graphs in Fig. 4, which can be interpreted as “Felix is on some object”, and “There is some animal sitting on that particular mat”. Joining these two graphs is not possible under the standard canonical formation rule for external join because there is no projection from one graph to the other. However, there are individual concepts which can be joined, such as the concept that “Felix is a cat” and “animal.” However, as discussed earlier, true unification is the knowledge conjunction of the two graphs. The unification of these two Conceptual Graphs would be similar to the unification of ψ -terms presented by [1]. The unification is therefore “Felix sat on mat number 47”, as shown in Fig. 5. Here, the more general concepts of ‘animal’, ‘on’, and ‘object’ have been replaced by their more specific instances. This illustrates that unification is more than an external join, and is composed of several operations, including join.

The external join rule can be used to ‘glue together’ two graphs in Willems’ sense, in that a few compatible concepts and relations can be joined together

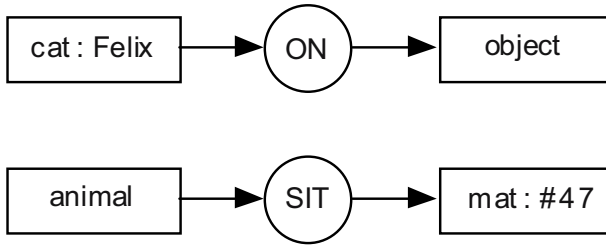


Fig. 4. Is Felix on the mat?

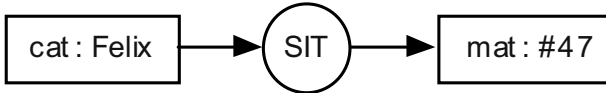


Fig. 5. Felix is on the mat

from two graphs to make a larger, joined graph. [95] then attempts to create a truly unified graph by finding the least upper bound of the two graphs that will validate this newly joined graph. As discussed earlier, in the true sense of unification, simply joining a few concepts and relations does not guarantee the conjunction of the knowledge contained in the graphs.

Unification, however, is somewhat more complicated, and also more interesting and useful. The unification of two graphs contains neither more nor less information than the two graphs being unified. Figure 5 shows that the unification of the two graphs in Fig. 4 still retains all the information of the original two graphs. This is the idea behind knowledge conjunction. Recall from earlier discussions, though, that unification of graphs can guarantee canonicity, but not validity. In other words, the graph produced by the unification is guaranteed to represent reasonable information in the domain (canonical), but it is not guaranteed to preserve the truth value of the statements (valid).

6 An Architectural Design Tool

The results discussed in this Section are those recorded from the application of the knowledge conjunction reasoning tool operating over the domain of architectural design. The point of automated search for the designer is to use computer media that engage designers in exploring design modifications. The design user may want to create new designs, or index, compare or adapt existing designs. This type of user requires efficient representations for the designs and states (of designs) in a symbol system [42]. The designer needs to be able to represent spaces of possibilities which are both relevant to the language and knowledge of design and lend themselves to tractable computations.

Consider a design for the kitchen of a custom-made house. In this design, the architect has specified some of the lighting design and that the floor area

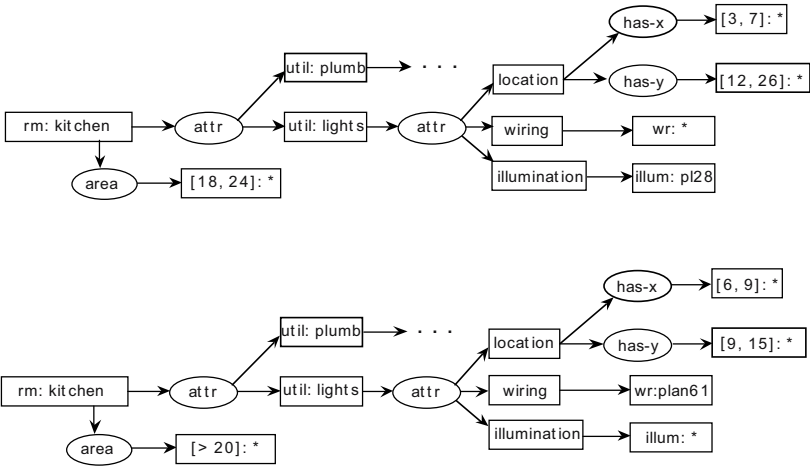


Fig. 6. Requirement for a kitchen design, together with a matching previous design

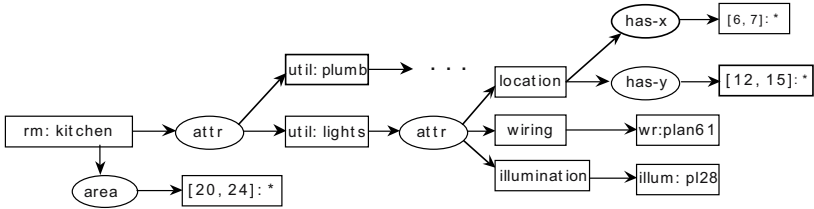


Fig. 7. Results of unification of the two graphs

must be greater than 20 square meters. The architect has also retrieved an old design, which specifies the remainder of the lighting design. The graphs specifying the partial design and the retrieved design are shown in Fig. 6.

The knowledge conjunction software discussed above combines these two graphs into a single result which represents neither more nor less knowledge than the original graphs (Fig. 7). In this graph, all the original knowledge of the first two graphs has been preserved, and the values in the concepts have been joined as specified.

The significance of these results is to demonstrate that not only can conceptual graphs be used to represent designs, but they are also a dynamic reasoning tool that can aid the designer in completing the designs. Further, since graph matching software exists, it can be used to find, retrieve and reuse previously stored designs.

7 An Architectural Design Tool: Results and Discussion

Conceptual Graphs can be used to efficiently represent a building design ontology. The use of Conceptual Graphs is an efficient method for representing not only the designs, but also constraints on the designs and knowledge

conjunction of designs. The system described in this Chapter allows general designs to be represented as concepts, and also allows values to be constrained by specifying real-valued constraints as intervals.

The three main areas where the architects want the contribution of Knowledge Conjunction are in type subsumption, knowledge-level reasoning, and pattern matching. First, architects want to be able to use type subsumption to make statements such as “An office (or kitchen, or corridor) is a kind of room. All the properties which apply to one should apply to its specializations”. This is distinct from the object-oriented objective of objects inheriting all the properties of a class of objects. The essential difference is in treating a kitchen as you would any generic room. A generic room can be placed, occupy space, and have attributes like color and number of doors. A *class* of rooms will have attributes, but cannot be said to occupy a space or have specific dimensions, or have a specific count or placement of doors.

The knowledge conjunction model that we developed gives this ability to the architects. The algorithm allows the user to specialize designs by matching (unifying) previous designs with the current design problem. Since all characteristics, attributes and constraints are carried along in the unification, the specialization represents all of the design concepts included in the more generic design. Further, and more importantly, there is no real separation between generic and specific, since all points in between can be represented. Conceptual Graphs combined with the ability to specialize using unification are the ideal tool for the knowledge conjunction approach and the constructive nature of architectural design.

The second major concern of architectural designers was the ability to have knowledge-level reasoning. That is, they want to be able to speak in the language of the architect, not the language of the computer (or Computer-Aided Design – CAD system). The user wants to be able to refer to the ‘North Wall’ or ‘door’ without resorting to discussing geometric coordinates in space. The user wants to depart from previous CAD-based data-level processing, and work at the knowledge level in the architecture domain.

This is certainly another area where Conceptual Graphs and unification combine to bring a solution to this domain. While spatial coordinates (and their constraints) can be stored in a graphical representation of a room, there is no need for the user to bother with using them. The graph can be manipulated as a whole, and treated as a room, rather than a square in a diagram. The completed system will not deal with lines and boxes, but rather with specializing entire designs for rooms (or houses, or office buildings). This approach frees the architect from dealing with data-level concerns of numbers and coordinates, and allows the architect instead to deal with the architectural design.

Finally, the users want to be able to start with a high-level, generic description of a building, and then make queries such as “Can this bay structure be used in the support structure?”, or “Do the constraints match up adequately

for a particular technology to be used? If yes, tell me the constraints under which it is usable”.

Once again, the work presented in this Chapter meets the requirements of the architects. A query is represented as a Conceptual Graph. The user can specify a type of structure for support, and make the query by attempting to unify the structure with the more generic design. If the unification fails, then the user knows that the proposed structure does not meet the constraints of the design problem. If the graphs unify, then the resulting graph will contain the constraints which must be met in order to make the design work.

Overall, the system of unification over constraints on Conceptual Graphs presented in this Chapter gives a set of tools to the designer. The ability to use knowledge conjunction with constraints to handle objects at the knowledge level greatly leverages the ability of the designer to work efficiently.

8 The Air Operations Officer

The results discussed in this Section are those recorded from the application of the knowledge conjunction reasoning tool operating over the defense domain. The domain knowledge is represented as Conceptual Graphs with constraints on either the structure of the graph or on the values in the concepts [11]. Here, we discuss the idea behind the reasoning mechanism by employing order sorted unification and constraints within the domain of architectural design. The concepts discussed previously were implemented in *Allegro Common Lisp* on a Sun Workstation.

An Air Operations Officer (usually known as an OPSO) is the defense officer responsible for deciding the appropriate defensive response to an air threat. A study of the Operations Officer decision-making methods was recently conducted, using a cognitive modeling technique ([26], [27]). The study was used to show the usefulness of cognitive modeling in deriving rules from expert knowledge. In this Section, we only make use of the rules which resulted from the study; the cognitive modeling technique is not discussed here.

In the domain of the Operations Officer, the magnitude of the response to an air threat is in proportion to the threat itself. So, if the opposing aircraft are very close, or if the aircraft is of a type which can cause a great deal of damage (known as a *strike* aircraft), then the response is large. If the threat is smaller, then the response is smaller. For example, Fig. 8 shows a rule in this domain (we have borrowed the style of [6] to express the rule, although we do not employ Cao’s fuzzy reasoning here.) This graph expresses the rule that if a fighter aircraft (small threat) is between 400 and 500 nautical miles distant, then assert a threat level of ‘alert 60’ (the lowest level of alert, in which response fighters must be ready to take off within sixty minutes), and a single fighter is assigned to deal with this threat.

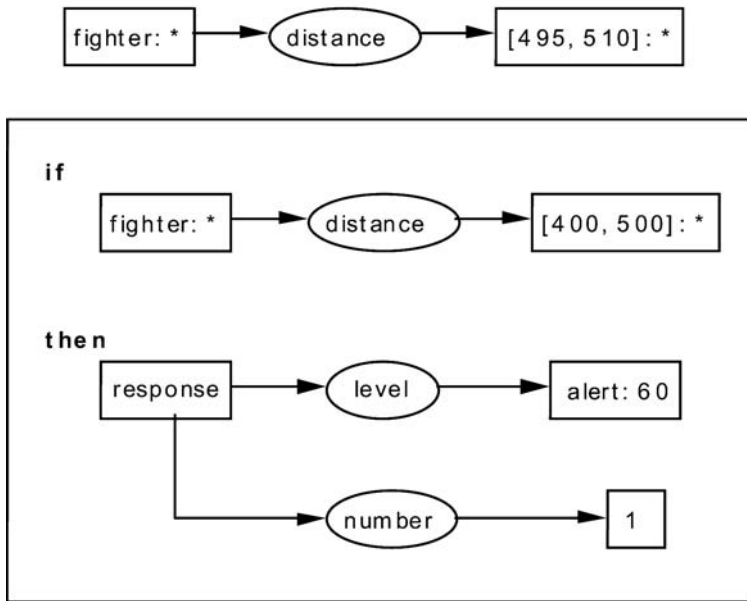
Assertion:

Fig. 8. An assertion and matching rule in the defense domain

The assertion shown in Fig. 8 unifies with the **if** portion of this rule, since the aircraft is certainly a fighter, and there is a join on the intervals of the distance concepts. A join exists here because we define interval concepts on an interval type hierarchy. Informally, the lattice is ordered on interval inclusion, such that two intervals have a join if there is some other interval which is in the ‘overlap’ area of the two intervals. Formal definitions and discussions on interval type hierarchies are described in [31] and [12]. The **then** portion of the rule represents the response to the situation, and it is asserted into the current world knowledge. In this manner, we can represent the decision-making capabilities of the Operations Officer.

The rule shown in Fig. 9 is used for a bigger and more impending threat. Any threat aircraft which is closer than 400 nautical miles is considered an immediate threat, and a response squadron must be ready very quickly. Further, a strike aircraft is one which can inflict a great deal of damage, and is therefore dealt with more severely than a fighter aircraft.

The assertion shown in Fig. 9 states that a bomber is known to be between 380 and 390 nautical miles distant. Our type hierarchy indicates that a bomber is a type of strike aircraft. Because of the proximity of the threat, the response aircraft are put on ‘alert 10’ status. Because of the enormity of the threat, two fighters are assigned to deal with the target aircraft. Again, the assertion unifies with the **if** portion of the rule, causing the **then** portion of the rule to be asserted.

Assertion:

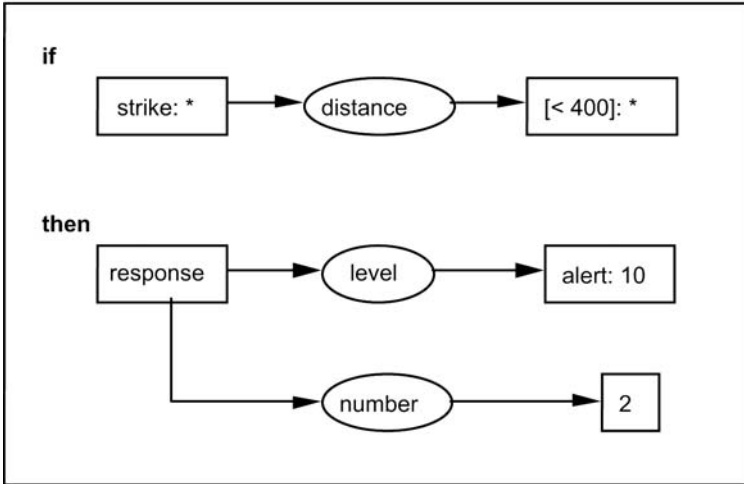


Fig. 9. Another assertion and rule from the same domain

9 The Air Operations Officer: Results and Discussion

Conceptual Graphs and knowledge conjunction can be used to efficiently represent a set of rules in the domain of the Air Operations Officer. The use of Conceptual Graphs is an efficient method for representing the complete ontology of the OPSO, not only in the rules, but also in the exploration and use of the knowledge of types of aircraft and responses. General rules can be represented as Conceptual Graphs, and then specialized dynamically to match the current situation and describe an appropriate response.

10 Conclusions: Semantics for a Knowledge Web

Much of the work on the semantic web has centered around the use of ontologies. There has been some previous work in formalizing the definitions and operations on ontologies, but so far they have been restricted either by structure, by limited inheritance rules, or by interoperability with ontologies designed by another designer. For an ontology to be truly flexible and useful, it must contain all of the following properties:

- deduction,
- relations among first-class objects,

- comparison or merging of concepts or complete ontologies, and
- subsumption (or some type of flexible inheritance).

Many researchers are working on furthering the idea of semantic annotations of web pages (the work of [5], for example). Many of these projects use XML and languages derived from XML (such as ebXML, XACML, and the like). Others are using RDF and languages based on OWL or DAML+OIL. The extensive work by ([25], [24], [23]) and others provides many tools for using, manipulating and even comparing ontologies, but still fall short in being completely expressive.

OWL [37] (and the language it was derived from, DAML+OIL) is another ontology tool which is based on Description Logics [36]. Fundamentally, DAML+OIL and OWL represent a set of syntax rules built on a decidable fragment of First Order Logic (FOL). The semantics of the language are derived from its FOL foundations, so there are useful definitions of properties such as deduction and soundness. However, when using the most common ontology tools, many authors have trouble when describing the relation between a class and an object [5]. For example, DAML+OIL makes a clear separation between object classes and data types [29]. Another major criticism of DAML+OIL was that it had a ‘weak semantics’, an issue that still has not been fully addressed in OWL (see [33] for a discussion and any resolution of OWL-related issues). Neither language supports flexible inheritance.

While some members of the semantic web research community are proclaiming that XML is the answer [14], it is nevertheless widely acknowledged that these languages fall short of what is really needed to support the semantics that will be necessary for the future of the web. It is already acknowledged that RDF and DAML+OIL still lack a serious semantics [43], and work continues on defining this [25].

We must conclude that while there has been a large body of research work generated around languages for the World Wide Web, many authors seem to be converging on the idea that a formal definition of the semantics of the web is now becoming necessary. What needs to be examined next is not whether XML, RDF and OWL can carry that kind of semantics, but whether in fact the type of semantics is really the problem. It is not a question of agreeing on a common representation standard, but of finding a method for comparing the semantics of various ontologies.

There are some very good ideas and tools contained in recent research (UML, ebXML, DAML+OIL, RDF, and so on) but we cannot just put all these formalisms in the same pot and stir, and expect a coherent, useful web authoring (or knowledge retrieval) tool to emerge. We can all acknowledge that Description Logics are useful and efficient at categorizing objects and creating a hierarchy of types. They can classify new concepts and specify constraints on the type hierarchy. But is this enough?

[18] asserts that the assumption that an ontology language should have an unambiguous, well-understood meaning does not hold anymore. Instead, it is now necessary to start asking the questions that need to be answered in order for the web to become a knowledge-based web. It is now time for the knowledge representation community to agree to disagree about representation schemes, and start to address the issue of how to understand an ontology constructed by another author.

Using Conceptual Graphs to represent the underlying ontology, we have demonstrated a method for automated reasoning on ontologies. Type hierarchies and the canonical formation rules efficiently specialize graphs into concrete instances. A simple unification operation, using join and type subsumption, is used to perform knowledge conjunction of the concepts represented as graphs. The significance of our work is that the previously static knowledge representation of ontology is now a dynamic, functional reasoning system. However, many questions still remain to be answered by the ontology research community.

References

1. Ait-Kaci H (1986) An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45(3): 293–351.
2. Ait-Kaci H, Nasr R (1986) LOGIN: a logic programming language with built-in inheritance. *J. Logic Programming*, 3(3): 185–215.
3. Amati G, Ounis I (2000) Conceptual graphs and first order logic. *The Computer J.*, 43(1): 1–12.
4. Baader F, Siekmann J (1994) Unification theory. In: Gabbay DM, Hogger CJ, Robinson JA (eds.) *Handbook of Logic in Artificial Intelligence and Logic Programming*. Clarendon Press, Oxford, UK, 2: 41–126.
5. Bechhofer S, Carr L, Goble C, Kampa S, Miles-Board T (2002) The semantics of semantic annotation. In: Meersman R, Tari Z (eds.) *Proc. Intl. Conf. Ontologies, Databases and Semantics*, 30 October – 1 November, Irvine, CA, Springer, New York, NY: 1152–1167.
6. Cao TH, Creasy PN, Wuwongse (1997) Fuzzy unification and resolution proof procedure for fuzzy conceptual graph programs. In: Lukose D, Delugach HS, Keeler M, Searle L, Sowa JF (eds.) *Proc. 5th Intl. Conf. Conceptual Structures*, August, Seattle, WA, Springer, New York, NY: 386–400.
7. Carpenter B (1992) *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, UK.
8. Chein M, Mugnier M-L (1992) Conceptual graphs: fundamental notions. *Revue d'Intelligence Artificielle*, 6(4): 365–406.
9. Cobb EE (2002) Will web services cause the widespread adoption of the Internet by business? In: Meersman R, Tari Z (eds.) *Proc. 1st Intl. Conf. Ontologies, Databases, and Application of Semantics*, 30 October - 1 November, Irvine, CA, IEEE Press, Piscataway, NJ.
10. Cogis O, Guinaldo O (1995) A linear descriptor for conceptual graphs and a class for polynomial isomorphism test. In: Ellis R, Levison R, Rich W, Sowa

- JF (eds.) *Proc. 3rd Intl. Conf. Conceptual Structures*, August, Santa Cruz, CA, Springer-Verlag, New York, NY: 263–277.
11. Corbett DR (2001) Conceptual graphs with constrained reasoning. *Revue d'Intelligence Artificielle*, 15(1): 87–116.
 12. Corbett DR (2001) Reasoning with conceptual graphs. In: Stumptner M, Corbett D, Brooks MJ (eds.) *Proc. 14th Australian Joint Conf. Artificial Intelligence*, December, Adelaide, South Australia, Lecture Notes in Computer Science 2256, Springer-Verlag, Berlin.
 13. Corbett DR (2002) Reasoning with ontologies by using knowledge conjunction in conceptual graphs. In: Meersman R, Tari Z (eds.) *Proc. Intl. Conf. Ontologies, Databases and Applications of Semantics*, October, Irvine, CA, Springer, New York, NY: 1304–1316.
 14. Corbett DR (2003) *Reasoning and Unification over Conceptual Graphs*. Kluwer Academic Publishers, New York, NY.
 15. Corbett DR, Woodbury RF (1999) Unification over constraints in conceptual graphs. In: Tepfenhart WM, Cyre WR (eds.) *Proc. 7th Intl. Conf. Conceptual Structures*, July, Blacksburg, VA, Lecture Notes in Computer Science 1640, Springer-Verlag, New York, NY: 470–479.
 16. Davey BA, Priestley HA (1990) *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK.
 17. Fikes R, McGuinness DL (2001) *An axiomatic semantics for RDF, RDF schema, and DAML+OIL*. Knowledge Systems Laboratory, Stanford University, Palo Alto, CA.
 18. Franconi E (2004) Using ontologies. *IEEE Intelligent Systems*, 19(1): 73–74.
 19. Heymans S, Vermeir D (2002) A defeasible ontology language. In: Meersman R, Tari Z (eds.) *Proc. Intl. Conf. Ontologies, Database and Applications of Semantics*, October, Irvine, CA, Springer-Verlag, New York, NY: 1033–1046.
 20. Knight K (1989) Unification: a multidisciplinary survey. *ACM Computing Surveys*, 21(1): 93–124.
 21. Leclère M (1997) Reasoning with type definitions. In: Lukose D, Delugach HS, Keeler M, Searle L, Sowa JF (eds.) *Proc. 5th Intl. Conf. Conceptual Structures*, August, Seattle, WA, Springer-Verlag, New York, NY: 401–414.
 22. Lehmann F (1992) Semantic networks. *Computers & Mathematics with Applications*, 23(2-5): 1–50.
 23. McGuinness DL (2003) Ontologies come of age. In: Fensel D, Hendler J, Lieberman H, Wahlster W (eds.) *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, Cambridge, MA: 171–197.
 24. McGuinness DL, Fikes R, Hendler J, Stein LA (2002) DAML+OIL: an ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5): 72–80.
 25. McGuinness DL, Fikes R, Rice J, Wilder S (2000) The Chimaera ontology environment. In: Kautz H, Porter B (eds.) *Proc. 17th National Conf. Artificial Intelligence*, July, Austin, TX, AAAI Press/MIT Press, Cambridge, MA: 1123–1124.
 26. Mitchard H (1998) Cognitive model of an operations officer. *PhD Thesis*, Department of Computer and Information Science, University of South Australia, Adelaide.
 27. Mitchard H, Winkles J, Corbett DR (2000) Development and evaluation of a cognitive model of an Air Defence Operations Officer. In: Davis C,

- van der Gelder TJ, Wales R (eds.) *Proc. 5th Biennial Conf. Australasian Cognitive Science Society*, May, Adelaide, South Australia, Springer-Verlag, Berlin: 479–492.
28. Mugnier M-L, Chein M (1996) Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle*, 10(6): 7–56.
 29. Müller T (1997) Conceptual Graphs as Terms: Prospects for Resolution Theorem Proving. *Technical Report TR97-01*, Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands.
 30. Nguyen P, Corbett D (2006) A basic mathematical framework for conceptual graphs. *IEEE Trans. Knowledge and Data Engineering*, 18(2): 261–271.
 31. Older WJ (1997) Involution narrowing algebra. *Constraints*, 2: 113–130.
 32. Reynolds JC (1970) Transformational systems and the algebraic structure of atomic formulas. *Machine Intelligence*, 5: 153–163.
 33. Smith MK (2002) *Web Ontology Issues W3C*, 5.10 (available online at <http://www.w3.org/2001/sw/WebOnt/webont-issues.html>. – last accessed 9 July, 2007).
 34. Sowa JF (1984) *Conceptual Structures: Information Processing in Mind and Machine*. Addison Wesley, Reading, MA.
 35. Sowa JF (1992) *Conceptual Graphs Summary. Conceptual Structures: Current Research and Practice*. Ellis Horwood, Chichester, UK.
 36. Sowa JF (1999) Conceptual graphs: draft proposed American National Standard. In: Tepfenhart WM, Cyre WR (eds.) *Proc. 7th Intl. Conf. Conceptual Structures*, July, Blacksburg, VA, Springer-Verlag, New York, NY: 1–65.
 37. van Harmelen F, Hendler J, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2004) *OWL Web Ontology Language Reference* (available online at <http://www.w3.org/TR/owl-ref/> – last accessed 9 July 2007).
 38. Wermelinger M, Lopes JG (1994) Basic conceptual structures theory. In: Tepfenhart W, Cyre W (eds.) *Proc. 2nd Intl. Conf. Conceptual Structures*, August, College Park, Maryland, Springer-Verlag, New York, NY: 144–159.
 39. Wille R (1996) Conceptual structures of multicontexts. In: Eklund P, Ellis G, Mann G (eds.) *Proc. 4th Intl. Conf. Conceptual Structures*, August, Sydney, Australia, Springer-Verlag, Berlin: 23–29.
 40. Wille R (1996) Short introduction to formal concept analysis. In: Eklund P, Ellis G, Mann G (eds.) *Proc. Intl. Conf. Conceptual Structures*, August, Sydney, Australia, Springer-Verlag, Berlin: 1–22.
 41. Willems M (1995) Projection and unification for conceptual graphs. In: Ellis, Levinson, Rich, Cruz (eds.) *Proc. 3rd Intl. Conf. Conceptual Structures*, August, Santa Cruz, CA, Springer-Verlag, New York, NY: 278–292.
 42. Woodbury R, Datta S, Burrow AL (2000) Erasure in design space exploration. In: Gero JS (ed.) *Proc. 6th Intl. Artificial Intelligence in Design Conf.*, June, Worcester, MA, Kluwer, New York, NY: 521–531.
 43. Yang G, Kifer M (2002) On the semantics of anonymous identity and reification. In: Meersman R, Tari Z (eds.) *Proc. 1st Intl. Conf. Ontologies, Databases and Applications of Semantics*, 30 October – 1 November, Irvine, CA, Springer-Verlag, New York, NY: 1047–1066.
 44. Yang G, Kifer M (2002) Well-founded optimism: inheritance in frame-based knowledge bases. In: Meersman R, Tari Z (eds.) *Proc. Intl. Conf. Ontologies, Databases and Applications of Semantics*, 30 October – 1 November, Irvine, CA, Springer-Verlag, New York, NY: 1013–1032.

Resources

1 Key Books

Sowa JF (1984) *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, MA.

(this is the classic text in which Sowa describes the fundamental ideas behind the development of conceptual graphs. He also discusses his philosophy of knowledge representation and the work of Charles Sanders Peirce, whose work was the inspiration for CGs)

Sowa JF (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA.

(this book captures some of the later research in the area and has a stronger emphasis on logic)

Corbett DR (2003) *Reasoning and Unification over Conceptual Graphs*. Kluwer Academic Publishers, New York, NY.

(this research monograph contains many detailed definitions and explanations of conceptual structures which the senior researcher will find as useful as the new student of knowledge representation)

2 Key Survey/Review Articles

Nguyen P, Corbett DR (2006) A basic mathematical framework for conceptual graphs. *IEEE Trans. Knowledge and Data Engineering*, 18(2): 261–271.

(this article is a good introduction to the formal side of conceptual graphs, containing the complete theory from the basics to advanced concepts of unification and ontology)

The following three articles are each about applications of the techniques in this Chapter to various domains. The interested reader will find descriptions of the theory and algorithms used to implement the systems:

Nguyen PHP, Corbett DR (2006) Building corporate knowledge through ontology integration. In: Hoffmann A, Kang B-H, Richards D, Tsumoto S (eds.) *Proc. Pacific Rim Knowledge Acquisition Workshop*, August, Guilin, China. Lecture Notes in Computer Science LNCS 4303, Springer-Verlag, Berlin: 223–229.

Morneau M, Mineau GW, Corbett DR (2006) Using an automatically generated ontology to improve information retrieval. In: de Moor A, Polovina S, Delugach H (eds.) *Conceptual Structures: Inspirations and Applications*, Proc. Conceptual Structures Tools Interoperability Workshop, 16–21 July, Aalborg, Denmark. Lecture Notes in Computer Science 4068, Springer-Verlag, Berlin: 27–36.

Corbett DR, Rouff C (2006) Self optimization using conceptual graphs for NASA autonomous systems. In: Sterritt R, Hinchey M, Bapty T (eds.) *Proc. 3rd IEEE Workshop Engineering of Autonomous and Autonomous Systems (EASE'06)*. April, Columbia, MD. IEEE Computer Society, Piscataway, NJ: 149–157.

3 Research Groups

Université Laval, Quebec City, CA
<http://www.ift.ulaval.ca/~moulin>
 (Cognitive Informatics Laboratory)
<http://www.lic.ift.ulaval.ca/Lic/Aceuil.html>

Virginia Polytechnic and State University (Virginia Tech)
 Automatic Design Research Group
<http://www.ee.vt.edu/~adrg/>

Knowledge Graphs Research Group LIRMM, Montpellier, France
<http://www.lirmm.fr/~cogito/>

4 Discussion Groups, Forums

There is an email list – conceptual graphs forum
 see <http://conceptualgraphs.org/> for instructions on how to join.

5 Key International Conferences/Workshops

There is an annual international conference on conceptual graphs that has traditionally moved among North America, Australia and Europe – see <http://www.iccs.info/> for information on the current conference.

The annual conference on ontologies, databases and applications of semantics, usually held in Europe, usually has several technical sessions on knowledge representation, ontologies and conceptual semantics
see <http://www.cs.rmit.edu.au/fedconf/> for information on the current conference.

6 (Open Source) Software

Amine is a multi-layer platform dedicated to the development of Intelligent Systems and Multi-agent Systems.

<http://amine-platform.sourceforge.net/>

CharGer is a prototype conceptual graph editor developed the University of Alabama in Huntsville, free to noncommercial use, and which runs under Java.

<http://sourceforge.net/projects/charger/>

CPE is a modular environment that provides functional modules to give functionality to a user without having to take the whole environment.

<http://port.semanticweb.org/CPE>

WebKB is a collection of tools for information retrieval and knowledge representation based on conceptual graph concepts.

<http://www.webkb.org/>

Prolog+CG is an object-oriented extension of PROLOG, based on CGs. CG (both simple and compound) is a basic data structure, like a term. PROLOG+CG is implemented with Java 2.

<http://prologpluscg.sourceforge.net/>

CoGITaNT is a set of several useful utilities: a set of library routines in C++ for conceptual modeling, some knowledge bases in conceptual graphs, and an XML specification for CGXML.

<http://cogitant.sourceforge.net/>

See http://wiki.anykb.org/CG_tools for a comparison of the many features of various CG tools.

An Ontology-Based Intelligent Mobile System for Tourist Guidance

Toby H.W. Lam¹, Raymond S.T. Lee², and James N.K. Liu¹

¹ Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, cshwlam@comp.polyu.edu.hk, csnkliu@comp.polyu.edu.hk

² IATopia Group Ltd., Tsim Sha Tsui, Kowloon, Hong Kong, raymond@iatopia.com

1 Introduction

It is common for Internet search engines such as **Google** and **Yahoo** to use keyword searching. Keyword searching employs techniques such as frequency-inverse document frequency (TF-IDF) [21] to determine the importance of a word in a document. These methods have several problems. Firstly, keyword searches locate web pages using the input keywords, without reference to semantics. The use of semantic information, perhaps in conjunction with frequency-based search methods, can be expected to produce meaning-based searches more relevant to users' meaning-driven queries.

Meaning-based or semantic searches cannot be conducted without semantically oriented technologies. One such technology which may facilitate semantic searches is the Semantic Web [2]. The Semantic Web has received substantial attention from the research community recently. The Semantic Web – the next generation World Wide Web – aims to provide a new framework that can enable knowledge sharing and reuse. The Semantic Web uses agent technology, ontology, and a number of standard markup languages such as RDF, OWL and RDFS to formally model information represented in web resources. This makes it accessible to humans and computers working together, perhaps in conjunction with intelligent network services such as search agents. Related to the development of the Semantic Web, new research findings have been forthcoming in areas such as Knowledge Engineering, ontology-based Information Retrieval and ontology-based agents.

Ontology is one of the main components used to enable knowledge interoperability within the Semantic Web. From a philosophical point of view, ontology means the study of entities and their relationships. From an Artificial Intelligence (AI) point of view, ontology is the explicit specification

of concepts. In fact, ontology is usually defined as an explicit specification of conceptualization. That is, ontology is an hierarchical relationship between terms within a domain that specifies defined terms and the relationships between those terms. A domain-specific ontology is a tool for modeling resource structures and meanings and allows software programs (agents) to perform automated tasks for users. These include searching, customizing and scheduling, which have as one of their points of reference the idea of meaning. In addition, ontology also plays a vital role in knowledge sharing and exploration, particularly in multiagent-based communication where the content of messages can be exchanged between different agents.

Given that two domains are sufficiently similar, one way to model a domain ontology is to reuse or extend an existing ontology. It is difficult, however, to model a domain ontology completely from scratch. The most common way to model a new ontology is to invite an ontology expert to model the domain. Unfortunately, few such experts exist. In this Chapter, we propose an approach where, instead of using domain experts, we model an ontology using structural information from a number of websites. In our case, we gathered structural information from websites then analyzed this information to produce an ontology for a travel website. This was motivated by our belief that structural information is the lay person's view of a specific domain. Instead of inviting a professional/expert to model the ontology, we instead model the domain ontology by using a common or lay person's understanding. In this Chapter, we describe in detail how to construct the ontology by using structural information.

We also employed the travel ontology to develop an agent-based tourist guiding system called – *iJADE FreeWalker*. There are many wireless devices such as Portable Digital Assistants (PDAs) and mobile phones on the market; these devices are usually small in size, lightweight and operate for a considerable time. We developed a prototype tourist guidance system on a Pocket PC, based on the modeled ontology. The main aim of this context-aware tourist guidance system is to provide tourists with more helpful tourist information. *iJADE FreeWalker* was developed under *iJADE* – an intelligent Java Agent-based Development Environment (<http://www.ijadk.com>). The system is integrated with a GPS (Global Positioning System) receiver to locate the user's position. The system is capable of gathering nearby tourist information, such as shopping, sightseeing, entertainment, restaurants, and so forth.

The rest of this Chapter is organized as follows. In Sect. 2, we provide an overview of the Semantic Web. In Sect. 3, we review some related work involving the Semantic Web and guidance systems. In Sect. 4, we describe the ontology-based tourist guidance system, including details of how we modeled the travel ontology, as well as the *iJADE FreeWalker* system architecture. Sect. 5 presents experimental results, while Sect. 6 rounds off with conclusions and suggested avenues for future research.

2 Background

2.1 The Semantic Web

The development of the Semantic Web proceeds in layers. The main purpose of this layering approach is that it is easier to achieve consensus in each layer. Figure 1 visualizes the Semantic Web architecture of the World Wide Web Consortium – W3C (<http://www.w3.org/2001/sw/>). Starting from the bottom of the ‘layer cake’, Extensible Markup Language (XML) (<http://www.w3.org/TR/xml11/>) is used for self-description documents. XML enables data exchange across the web, but it does not represent any meaning or knowledge embedded within the data. On top of XML reside the Resource Description Framework (<http://www.w3.org/RDF>) and RDF Schema (RDFS) (<http://www.w3.org/TR/rdf-schema>). RDF is a metadata model for making statements about web resources, in the form of a subject-predicate-object expression called a RDF ‘triple’. Since RDF has an XML-based syntax, it is located on top of the XML layer. RDFS is a language for describing vocabularies in RDF; it is a semantic extension of RDF – a primitive language for writing ontologies. The upper layer of the Semantic Web architecture consists of a logic layer, a proof layer, and a trust layer. The logic layer is used to enhance the ontology language for writing application-specific knowledge. The proof layer executes rules and evaluates these together with the trust layer mechanism for applications, allowing a decision to be made as to whether to

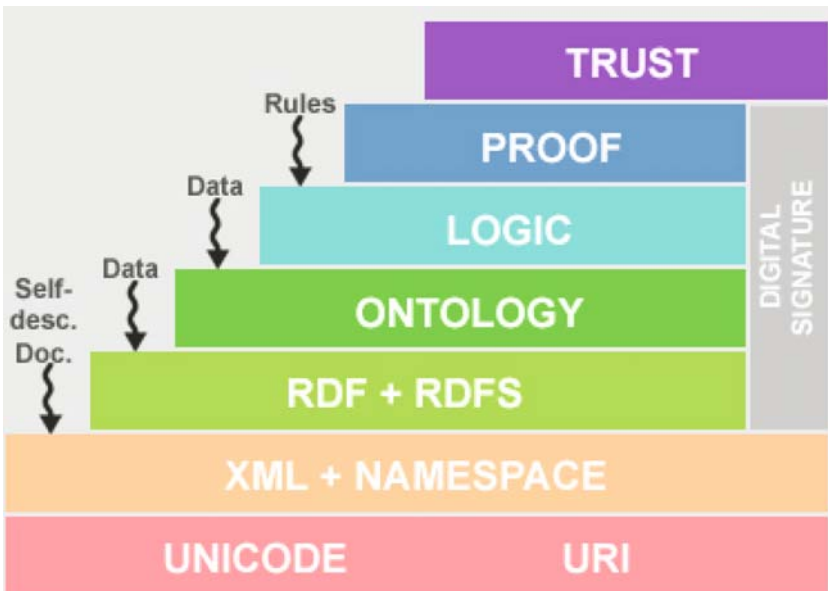


Fig. 1. Semantic web architecture

trust the given proof or not. Background information on XML, RDF, RDFS and OWL is now provided.

Extensible Markup Language (XML)

Hypertext Markup Language (HTML) is derived from Standard Generalized Markup Language (SGML), and is one of the languages used for creating web pages. The main uses of HTML are to format documents and display information. HTML does not contain any structural information. This leads to problems such as high recall and low precision in search results.

XML is a language that allows users to define tags. In HTML, all the tags are pre-defined and users cannot make any changes or create new definitions. The main aim of XML is to *extend* the markup, which ensures a uniform data exchange format between applications and supports machine processing of information. During communication between applications, application developers are required to come to a consensus on the vocabulary (tags) used in XML, otherwise there will be problems in communication and collaboration between applications. However, under such an approach, the semantics of XML documents is only accessible to the people who defined it; machines are incapable of understanding the meaning of the data.

XML is useful for data exchange between applications if the involved parties have already defined what the data is during communication. If a new communication partner becomes involved, then the model and mapping must be re-engineered [10], [16]. This is mainly because XML only structures the document. It does not provide any semantic information about the document [6]. Ultimately, while XML may be suitable for communication and collaboration in a small community where there are high levels of shared knowledge, it is not appropriate for global communications involving diverse discourse communities. Figure 2 shows a typical XML document.

Resource Description Framework (RDF)

RDF has been proposed as one way to overcome the limitation of XML that it does not support data semantics. RDF is a language based on XML and used for representing information concerning the resources available on the World Wide Web (<http://www.w3.org/RDF>). RDF can also be used to represent information about resources that cannot be directly retrieved on the Web. RDF is a framework which can be used for expressing and exchanging information between applications without loss of meaning. By using RDF, applications designers can use different kinds of RDF parsers and processing tools in their development. The information in a particular application can also be used by others. This illustrates the underlying intention of the ontology, namely knowledge reuse and sharing. RDF is a

```

<?xml version = "1.0"?>
<book isbn = "12312312">
  <title>Thomson&apos;s A Guide to Oracle 8</title>
  <author>
    <firstName>Morrison</firstName>
    <lastName>Jolinel</lastName>
  </author>
  <chapters>
    <preface num = "" pages = "3">
      A Guide to Oracle 8</preface>
    <chapter num = "1" pages = "22">
      Introduction to Client/Server Database</chapter>
    <chapter num = "2" pages = "27">
      Create and Modify Tables</chapter>
  </chapters>
</book>

```

Fig. 2. Sample XML document



Fig. 3. Graphical representation of the example statement

form of subject-predicate-object statement which is called a triple statement. The triple statement is commonly written as (s, p, o) where the subject s has an attribute p with value o , for example, “Toby is the creator of the webpage <http://www.comp.polyu.edu.hk/~cshwlam>”. In triple form, then this statement becomes (‘Toby’, <http://www.comp.polyu.edu.hk/~cshwlam>/creator, <http://www.comp.polyu.edu.hk/~cshwlam>) The property ‘creator’ is identified by the URL and the other value is a string. Figure 3 shows a graphical representation of this triple, and Fig. 4 the RDF representation.

RDF is limited since it describes resources by using named properties and values only. RDF Schema (RDFS) is a semantic extension of RDF. RDFS can describe the relationships between resources and groups of related resources such as classes, subclasses, domains and ranges. Since RDFS still lacks some important primitives, it is necessary to have another layer on top of RDF/RDFS.

Web Ontology Language (OWL)

Web Ontology Language (OWL) extends RDF(S) in order to make it easier to express meaning (semantics). Compared with XML, RDF and RDFS, OWL has a better ability to represent machine interpretable content on the

```

<?xml version = "1.0"?>
<rdf:RDF
  xmlns:rdf=
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:example="http://www.example.org/my-rdf-ns">
  <rdf:Description rdf:about=
    "http://www.comp.polyu.edu.hk/~cshwlam">
    <example:creator>
      Toby
    </example:creator>
  </rdf:Description>
</rdf:RDF>

```

Fig. 4. RDF representation of the example statement

Web (<http://www.w3.org/TR/owl-features>). There are three different sub-languages in OWL: OWL Lite, OWL DL and OWL Full (the latter being the superset of OWL Lite and OWL DL). OWL DL is based on description logic, whereas its subset OWL Lite is based on less expressive logic.

OWL Lite supports a user's need for hierarchy classification and simple constraints. OWL DL maximizes the expressiveness while retaining computational completeness; OWL Full maximizes expressiveness but has no computational guarantees. Figure 5 shows a typical example.

2.2 Agent

An agent is a complex software entity which is situated in an environment to achieve goals for the user [11]. It differs from an arbitrary program in that it is goal-orientated, persistent, reacts to the environment and is autonomous [7]. There are several different types of agents, including:

1. *Autonomous agents* – which can decide when action is appropriate without the need for human intervention.
2. *Cooperative agents* – which can interact with other agents or with a human by means of a communication language, such as the FIPA Agent-based Communication Language (<http://www.fipa.org/specs/fipa00061/>) or KQML (<http://www.cs.umbc.edu/kqml>), to solve problems collaboratively.
3. *Intelligent agents* – which have the ability to learn user preferences and to adapt to the external environment.

Recently, researchers have attempted to build intelligent agents that can mimic human intellectual behavior in problem solving, scheduling, data mining, and to generally assist humans in all their activities. Some developers have implemented various multi-agent systems, targeted towards planning and

```

...
<owl:Class rdf:ID="Custom_Tailor">
  <rdfs:subClassOf rdf:resource="#Shop_Type"/>
</owl:Class>
<owl:Class rdf:ID="Useful_Telephone_Number">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="General_Information"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Handbags_Shoes_and_Leather_Goods">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Shopping"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Hotel">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Accommodation"/>
  </rdfs:subClassOf>
</owl:Class>
</owl:ObjectProperty>
...

```

Fig. 5. RDF representation of the example statement

scheduling [18]. Compared with traditional client-server and code-on-demand technologies, systems developed using agent technology offer the following advantages [14]:

1. *Reduced network load* – Traditional distributed systems mainly rely on communication protocols involving multiple interactions to perform a task. Consequently, there is a lot of network traffic. It is possible for users to package the sequence with an agent and dispatch it to its destination host. By using this approach, network load could be significantly reduced.
2. *Reduced network latency* – Real time response is critical for control systems, and these are impacted significantly by network latencies. By using agent technology, we can dispatch an autonomous agent to the system controller. This agent can execute commands *directly* to the external environment.
3. *Asynchronous process* – Agents can be dispatched onto different hosts throughout the network. After dispatch, they become independent, and are able to operate asynchronously and autonomously.
4. *Heterogeneous* – Agents work in nominated environments. This enables them to work on different hardware and software configurations. This provides a seamless environment for heterogeneous system integration.

3 Related Work

There have been a number of research projects which have concerned themselves with tourist guidance systems. CRUMPET is a research project funded by the European Union, whose main aim is to create user-friendly and personalized mobile services for tourism [19]. CRUMPET adopted multi-agent and GPS technology to create a context-aware system. It provides tourists with two different kinds of information: static and dynamic. Static information is information collected according to the user's profile and request, whereas dynamic information is information gathered according to the user's location. CRUMPET also learns user preferences and interests to further filter irrelevant information, but does not adopt reusable knowledge from other domains.

Cyberguide [1] is a personalized tourist guide for museum visitors, which provides information relevant to the user's position and orientation. There are two different types of Cyberguide: indoor and outdoor. This is similar to CRUMPET, in that it has weak support for both knowledge sharing and reasoning.

GUIDE is a context-aware tourist guidance system for the city of Lancaster which is publicly available to visitors [5]. The system supports the collection of context-specific information regarding the user's location, navigating the city using a map, and sending messages to other visitors. Instead of using GPS or infrared to determine location information, the user's geographical information is determined by receiving location messages from position base stations.

MyCampus [20] is a research project developed at Carnegie Mellon University. It is a Semantic Web environment for context-aware mobile services. The current implementation of MyCampus combines a number of technologies such as OWL, reasoning, context-aware agents and OWL Rule Extension. The system can provide location-based movie recommendations and weather information. MyCampus is user-friendly, intelligent, and fully utilizes the latest Semantic Web technologies.

A web-based virtual exhibition system (VES) based on a XML-based digital archive is reported in [15]. It incorporates a layered set of metadata of image and text artifacts. This collection of metadata helps the development and maintenance of the system content. The idea of a virtual exhibition system is similar to the proposed portal described in Sect. 4. VES uses XML for processing, and this provides syntax for structured documents but it has no semantic constraints, such as that data can only be of integer type, for instance. In this Chapter, we show how we integrated a tourist information portal with the latest Semantic Web Technology.

4 Ontology-Based Tourist Guide

In this Section, we provide details of the ontology-based tourist guidance system *iJADE FreeWalker*. In Sect.4.1, we describe the *iJADE* (intelligent Java-based Agent Development Environment) framework. In Sect.4.2, we describe how the travel ontology is constructed. In Sect. 4.3, we describe *iJADE FreeWalker*. The system architecture of *iJADE FreeWalker* is presented in Sect. 4.4.

4.1 iJADE Framework

iJADE (<http://ijadk.com>) is an intelligent agent-based development environment capable of developing a fully integrated intelligent multi-agent based system. It is a basic framework and development environment for intelligent agent-based applications. *iJADE* consists of four layers (see Fig.6):

1. *The Application Layer:* This is the uppermost layer and consists of different intelligent agent-based applications. This layer accepts data from the conscious layer and is connected to the external application.
2. *The Conscious Layer:* This intelligence layer includes a Sensory Area, a Logic Reasoning Area and an Analytical Area.
3. *The Technology Layer:* This layer provides all the necessary mobile agent implementation APIs for the development of the intelligent agent components contained in the ‘Conscious Layer’.
4. *The Supporting Layer:* This layer provides a programming language and protocols necessary to support the development of the ‘Technology Layer’.

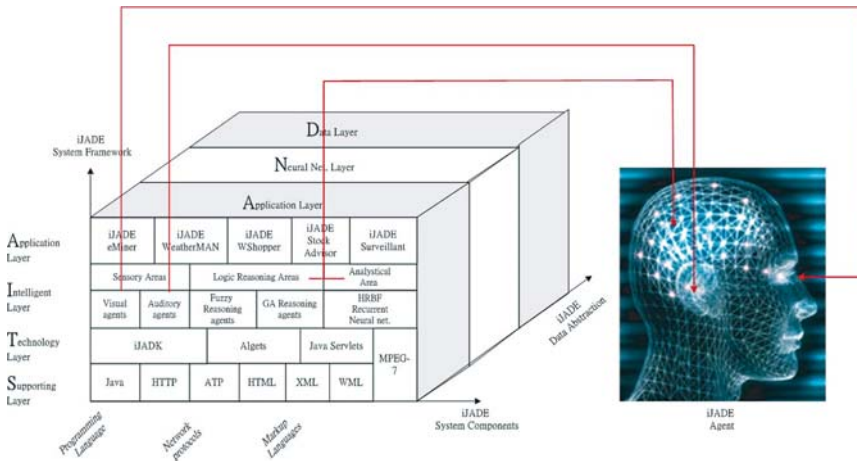


Fig. 6. *iJADE* framework

4.2 Construction of the Travel Ontology

In this Section, the method of collection and use of structural information for the ontology of the travel portal is described. Instead of modeling *all* related information, we define only the upper-level ontology, since travel ontology is related to areas such as accommodation, dining and sightseeing. The upper-level ontology provides a set of generic concepts which can be shared and reused by future users. To model this upper-level travel ontology, a number of travel guide related websites are collected and analyzed. This new travel ontology is then used to develop the tourist guidance system.

Structural Data Collection

Structural information was collected from a number of travel websites. This also included terms used in the site map and the website menu. Web developers often group related content into categories. The site structure is a common way for people to define a domain (for instance, travel). Instead of inviting an ontology expert to undertake the modeling, we collected and recorded structural information from a number of websites. Websites related to Hong Kong travel were obtained from Google Web Directory and Open Directory (<http://www.domz.org>). In Google Web Directory, there are 27 websites that relate to Hong Kong Travel (Regional>Asia>Hong Kong>Travel and Tourism>Travel Guides). In Open Directory, there are 31 websites related to Hong Kong Travel (Regional>Asia>Hong Kong>Travel and Tourism>Travel Guides). Since Google Web Directory integrates its search technology, PageRank [3], with Open Directory searching, some websites in Open Directory are duplicated in the Google Directory. After removing duplicates, 32 websites remained (2 being unreachable). Table 1 shows information about these websites. We visited each website and recorded the structural information. There were 153 terms. After filtering and grouping similar terms, we found the most common term to be ‘Shopping’; twelve websites contained this term. Table 2 shows the top ten most commonly used terms in Hong Kong travel-related websites. We further filtered and grouped together terms with similar meanings (See Table 3), then used this information to model our upper-level travel ontology.

Travel Ontology Design

To ensure that the ontology would be reusable and available to share with others, we modeled an upper-level travel ontology. Consequently, this ontology could be reused by different users in different parts of the world. After collecting and analyzing the structural information, we then defined an upper-level travel ontology. This upper-level ontology is used to provide a set of basic concepts (if we modeled *all* the concepts, the ontology would be too specific and could not be reused by others). The travel ontology contains seven main

Table 1. Websites related to Hong Kong travel guides (†= dead link)

Website Name	Web Address (URL)
1 Lonely Planet Hong Kong	http://www.lonelyplanet.com/dest/nea/hong.htm
2 Footprint Guides Hong Kong	http://www.footprintguides.com/Hong-Kong/
3 Regi Tour	http://www.regit.com/regitour/hongkong/regitour.htm
4 Dr Martin Williams	http://www.drmartinwilliams.com
5 Walk The Talk	http://www.walkthetalk.hk/
6 Arthur Frommer's Budget Travel Online	http://www.frommers.com/destinations/hongkong/
7 rec.travel Guide to Hong Kong	http://www.math.toronto.edu/~joel/hongkong.html
8 Hong Kong Tourist Guide(†)	http://www.yip.com.hk/yptourist_e03/en/html/tourist_index.aspx
9 PassPlanet.com Hong Kong(†)	http://www.passplanet.com/HK/index.htm
10 Hong Kong Hotels Guide	http://www.hong-kong-hotels-guide.com/
11 12hk: The Unofficial Guide	http://www.12hk.com
12 Hong Kong Fast Facts	http://www.hkfastfacts.com/
13 Hong Kong Travel	http://www.hong-kong-travel.org/
14 Writing, Photography and Nature Tourism in East Asia	http://martinwilliams.tripod.com/index.html
15 Travelocity's Destination Guide	http://dest.travelocity.com/DestGuides/geo_main/0,1743,TRAVELLOCITY 2771,00.html
16 Stueie's Hong Kong Page	http://uk.geocities.com/expatbeamish/SB/HK/
17 Hong Kong Tong	http://www.hongkongtong.net/
18 Hong Kong Travel Guide	http://www.luketravels.com/hong-kong/
19 Hong Kong on Web	http://www.hongkongonweb.net/
20 Hong Kong Help	http://www.hongkonghelp.com/
21 Visiting Hong Kong	http://www.visitinghongkong.co.uk/
22 Hong Kong Travellers	http://home4u.hongkong.com/lifestyle/travel/hktravellers
23 Discover Sai Kung	http://www.discoversaikung.com/
24 Travallo: Hong Kong	http://www.travallo.de/laender/asia/china/hongkong.html
25 Worldsurface.com - Hong Kong Guide	http://www.worldsurface.com/browse/location-country.asp?locationid=112
26 BootsnAll	http://www.BootsnAll.com/asiatravelguides/hk/hk.shtml
27 CNN City Guides: Hong Kong	http://www.whatsontheplanet.com/wow/ptnr/cnn/page.jsp?fx=destination&loc_id=147486&xml_set=wow.city
28 Asia Friends Network Hong Kong Tourism	http://www.countries.asiafriendsnetwork.com/HongKong/
29 Explore Sai Kung	http://www.exploresaikung.com/
30 Hong Kong Streets	http://www.hkstreet.com/
31 I Love Hong Kong	http://free.hostdepartment.com/i/ihearthk/
32 UnRealHong Kong.com	http://www.unrealthongkong.com/

Table 2. Ten most commonly used categories in HK travel guide websites

Rank	Category Term	Frequency
1	Shopping	12
2	Hotel	6
3	Getting Around	6
4	Link	6
5	Food	5
6	History	5
7	Attraction	5
8	Festival	5
9	Accommodation	5
10	Transportation	4

Table 3. The ten most commonly used categories after filtering and grouping

Rank	Category Term	Frequency
1	See/Sightseeing/Sight/Spots/Unique Sights/Interesting Places/Attractions/Landmarks/Places to Visit/Getting Around/Go Around	18
2	Food/Criuses/Restaurant/Dining/Eat/Bars and Restaurants/Eating and Drinking/Food and Drink	16
3	Shopping/Shop/Shopping and Malls/Buy	15
4	General information/General/Overview/General HK Info/City facts and info/Country info/History	14
5	Accommodation/Places to stay/Sleep/Hotels	13
6	News and events/News/Events/Festivals and events/Festival/Festivals and holidays/Public holidays	11
7	Transportation/Transport/Getting there	10
8	Others/Miscellaneous/Link	6
9	Nightlife/Night	5
10	Weather/Local weather	4

classes, each of which contains subclasses. Figure 7 shows details of the class relationships in this travel ontology.

To allow the ontology to be reused by others, the travel ontology is marked up using Web Ontology Language (OWL). The travel ontology is modeled using Protégé (<http://www.protege.stanford.edu>). Protégé is a free, open-source platform with a user friendly interface. It provides a set of tools for constructing domain model and knowledge-based applications with ontologies.

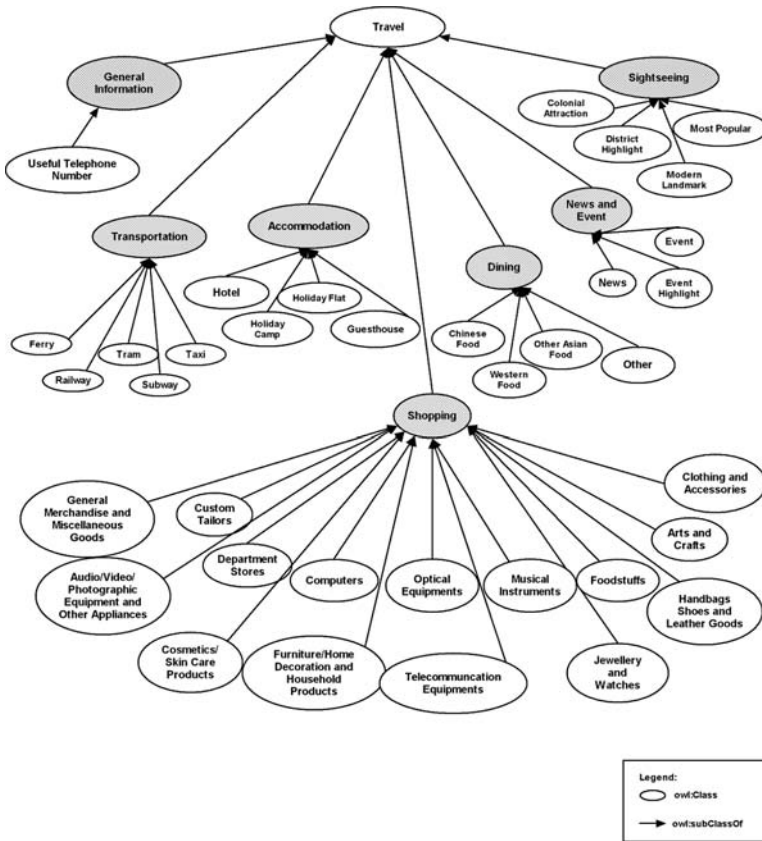


Fig. 7. Travel ontology

Figure 8 depicts a partial OWL serialization of the travel ontology. We also created three other ontologies: Cuisine, Shopping and District information. Figure 9 shows the class relationships of the cuisine ontology.

Travel Ontology Properties

There are seven main classes and a number of class properties in the travel ontology. Each class has its own subclasses and class properties, such as:

- hasDesc*: Description of the resource.
- hasName*: Name of the resource.
- hasTelephoneNumber*: Telephone number of the resource.
- hasEmailAddress*: Email address of the resource.
- hasFaxNumber*: Fax number of the resource.
- hasOpeningHours*: Opening hours of the resource.
- hasRoom*: Number of rooms in the resource.

```

...
  <owl:Class rdf:ID="Custom_Tailor">
    <rdfs:subClassOf rdf:resource="#Shop_Type"/>
  </owl:Class>
  <owl:Class rdf:ID="Useful_Telephone_Number">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="General_Information"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Handbags_Shoes_and_Leather_Goods">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Shopping"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Hotel">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Accommodation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="hasCuisine">
    <rdfs:range rdf:resource="#Cuisine_Type"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Chinese_Food"/>
          <owl:Class rdf:about="#Other_Asian_Food"/>
          <owl:Class rdf:about="#Western_Food"/>
          <owl:Class rdf:about="#Other"/>
        </owl:unionOf>
      </rdfs:domain>
    </owl:ObjectProperty>
...

```

Fig. 8. Partial OWL serialization of the travel ontology

hasStar: Rank (number of stars) of the resource.

hasURL: URL of the resource.

We used more than twenty properties to model the ontology. Some class properties, such *hasName* and *hasDesc*, are used by several different classes. The cuisine ontology has four subclasses, each subclass having a number of instances. The travel ontology ‘dining’ subclass has a property called *hasCuisine* which refers to these instances. The travel ontology has around 420 instances among seven main classes for demonstration and evaluation. Figure 10 shows an instance of the travel ontology ‘hotel’ class.



Fig. 9. Cuisine ontology

4.3 iJADE FreeWalker

iJADE FreeWalker integrates mobile agent technology and ontology to form an intelligent tourist guidance system. In general, location awareness means that the execution of the service can be dynamically adapted depending on a user’s current location. Modern location-aware mobile tourist guidance systems have been designed using two major approaches. The first is a client-server communication model that uses a remote procedure call (RPC)

```

<Hotel rdf:ID="THE_MARCO_POLO_HONGKONG_HOTEL">
  <hasTelephoneNumber rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#string"
  >21130088</hasTelephoneNumber>
  <hasFaxNumber rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#string"
  >21130011</hasFaxNumber>
  <hasEmailAddress rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#string"
  >hongkong@marcopolohotels.com </hasEmailAddress>
  <hasURL rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#string"
  >www.marcopolohotels.com </hasURL>
  <hasDistrict rdf:resource="#Tim_Sha_Tsui"/>
  <hasRoom rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#int"
  >710</hasRoom>
  <hasStar rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#float"
  >5.0</hasStar>
  <hasStreet rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#string"
  >Canton Road</hasStreet>
  <hasName rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#string"
  >The Marco Polo Hongkong Hotel </hasName>
  <hasAddress rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#string"
  >Harbour City, 3 Canton Road, Tsimshatsui, Kowloon
  </hasAddress>
  <hasStandardRoomPrice rdf:datatype=
    "http://www.w3.org/2001/XMLSchema#float"
  >1050.0</hasStandardRoomPrice>
</Hotel>

```

Fig. 10. An instance of 'hotel'

technique to transmit the location information. The second approach is to use mobile agent technology.

In the RPC approach, two separate computers communicate with each other over the network by sending and receiving messages. These messages are either requests from clients or responses from the server. By using this approach, the network traffic is quite large [12] since the responses often contain a large volume of data.

Mobile agents have several advantages over client-server communication approaches. Firstly, a mobile agent can roam the Internet with a varying

degree of autonomy. This ability to freely migrate within a network allows the agents to perform tasks on behalf of users in heterogeneous network environments. The second advantage is that agent technology allows clients and servers to interact even when the user is off-line. This is more efficient for users and can reduce network traffic. The mobile agent technique reduces network traffic and communication delay significantly, since it is not necessary to transfer a large amount of data over the network. Hence, network bandwidth is utilized in a more effective way.

Since mobile devices such as mobile phones and PDAs have limited storage, bandwidth and calculation power, a mobile agent is suitable for these handheld devices. To ensure system usability, the proposed tourist guidance system was developed using an agent platform. The location awareness mobile agent captures geographical information by using a GPS system. An ontology-based context model is proposed to represent tourist information. The context model enables knowledge sharing and context reasoning in the tourist domain.

4.4 iJADE System Architecture

iJADE FreeWalker comprises three major components, these being:

1. the iJADE FreeWalker Client,
2. the GPS Agent, and
3. the iJADE Tourist Information Center.

Figure 11 shows the system diagram of iJADE FreeWalker.

iJADE FreeWalker Client

The iJADE FreeWalker Client is a Graphical User Interface (GUI) which displays map and tourist information for the user. The client gathers the

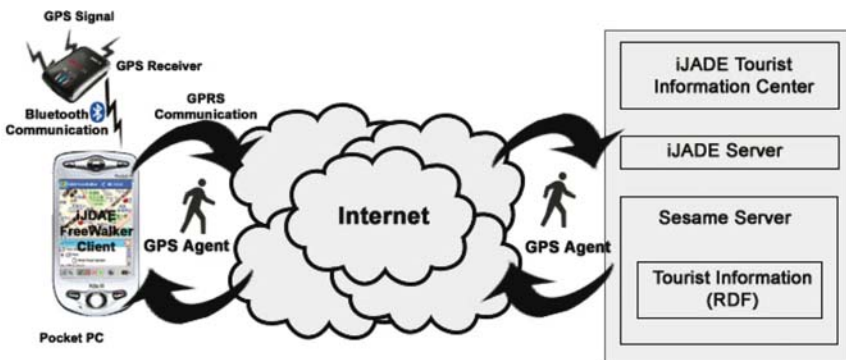


Fig. 11. iJADE FreeWalker system architecture



Fig. 12. Screen shot of iJADE FreeWalker client

user's location information by using the GPS receiver. The GPS receiver receives simultaneous GPS data and can ascertain the user's location. The GPS receiver is connected with the Pocket PC using Bluetooth. Figure 12 shows the iJADE FreeWalker screen.

GPS Agent

Mobile devices have narrow bandwidth wireless connections, and this is a critical problem when developing a mobile information retrieval system. To overcome this problem, we utilize mobile agent technology in iJADE FreeWalker. For a single request, the agent may conduct multiple interactions with several information database systems. The results are then returned to the device to reduce network load.

The GPS Agent is a software agent which can freely migrate from one host to another. The GPS Agent first captures user location information using the GPS receiver. It then migrates from the client's handheld device to a remote iJADE Tourist Information Center through GPRS (General Packet Radio Services) communication. When the GPS Agent reaches the information center, it queries the server for tourist information concerning the user's geographical location by using SPAQR (see below). The GPS Agent returns relevant tourist information to the client. The client assembles information from the GPS Agent and displays it to the user.

iJADE Tourist Information Center

The iJADE Tourist Information Centre has two components: iJADE server and Jena Framework [4]. Jena is an open source Java framework used for building Semantic Web applications. It offers a number of APIs for handling RDF, RDFS and OWL. The main purpose of using Jena is to parse and query the travel ontology. Jena searches the data from the travel ontology by using the Simple Protocol And RDF Query Language (SPARQL). Figure 13 shows an example of using SPARQL to obtain information about a 'Guesthouse'. The search returns as an RDF graph and sends it back to the tourist information server.

In Fig. 13 (lines 2–5), we have used the PREFIX keyword to declare the RDF name spaces. After declaring the name spaces, we can use the label (for example, rdfs, rdf, owl, travel) directly in the query. The SELECT clause (line 6 in Fig. 14) is used to obtain the desired information (such as name, address, district, phone number) of the Guesthouse. The WHERE clause (line 7 in Fig. 14) is used to ensure that the accommodation type of the selected results is 'Guesthouse'.

```
# Select related information about Guesthouse
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX travel: <http://www.comp.polyu.edu.hk/~cshwlam/ontology/travel.owl#>
SELECT ?URI ?NAME ?ADDRESS ?DISTRICT ?ROOM ?URL ?EMAIL ?PRICE ?FA X
?PHONE
WHERE { ?URI rdf:type travel:Guesthouse .
        ?URI travel:hasName ?NAME .
        ?URI travel:hasAddress ?ADDRESS .
        ?URI travel:hasDistrict ?DISTRICTURI .
        ?DISTRICTURI travel:districtName ?DISTRICT .
        ?URI travel:hasRoom ?ROOM .
        OPTIONAL {?URI travel:hasURL ?URL } .
        OPTIONAL {?URI travel:hasEmailAddress ?EMAIL } .
        OPTIONAL {?URI travel:hasStandardRoomPrice ?PRICE } .
        OPTIONAL {?URI travel:hasFaxNumber ?FA X } .
        OPTIONAL {?URI travel:hasPhoneNumber ?PHONE }
}
```

Fig. 13. A SPARQL example as used in iJADE FreeWalker



Fig. 14. iJADE FreeWalker

The iJADE Server acts as a communication platform. It is a container used to receive and send the GPS Agent through GPRS. When the GPS Agent arrives at the information center, it uses a SPARQL statement to parse OWL. The related tourist information is returned to the client. After processing, the client shows the relevant tourist information to the user.

5 Performance Evaluation

In this Section, we describe two sets of tests which have been used to determine the performance of iJADE FreeWalker. Our trial group comprised 30 invited candidates who assisted in judging the 'effectiveness' of the system – details are given below. We used O2 XDAlI Pocket PCs with Java Virtual Machine

as the mobile device. The GPS receiver was a Holux GR-230. The pocket PC and GPS receiver were connected together using Bluetooth. The iJADE platform and iJADE Tourist Guide are installed on the Pocket PC. Figure 14 shows iJADE FreeWalker running on the mobile device. In this study, the testers were required to visit a number of places in the Tsim Sha Tusi district of Hong Kong using iJADE FreeWalker.

5.1 Precision Test

The aim of this test was to evaluate the precision of received geographical information. In our proposed system, the GPS coordinates are used to locate the positions of both the user and of nearby landmarks. We invited 30 candidates to use the system, and then asked them to complete a survey on system accuracy. During the test, the system is connected to the GPS receiver to receive real time GPS location information. The system then gathers tourist context information using mobile agents. These results are tabulated in Table 4. Seventeen candidates thought the accuracy acceptable, 6 failed to receive the GPS data altogether, while 7 thought the results deviated from the true result.

The failure cases were analyzed and it was concluded that there are errors in receiving valid GPS data for two reasons: (i) candidates' improper use of the GPS receiver, and (ii) poor weather conditions. The performance of the GPS receiver is dependent on both the weather and the outdoor environment. A new generation of GPS receiver would most likely overcome these problems. We concluded that 57% of candidates relied on the system to provide them with correct location and tourist information by use of GPS receivers operating on these principles.

5.2 Usability Test

In this test, we evaluated the effectiveness of the modeled ontology. After the previous test, we gave 30 candidates a questionnaire to ask whether the system is potentially suitable for tourists. 17 candidates thought that

Table 4. Accuracy of the iJADE Tourist Guide system using a real GPS receiver

Is iJADE Tourist Guide accurate?	Number of Candidates
very accurate	2
accurate	10
normal	5
inaccurate	7
very inaccurate	6

iJADE FreeWalker could replace guidebooks in the future for traveling, and 13 candidates viewed iJADE FreeWalker as a subsidiary travel tool. These results show that most candidates agreed the system can help tourists find tourism-related information. The system was able to receive categorized and location-aware tourist information according to their geographical information. 25 candidates thought the categorization very detailed, whereas 5 thought the information was not precise enough. Five candidates suggested the system should provide some preference information (for instance, coupon/discount) for restaurants and shops. Since iJADE FreeWalker can run on handheld devices (such as Palm Pilots, Pocket PCs), *all* candidates agreed that the system has a high degree of mobility and flexibility.

6 Conclusion and Further Work

In this Chapter, we have showed how we modeled the travel ontology by collecting and analyzing structural information from a number of travel related websites. Using Google Directory and Open Directory, we examined 32 Hong Kong relevant travel guide websites. Commonly used terms were collected for subsequent analysis. After grouping and filtering terms with similar meanings, seven main classes were defined for use in our travel ontology. Each class had its own subclasses and properties. We had an input of around 400 instances based on the travel ontology.

To demonstrate how the ontology works, we developed an ontology-based tourist guidance system – iJADE FreeWalker. We showed how to integrate mobile agent technology, GPS technology and ontology to create a tourist guidance system. System efficiency and scalability were significantly enhanced by use of mobile agent technology. The system can run on handheld devices such as PDAs and Pocket PCs. We further demonstrated that the proposed system is highly flexible and mobile. In addition, iJADE FreeWalker supports location awareness. It can provide tourist information for sightseeing, entertainment and restaurants, according to their geographical position. Such a system is able to retrieve relevant tourist information, and furthermore filter out irrelevant information.

Experimental results showed that most users are favorably disposed towards iJADE FreeWalker. They thought the system innovative and useful for tourists. Since we are only in the early stages of development, the present functionalities of iJADE FreeWalker are somewhat limited. Based on users' comments/suggestions, we plan to incorporate the following additional functionality in the future:

1. *Voice interface* – to extend system usability and interaction, incorporation of a voice interface would be useful. Further, the system could use Natural Language Processing (NLP) to mimic a human tourist guide. A user could

chat with the tourist guide and query the relevant tourist information by using such a voice interface.

2. *Learn user preferences* – to further increase the usability and functionality of the system, we would add a Neural Network (NN) module so as to learn individual user preferences regarding tourist information.
3. *Route planning & recommendation* – to optimize the tourist route from a set itinerary having time and money constraints.
4. *Fuzzy searching* – to provide customized tourist information based on a user's preferences.

In conclusion, we have proposed a new approach to model a domain ontology. We have successfully developed both a travel ontology and a mobile ontology-based location-aware agent-based tourist guidance system. *iJADE FreeWalker* integrates GPS and agent technology to provide a location-aware tourist information retrieval system. It provides well-organized tourist information by using the aforementioned ontology. Tourists are able to access nearby tourist information anywhere, anytime, using a small, handheld device with limited computational power and network bandwidth. In the future, we intend enhancing the system by incorporating features such as a voice interface, the ability to learn user preferences, and a route recommendation facility.

Acknowledgement

The authors would like to acknowledge the partial support of research grants RGAA and 1-ZV41 from The Hong Kong Polytechnic University.

References

1. Abowd GD, Atkeson CG, Hong J, Long S, Kooper R, Pinkerton M (1997) Cyberguide: a mobile context-aware tour guide. *ACM Wireless Networks*, 3(5): 421–433.
2. Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. *Scientific American*, 284: 34–43.
3. Brin S, Page L (1998) The Anatomy of a Large-scale Hypertextual Web Search Engine. In: *Proc. 7th Intl. World Wide Web Conf.* 14–18 April, Brisbane, Australia, Elsevier, Amsterdam, The Netherlands: 107–117.
4. Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K (2004) Jena: implementing the semantic web recommendations. In: Feldman SI, et al. (eds) *Proc. 13th Intl. World Wide Web Conf.* 17–20 May, New York, ACM, New York, NY: 74–83.
5. Cheverst K, Smith G, Mitchell K, Friday A, Davies N (2001) The role of shared context in supporting cooperation between city visitors. *Computers & Graphics*, 25(4): 555–562.

6. Decker S, Melnik S, Van Harmelen F, Fensel D, Klein M, Broekstra J, Erdmann M, Horrocks I (2000) The semantic web: the roles of XML and RDF. *IEEE Internet Computing*, 4(5): 63–73.
7. Franklin S, Graesser A Is it an agent, or just a program?: a taxonomy for autonomous agents. In: Muller JP, et al. (eds) *Proc. Intelligent Agents III: Agent Theories, Architectures, and Languages*. 12–13 August, Budapest, Hungary. Springer-Verlag, Berlin: 21–35.
8. Hagras H, Callaghan V, Colley M, Clarke G, Pounds-Cornish A, Duman H (2004) Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems*, 19(6): 12–20.
9. Hunter J (2003) Enhancing the semantic interoperability of multimedia through a core ontology. *IEEE Trans. Circuits and Systems for Video Technology*, 13(1): 49–58.
10. Jannink J, Mitra P, Neuhold E, Pichai S, Studer R, Wiederhold G (1999) An algebra for semantic interoperation of semistructured data. In: Scheuerman P, et al. (eds) *Proc. 3rd IEEE Knowledge and Data Engineering Workshop*. November, Chicago, IL, IEEE Computer Society Press, Piscataway, NJ: 77–84.
11. Jennings NR, Sycara K, Wooldridge MJ (1998) A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1): 7–38.
12. Karnik NM, Tripathi AR (1998) Design issues in mobile-agent programming systems. *IEEE Concurrency*, 6(3): 52–61.
13. Khedr M, Karmouch A (2004) Negotiating context information in context aware systems. *IEEE Intelligent Systems*, 19(6): 21–29.
14. Lange DB, Oshima M (1999) Seven good reasons for mobile agents. *Communications ACM*, 42(3): 88–89.
15. Lim JC, Foo S (2003) Creating virtual exhibitions from an XML-based digital archive. *J. Information Science*, 29(3): 143–157.
16. McGuinness DL, Fikes R, Rice J, Wilder S (2000) The Chimaera ontology environment. In: *Proc. 17th National Conf. Artificial Intelligence – AAAI2000*. 30 July – 3 August, Austin, TX, MIT Press, Cambridge, MA: 1123–1124.
17. Maedche A, Staab A (2001) Ontology learning for Semantic Web. *IEEE Intelligent Systems*, 16(2): 72–79.
18. Pechoucek M et al. (2006) Agents in industry: the best from the AAMAS 2005 Industry Track. *IEEE Intelligent Systems*, 21(2): 86–95.
19. Poslad S, Laamanen H, Malaka R, Nick A, Buckle P, Zipf A (2001) CRUMPET: creation of user-friendly mobile services personalized for tourism. In: *Proc. 2nd Intl. Conf. 3G Mobile Communication Technologies*. 26–28 March, London, UK, IEEE Computer Society Press, Piscataway, NJ: 28–32.
20. Sadeh N, Chan E, Shimazaki Y, Van L (2002) MyCampus: an agent-based environment for context-aware mobile services. In: *Proc. AAMAS02 Workshop Ubiquitous Agents on Embedded, Wearable, and Mobile Devices*. 16 July, Bologna, Italy (available online at <http://autonomousagents.org/ubiquitousagents/2002/papers/papers/29.pdf> – last accessed April 2007).
21. Salton G, Buckley C (1988) Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5): 513–523.
22. Susperregi L, Maurtua I, Tubio C, Segovia I, Perez MA, Sierra B (2005) Context aware agents for ambient intelligence in manufacturing at Tekniker. *AgentLink Newsletter*, 18: 28–30.

Resources

1 Key Books

Antoniou G, van Harmelen F (2004) *A Semantic Web Primer*. MIT Press, Cambridge, MA.

Wooldridge M (2002) *An Introduction to Multiagent Systems*. Wiley, New York, NY.

2 Key Survey/Review Articles

Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. *Scientific American*, 284: 34–43.

Fensel D, Horrocks I, van Harmelen F, McGuinness D, Patel-Schneider P (2001) OIL: an ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2): 38–45.

3 Websites

OWL Web Ontology Language Overview:

<http://www.w3.org/TR/owl-features>

Resource Description Framework (RDF) – Concepts and Abstract Syntax:

<http://www.w3.org/TR/rdf-concepts/>

RDF Vocabulary Description Language – RDF Schema:

<http://www.w3.org/TR/rdf-schema/>

W3C Semantic Web:

<http://www.w3.org/2001/sw/>

4 Key International Conferences/Workshops

Intl. Conf. Autonomous Agents and Multiagent Systems (AAMAS)
Association for Computing Machinery

Intl. Conf. Knowledge-Based and Intelligent Information & Engineering Systems (KES)
KES International

Intl. Conf. Intelligent Agent Technology (IAT)
IEEE Computer Society; Web Intelligence Consortium (WIC); ACM

Intl. Conf. Web Intelligence (WI)
IEEE Computer Society; Web Intelligence Consortium (WIC); ACM

International Semantic Web Conference (ISWC)
Semantic Web Science Association

International World Wide Web Conference (WWW)
International World Wide Web Conference Committee

5 (Open Source) Software

iJADE – an intelligent Java Agent Development Kit
<http://www.ijadk.com/>

Jena – a Java framework for building Semantic Web applications
<http://jena.sourceforge.net/>

Protégé – an ontology editor and knowledge base framework
<http://protege.stanford.edu/>

6 Data Bases

DAML Ontology Library
<http://www.daml.org/ontologies>

Protégé Ontologies Library
<http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary>

Intelligent Agents

Open Source Agent-Based Modeling Frameworks

Russell K. Standish

School of Mathematics and Statistics, University of New South Wales, NSW 2052, Australia, R.Standish@unsw.edu.au

1 Introduction

1.1 Artificial Life (Alife)

Artificial Life as a field of study was inaugurated by Chris Langton, who described it as the study of man-made systems exhibiting behaviors characteristic of life. As such it is complementary to traditional biology, locating ‘life-as-we-know-it’ within the larger picture of ‘life-as-it-could-be’ [17].

The core of artificial life research involves putting together simple computational objects, or *agents* that interact to produce ‘lifelike’ behavior. The key term is *emergence*¹ of nontrivial, possibly even unexpected, behavior from the interactions of the agents.

The term ‘agent’ is often used to refer to pieces of software exhibiting autonomy, reactivity, goal orientation and persistence [9] running on a computer or migrating between hosts of a computer network. Agent-based modeling is not about these sorts of agents, *per se*. In an agent-based model, the thread of execution is passed back to the simulation environment after each agent’s method is completed, which is analogous to the way execution control is passed between independent tasks in a multitasking operating system. Nor do they often have goal orientation. What distinguishes agent-based modeling from other sorts of modeling is a focus on building models ‘bottom up’, constructing model systems from a large number of small interacting software components, that in themselves model a component of the modelled system. Thus in a model of schooling fish, each fish is individually modelled, as opposed to aggregate concepts like ‘school of fish’.

¹ The concept of emergence is a difficult one for philosophers to pin down. I can recommend [3, 10, 13] for discussions of the topic, and [32] for my own take on it. However, for the purposes of this Chapter, we can rely on our intuitive understanding of novel behavior emerging from the interactions between parts of a system.

In artificial life the focus is on the systems themselves without reference to external systems, except perhaps by analogy. The agents are often called *digital organisms* [30] to stress this fact. In *agent-based modeling* (ABM), the same methodology of constructing artificial analogues from the bottom up is applied to modeling real world systems – forest fires, stock markets, traffic, to name but a few. The agents in the model will correspond in some way to physical objects in the system being modelled.

In biology, *individual-based modeling* (IBM) [15] has become increasingly important, as inadequacies of traditional population-based modeling have become apparent. Individual-based models track individual animals or plants rather than population aggregated quantities. Individual-based models may be constructed in an agent-based way, with computational objects representing each individual organism, or alternatively each individual is represented by a collection of numbers, and the population of individuals is therefore a collection of vectors. For the sake of clarity, let us call this type of model a *vector-based IBM*. Conceptually, a vector-based IBM is little different from a specialized agent-based model, where each agent consisting of a collection of numbers, however in practice data is laid out differently in the computer's memory, which leads to substantially different performance characteristics.

In physics, the main form of individual-based modeling is molecular dynamics (MD) simulations. Here, each molecule of the physical system of interest is represented by a collection of numerical properties: position, momentum, mass, charge, and so on, and the corresponding position and momentum vectors are updated according to the laws of classical dynamics. Whilst MD simulations could be implemented in an agent-based fashion, it is rarely done due to the performance degradation experienced in doing so.

Many artificial life systems have been created over the years, of prominent note are *Tierra*, *Avida*, *Echo*, and *Framsticks*, to name but a few of the most well known. Each of these systems is implemented from the ground up in a general purpose programming language like C or C++. A typical simulation needs to implement not only the agents, but also an environment, an event generator, a means to specify input parameters, as well as visualization and analysis tools. In an attempt to introduce some commonality and code reuse between these disparate artificial life models, Langton initiated the Swarm project, which produced an agent-based modeling platform into which scientists could insert their agents into an environment adapted from a library of containers (aka 'Swarms'), and use event generators and visualization probes to analyze the progress of the simulation.

Over the years, a number of similar agent-based modeling platforms have been created, each with a differing rationale for existence. Each platform specifies a particular implementation language for the agents and has a different balance between performance, scalability, generality and usability.

This Chapter surveys open source (see Sect. 3), agent-based modeling platforms. Being open source is important, for ensuring replicability of results between different research groups, and also for auditing against implementation artifacts. This chapter does not examine commercial agent-based modeling options.

2 Applications

Agent-based models (ABMs) have been used in a wide variety of application areas, so this Section will necessarily be illustrative. ABMs are commonly used in social science settings, to test theories for why particular customs have arisen. Any recent issue of the *J. Artificial Societies and Social Simulation* will provide a number of agent based models. Economics too uses agent-based models to model the behavior of markets. Here, though, agent-based models compete with more traditional Monte Carlo techniques that model each economic agent as a simple set of state variables.

Another important area of application is traffic modeling, with vehicles in a road network being represented by software agents with intended destinations, and differing levels of behavior (such as conservative, experimental, and so on). Here the concern is quite pragmatic – what happens if a new road is created here, or another road blocked? Similar considerations have motivated research into modeling crowd behavior within restricted environments such as a sporting venue.

Grimm [12] presents a decade health check of individual based modeling in ecology. [15] argued that individual-based models had the potential to ‘unify ecological theory’, yet Grimm found that a decade on from [15], this potential had not been realized. Individual models have their place in answering particular questions inaccessible to more traditional simulation techniques, but linking the results back to theoretical concerns has not proved easy.

Finally, to artificial life, the field that inspired Swarm and later agent-based modeling platforms. Agent-based modeling is an important tool for the generation of complex life-like behaviors, others being cellular automata and boolean networks [36]. However, curiously, general purpose agent-based modeling environments such as **Swarm** and **Repast** have rarely been used in the artificial life literature, with researchers developing or making use of more special purpose simulation software such as **Avida** [1].

The following specific models are well known classical models that illustrate the sorts of modeling ABM’s are applied to. They are exemplars only; agent-based modeling as a field has grown far beyond the bounds of a single Chapter like this one.

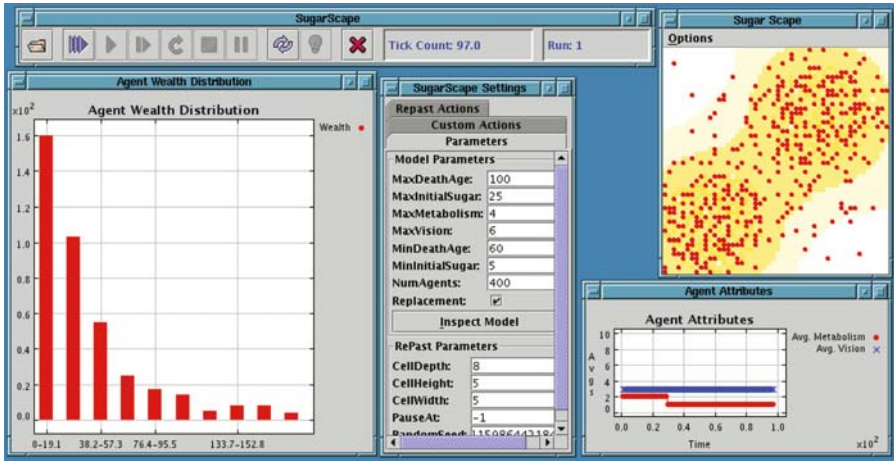


Fig. 1. *Sugarscape* model implemented in Repast

2.1 *Sugarscape*

Sugarscape [7] is a classic agent-based model of a society of agents living on a 2D grid (Fig. 1). Each agent has properties of metabolism and vision, inherited from their parents. Agents have separate requirements for ‘sugar’ and ‘spice’. They need both goods to survive and exhaustion of either will lead to death by starvation. The agents therefore need to search for these goods and accumulate them in order to survive. They do so by following a simple set of connected instructions referred to as a rule set. For example, the rule set for ‘gathering’ is:

Algorithm 1 gathering rule set

Evaluate personal stocks and determine which good is needed more urgently (preferred good).

Look around as far as vision permits and identify the site with the greatest value of the preferred good.

if the greatest value exists at multiple locations **then**
 select one randomly.

Move to that site and harvest all resources from that site.

if no value is found within the visible grid **then**
 the citizen randomly relocates to one of the furthest cells within its vision range.

Agent replacement is either handled randomly upon the death of any agent, or agents will mate according to the ‘mating’ rule set, giving rise to offspring agents. The offspring agents inherit part of their parents’ stores of sugar and spice.

Agents need both sugar and spice to survive. They have independent metabolism rates for each resource. There may arise situations where agents starve to death due to a paucity of one resource, despite having a plentiful supply of the other. *Sugarscape* allows agents to trade resources.

The ‘personality’ of the agent is an important attribute affecting their approach to trade. Personality is randomly assigned at birth and determines the trading strategy pursued by the citizen. A ‘bear’ (cautious) personality seeks to minimize surplus and will only trade the surplus commodity. A ‘bull’ (aggressive) personality seeks to maximize trades even if it involves trading the scarce commodity. The bull only trades the minimum quantity required to receive one unit of the other commodity. By maximizing trades, the bull seeks to hedge its exposure to unfair trades. For instance, the bear seeks trading partners that possess a surplus of its scarce commodity. It then attempts to trade a certain proportion of its surplus such that the quantity received in exchange can be combined with the balance of its surplus to mitigate the risk of depletion of any one commodity. The bear strategy is at risk of wild fluctuations in the exchange price, depending on the variance in the marginal rate of substitution (MRS) values of the trading partners. Since they attempt to sell all available surplus immediately, bears could end up trading all their surplus in a single unfavorable trade. The bull strategy, by trading unit amounts with as many traders as possible, seeks to average out price fluctuations and arrive closer to the equilibrium price.

Sugarscape has been implemented in *Swarm*, *Repast*, *Mason*, *Cormas* and *NetLogo*, with the *Mason* version being perhaps the most complete.²

2.2 The *Santa Fe* Artificial Stock Market

The *Santa Fe* artificial stock market was developed by [2, 26]. The market consists of a population of heterogeneous agents that buy, sell, and hold stocks and bonds (Fig. 2). An agent’s ‘buy’, ‘sell’, and ‘hold’ decisions are made on the basis of that agent’s beliefs about whether the stock’s dividend is likely to go up or down, and those beliefs are determined by a set of market forecasting rules that are continually being assessed as to accuracy. Over time an agent’s set of market forecasting rules evolve under the action of a genetic algorithm.

The market contains a fixed number N of agents that are each initially endowed with a certain sum of money. At each time step, each agent must decide whether to invest her money in a risky stock or in a risk-free asset analogous to a real-world Treasury Bill. The risk-free asset is in infinite supply and pays a constant interest rate r . The risky stock, issued in N shares, pays a stochastic dividend that varies over time. The stock’s dividend stream is an exogenous stochastic process whose present value is unknown to the agents.

² Sean Luke, private correspondence.

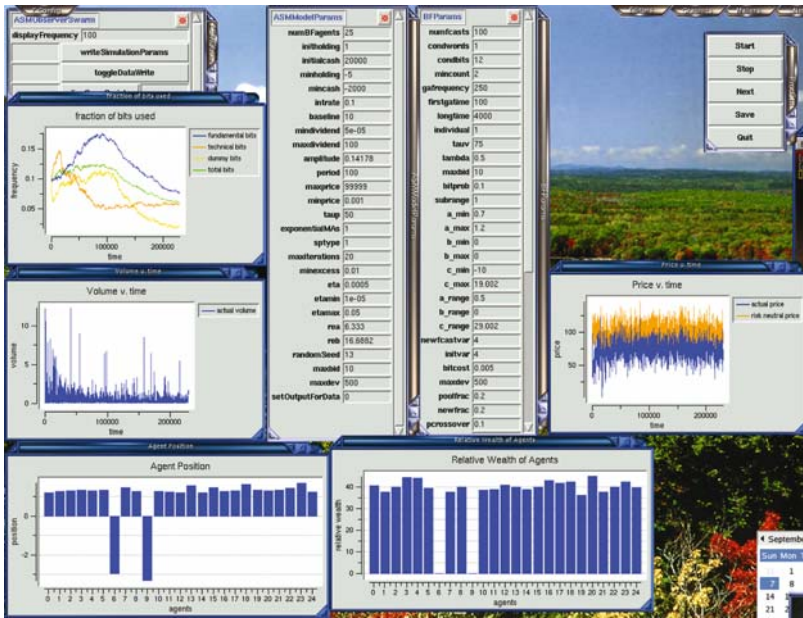


Fig. 2. A screen shot of the Artificial Stock Market

Agents apply their market forecasting rules to their knowledge of the stock’s price and dividend history to perform a risk aversion calculation and decide how to invest their money at each time period. The price of the stock rises if the demand for it exceeds the supply, and falls if the supply exceeds the demand. Each agent in the market can submit either a bid to buy shares, or an offer to sell shares – both at the current price p_t – or neither. Bids and offers need not be integers; the stock is perfectly divisible. The aggregate demand for the stock cannot exceed the number of shares in the market. The agents submit their decisions and offers to the market specialist – an extra agent in the market who controls the price so that his inventory stays within certain bounds. The specialist announces an initial trial price, collects bids and offers from agents at that price, from these data announces a new trial price, and repeats this process until demand and supply are equated. The market clearing price serves as the next period’s market price.

The agents make their investment decisions by using a set of hypotheses or rules about how to forecast the market’s behavior. At each time period, each agent considers a fixed number of forecasting rules. The rules determine the values of the variables a and b which are used to make a linear forecast of next period’s price:

$$E(p_{t+1} + d_{t+1}) = a(p_t + d_t) + b \tag{1}$$

where p_t is the trial price, d_t the dividend and a and b are the forecasting parameters. The forecasting rules have the following form:

$$\mathbf{if} \text{ (the market meets condition } D_i) \mathbf{then} \text{ (} a = k_j, b = k_l) \quad (2)$$

where D_i is a description of the state of the market and k_j and k_l are constants.

Market descriptors (D_i) match certain states of the market by an analysis of the price and dividend history. The descriptors have the form of a boolean function of some number of market conditions. The set of market conditions in each rule is represented as an array of bits in which 1 signals the presence of a certain condition, 0 indicates its absence, and # indicates ‘don’t care’. The breadth and generality of the market states that a rule will recognize is proportional to the number of # symbols in its market descriptor; rules with descriptors with more 0s and 1s recognize more narrow and specific market states. As these strings are modified by the genetic algorithm (GA), the number of 0s and 1s might go up, allowing them to respond to more specific market conditions. An appropriate reflection of the complexity of the population of forecasting rules possessed by all the agents is the number of specific market states that the rules can distinguish, and this is measured by the number of bits that are set in the rules’ market descriptors.

An example may help clarify the structure of market forecasting rules. Suppose that there is a twelve bit market descriptor, the first bit of which corresponds to the market condition in which the price has gone up over the last fifty periods, and the second bit of which corresponds to the market condition in which the price was 75% higher than its fundamental value. Then the descriptor 10##### matches any market state in which the stock price has gone up for the past fifty periods and the stock price is not 75% higher than its fundamental value. The full decision rule

$$\mathbf{if} \text{ 10##### then } (a = 0.96, b = 0) \quad (3)$$

can be interpreted as “If the stock’s price has risen for the past fifty periods and is now not 75% higher than its fundamental value, then the (price + dividend) forecast for the next period is 96% of the current period’s price.”

If the market state in a given period matches the descriptor of a forecasting rule, the rule is said to be *activated*. A number of agent forecasting rules may be activated at a given time, thus giving the agent many possible forecasts to choose from. The agent decides which of the active forecasts to use by measuring each rule’s accuracy and then choosing at random from among the active forecasts with a probability proportional to accuracy. Once the agent has chosen a specific rule to use, the rule’s a and b values determine the agent’s investment decision.

The Artificial Stock Market website is implemented in *Swarm* (both Java and Objective C versions), and available from <http://artstkmkt.sourceforge.net>.

2.3 Heatbugs

Heatbugs was originally written as a ‘demonstrator’ model for *Swarm*, illustrating the main techniques for setting up agents in a 2D grid, and having the agents interact with the environment (Fig. 3). Because it is a fairly simple, yet nontrivial model, and well documented, it has been ported to a number of other agent-based modeling environments, including *Repast* and *Mason*.

An agent in *Heatbugs* (the ‘bug’) emits a certain quantity of heat per time step into the environment, which then diffuses by the standard heat equation. Each ‘heatbug’ has a preferred temperature, so by tuning the model parameter one can see the formation of clusters of bugs that manage to heat their environment to around the desired temperature.

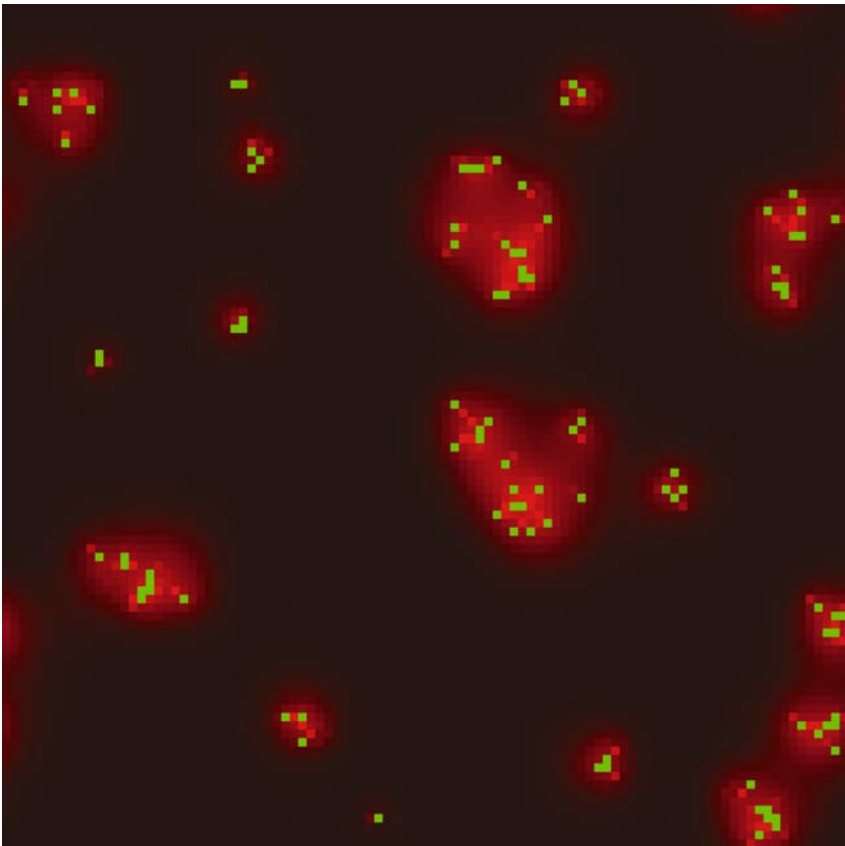


Fig. 3. Snapshot of the world display of the *Heatbugs* model, running under *Repast*

2.4 Mousetrap

Mousetrap was again a demonstrator model for *Swarm*, to illustrate the use of dynamic scheduling. It consists of a plane of loaded mousetraps. The initial event consists of a ball dropped on the central mousetrap, which releases the mousetrap sending two balls at random to other mousetraps, releasing them in turn in a chain reaction. In fact, the mousetrap model was originally introduced as popular means of conveying the idea of a nuclear chain reaction [29]. There is no concept of a time step, actions happen when caused by previous actions.

Mousetrap has been implemented in *Swarm*, *Repast* and *Mason*.

3 Software Modeling Tools

3.1 Open Source *versus* Freeware

The English language has an unfortunate ambiguity with the word ‘free’, which can mean free of restrictions or alternatively available for zero cost. This ambiguity is not always present in other languages, for example, French ‘libre/gratuit’ or German ‘frei/kostenlos’. The Free Software Foundation³ defines free software as having freedom from restrictions on how to use or distribute the software. Think of ‘free’ as in ‘free speech’, not as in ‘free beer’ [8]. Free software need not be free as in beer, as one may still have to pay distribution costs (media, handling charges, and the like), but in practice the modern internet means these charges are negligible.

One of the most important characteristics of free software is that of being ‘open source’, namely that the source code for the software is publicly available for study and improvement.

In scientific modeling, open source is crucial to allow independent validation of scientific experiments. Often, when computational experiments are replicated by a second research group, differences in behavior are observed between the original reported results, and the reproduced experiment. It becomes important to understand whether the problem is due to implementation bugs in either the original, or replicated code, or whether the published model specification is inadequate [21].

If the source code of the computational experiment is openly published along with a scientific article describing the experiment, then it becomes possible for later researchers to tease apart any anomaly that might appear. Unfortunately, it is not yet commonplace for researchers to publish the source code of their experiments, however the artificial life community encourages the practice through asking reviewers to check and comment on the availability of experimental source code.

³ <http://www.fsf.org>

What about the software components used to build the computational experiment? To help fix the magnitude of the problem, it is worth imagining being a researcher thirty years from now attempting to replicate a contemporary experiment (as McMullin found himself attempting to replicate Varela's work [21]). Assume that the source code used for the original experiment was available. To be able to re-run the original experiment, you would need a compiler for the language the experiment was coded in, and also a copy of any libraries used. Since you probably do not have access to the original hardware (how many 30-year old functional computers are you aware of?), and unless you have a functional emulator, you will need the source code for any libraries as well.

The language used in coding will probably no longer be used (Fortran and C are exceptional in being languages maintaining backward compatibility over this sort of time frame), so you may also need an open source copy of the language compiler. At very least, with well documented language standards (such as ANSI/ISO standard C, C++) and associated standard libraries, it may be possible to manually translate an existing open source program into a modern language with some sort of fidelity.

Agent-based modeling (ABM) systems fall into the 'any libraries' category. Since it is unlikely for any ABM system to be perfectly documented, nor for the scientific report to list precisely which version of the tool's specification is used, nor for the actual ABM system to be bug-free, it is vitally important that the ABM system's source code is available for perusal.

As mentioned above, it is perhaps not so important for the implementation language and standard library to be open source, provided it is one of the well documented standard languages. However, it is important that the language is freely available (as in beer), to remove any barriers to independent verification of computational models. The use of Java is a case in point. Java is developed by Sun, with a well documented language and standard library, and a free (as in beer) reference implementation available for most modern computing platforms. With the GNU gcj project,⁴ Java will become an open source option in the future, providing the language does not evolve too fast for the gcj development effort to keep up.⁵

3.2 Programming Languages

Traditional scientific modeling has been implemented using a general purpose high level language such as Fortran, C and more recently, C++ and Java. Standard libraries of numerical methods are employed where relevant, but

⁴ <http://gcc.gnu.org/java>

⁵ As this chapter was being prepared, Sun has open-sourced the core parts of its Java development environment, so even this is no longer an issue.

these tend to be oriented towards models expressible in terms of linear algebraic operations: vectors, matrices and so on. Much of a scientific code deals with reading model parameters, and reporting results. If the calculation takes more than a day or so to complete, additional code needs to be added to allow the calculation to be paused, resumed and migrated as computational resource availability varies. Further code will need to be added to distribute the calculation across multiple processors to enable computations to finish in a reasonable time. The upshot is that a sizable fraction, perhaps as much as 50% of the lines of code of a scientific application, is not directly implementing the scientific calculation, but performs these incidental tasks.

The amount of extra effort needed to obtain a functioning scientific application has led in many areas to ‘application frameworks’, where for a limited range of scientific models, users can plug in problem-specific methods. One example of this is in the area of computational fluid dynamics, where the leading packages **Fluent** and **CFX** allow users to supply subroutines coded in C or Fortran to implement specific physical models not supplied in the core functionality.

Agent-based models have not only the usual demands for scientific models, but also need interactive modes of exploration. Many phenomena of interest may only occur in specific scenarios, so it is useful to be able to restart the model from a known point, and to drill down to individual agents and their interactions, to establish what agent behavior is essential for the phenomena to occur. The process of designing an agent-based model typically involves an interactive ‘playing with the model’ stage, in which the modeler develops a feel for how the agents behave, and what might be the most significant parameters of interest, followed by a second stage of ‘parametric’ exploration to establish in what range of model parameters the phenomenon of interest occurs.

The first question to ask of any agent-based modeling platform is what language is used to implement the agents. Since agents have *behavior*, they cannot be represented just by numbers, as might be the case in other scientific computations. Some simulation systems allow a limited range of agent types to be implemented without programming knowledge (for example, **StarLogo** or **RepastPy**), but for greatest generality, agents should be implemented in a general-purpose programming language, preferably object-oriented as this matches the agent-based paradigm.

The choice of ABM systems surveyed in this Chapter is organized by this agent implementation language. I have chosen systems that use a widely implemented object oriented programming language, as this allows programmers familiar with that language to become productive in a short period of time. Also the design of a new language is a nontrivial task – using existing languages ensures that bugs have known workarounds, and that efficient implementations are available.

The most popular ABM systems are based on the Java language, an object oriented language influenced by the C/C++ language family, but from the outset designed to be a simpler language than C++, in both usability and functionality. Java compilers are freely available for most platforms, and the language and its system library publicly documented. Whilst open source Java compilers are not as mature as the closed source options; as mentioned previously, this is not such an issue for scientific computing. Nevertheless there are issues with Java's floating point model one should be aware of [16].

Because Java is widespread, and is a simpler language to learn, it is often the language of choice for learners of object-oriented programming. Its main competitors are C#, which is still somewhat tied to the MS-Windows .NET environment (although Mono⁶ is now available as an open source implementation of the .NET runtime and C# compiler), and C++, whose feature rich capabilities typically take a couple of years to master.

Java, and its .NET cousins compile to a virtual machine. By providing a uniform machine model for the compiler to target, it is easier to write portable code. However, the virtual machine needs to be emulated by the physical computer, and this introduces a performance penalty. The use of just-in-time compilation technology substantially reduces this performance penalty, though not completely. For full performance, which is just as important in ABMs as in general scientific computing, a compiler that targets the native machine is needed. For Java, there is an experimental Java front end for the GCC family of compilers called gcj. It can not only compile Java source code to Java byte code (replicating Sun's Java compiler), but also compile the byte code directly to native machine code. That said, in a direct comparison of the same simple agent based model implemented in Java using *Repast*, and C++ using *EcQab*, both the Java and C++ versions executed at the same speed (Sect. 4).

For compiling to the native machine, the main choices are Objective C (as used in *Swarm*) and C++ (as used in *EcQab*). Objective C was chosen by the *Swarm* project as being a very minimal object-oriented extension to C that would not impose too much of a learning curve on prospective users. Unfortunately, the very simplicity of this language means that object management is largely the responsibility of the programmer, and places a large burden on the programmer to get the code functioning correctly. One other downside to Objective C is that GCC is the only commonly available compiler, and GCC tends not to produce as well optimized code as do commercially available compilers (although on Intel x86 architectures, the reverse can often be true).

The other mainstream language is C++, which has consistently held the 3rd spot (behind Java and C) in the TIOBE Programming Language Community Index⁷ over the last 5 years. Vendor-optimized compiler optimizers are

⁶ <http://www.mono-project.com>

⁷ http://www.tiobe.com/tiobe_index/

available for obtaining performance, and an open source reference implementation exists (GCC). There are two main advantages of C++ over Java: more opportunities for optimizing performance and operator overloading. Operator overloading allows the creation of mathematical types like vectors, or complex numbers, and express mathematical operations on them using conventional algebraic notation (+, * and the like) instead of functional notation (`add(,)`, `mul(,)` and so forth). This is an important feature in scientific computing.

Ultimately, the choice of ABM platform should probably depend on the programming language you are most familiar with. If you are familiar with Java, then `Repast` or `Mason` would be a good choice. If C++ is your familiar language, then `Ecqab` would make a good choice. If C was your language, then you might consider `Swarm`. If programming is not your forte, then perhaps `NetLogo` might make a good choice for dipping into the world of agent-based modeling.

3.3 Reflection

As previously mentioned, a large part of a typical scientific code is involved in reading in the model's parameters, and in providing checkpoint-restart functionality for long running codes. For a rapidly evolving model, as many agent-based models are, each time the model accumulates another instance variable, or deletes one, this ancillary code needs to be updated to keep track of the changing model.

The notion of *reflection* is the ability to determine an object's structure at runtime, the names and types of all its instance variables, and lists of methods. By using reflection, this ancillary code can automatically track model changes without further burden on the programmer. Furthermore, the concept of a 'probe', or a dynamic visualizer of agents making up the model system, needs to make use of reflection to understand and represent the instance variables and methods of the object.

Unfortunately, traditional compiled languages such as C and C++ throw away this information at compile time, whereas other popular languages such as Java and Objective C have in-built reflection capability. Reflection was the other main reason for the choice of Objective C over C++ in the `Swarm` system.

For C++, `Ecqab` uses a C++ language processor called `Classdesc` [20], that emits overloaded C++ function calls that walk the structure of the object. This allows serialization of objects to a binary representation for checkpointing and other functionality, and exposure of object internals for probing.

3.4 User Interface and Scripting

All the agent-based modeling frameworks mentioned here have a GUI interactive mode with the ability to attach probes to objects, and to plot basic

statistics and display histograms of the system behavior. Once interactive development of the model is over, it is then usually desirable to turn off all graphical elements and run the model from a batch script. Only *EcQab* has this capability without recompilation, as all graphical elements are implemented as distinct script commands from those that implement the model. Other models require distinct BatchModel and ObserverModel implementations.

EcQab's script interface (which uses the TCL programming language) has the advantage that model parameters can be simply set from the script without the programmer having to write a single line of I/O code. [25] eloquently argues for the advantages of scripting interfaces to improve the plasticity of software, particularly for the development phase. With scientific codes, the development phase is often never finished. None of the other ABM packages offers a script interface, however **Repast** does offer a simple parameter file syntax, that allows just the setting of constant value parameters. Furthermore, **RepastPy** integrates the Python scripting language into **Repast** to produce a simple to use rapid development environment.

The TCL scripting language used by *EcQab* also includes a complete platform independent GUI programming environment. This technology is also used by **Swarm**, but encapsulated to hide the TCL interface from the user. Java systems have their own GUI programming environment, in fact several are possible: AWT, Swing and SWT.

3.5 Discrete Event Scheduling

Being the first package to provide explicit support for agent-based modeling, **Swarm**'s characteristics provides a benchmark for subsequent frameworks. The most important feature of **Swarm** is its discrete event scheduler – this allows for agents to register their method calls to occur at specific times during the simulation. Frequently, but not always, these actions are registered to take place periodically, defining the model's time step. An asynchronous simulation would simply consist of scheduling one event, which in turn causes further actions to happen.

3.6 Random Number Library

Most agent-based simulations rely upon streams of random numbers. Unfortunately, real sequences have notoriously low generation rates, and in any case are not reproducible, which is a problem if you want to study an effect that only occasionally makes its appearance. Usually, algorithmically generated, or pseudo-random number generators are used. However, any algorithmically generated sequence of numbers is correlated by definition, and this may or may not be a problem for the system being studied. Many evolving artificial life systems are known to exploit bugs unintentionally left by the programmer (see [35]), so it would not be surprising if evolving systems could exploit

a weak random generator. The choice of random number generator can also have a significant effect over the result in Monte Carlo simulations [27]. It is therefore desirable for an ABM framework to provide a well stocked library of different random number implementations, and allow for different generators to be swapped in easily.

3.7 Swarm

Swarm [22] is very much the ‘grand-daddy’ of agent-based modeling frameworks. It was initiated by Chris Langton as a reaction to the many and various implementations of artificial life models, complete with ‘life-support systems’ to handle I/O, initialization and visualization. The idea was to provide a software framework into which a scientist could plug just the computational representation of the model, rather than requiring the scientist to create all the necessary extra parts needed to support the computation. Just as we no longer expect scientists to grind lenses, or wire up their own custom built particle detectors, we shouldn’t expect them to have to build the tools needed to analyse their models.

When **Swarm** was originally designed, C and Fortran were the predominant scientific programming languages. Neither of these languages provide explicit object oriented programming support, and Fortran in particular was only widely available as Fortran 77 (as `f2c`, and later `g77` compilers) which lacked many modern programming features (now rectified with the Fortran 90 programming language). The GNU C compiler (`gcc`) supported two important object oriented extensions to C: Objective C and C++. Objective C had the advantage of being relatively simple for programmers to learn the object oriented syntax, and moreover had inbuilt support for reflection (see Sect. 3.3) which C++ does not (*EcQab* uses an additional C++ language processor to implement the necessary reflection functionality). So the choice of Objective C as an implementation language for **Swarm** is obvious.

At the time **Swarm** was developed, there was only one cross-platform GUI technology in the form of TCL/Tk (see Sect. 3.4). So this was adopted for the visualization components of **Swarm**. BLT, an add-on package for TCL/Tk containing implementations of plotting widgets was a particularly useful component. For similar reasons, TCL/Tk was adopted by *EcQab*, whose development also started around the same time. However, there was one key design decision made by **Swarm** developers that differs from *EcQab*. In **Swarm**, the TCL components are wrapped by Objective C classes so users of **Swarm** do not see the TCL interface. **Swarm** does not provide a scripting interface for the user – users need to provide their own. By contrast, *EcQab* makes a feature of the TCL interface – users are expected to write, or adapt existing, TCL scripts to reflect the requirements of their experiment.

To implement a **Swarm** model, one needs to implement three separate Objective C components called ‘swarms’: the `ModelSwarm`, which implements

the computational model under study; the `ObserverSwarm`, in which the experimenter must specify all the tools and visualization widgets to be used for interactive model exploration; and `BatchSwarm`, for performing extensive model surveys such as data collection or parametric surveys.

A Java interface to `Swarm` was developed, which allowed the `Swarm` library to be accessed from Java, and also Java implemented agents to be executed by a callback mechanism. Performance tends to be lacking compared with native Objective C model implementations, and more recent pure Java-based packages such as `Repast` or `Mason` have made Java- `Swarm` somewhat obsolete.

Also, an experimental DCOM interface to `Swarm` was tried by Daniels,⁸ which allowed `Swarm` to be used by any language supporting the DCOM interface,⁹ but this version of `Swarm` was never integrated into the production version.

`Swarm`'s most distinctive feature is its discrete event scheduler (Sect. 3.5), a feature that has been copied by `Repast` and `Mason`. It also blazed the way with dynamic object probes and plotting and histogramming widgets derived from the `BLT` library. The other main feature is the `Space` library, which implements a 2D grid in which agents can act.

A random number library is provided that provides the usual range of uniform generators such as linear congruential and Mersenne twister, and a number of nonuniform distributions such as the normal distribution, gamma distribution and arbitrary user specified distributions.

`Swarm` also provides a containers library – lists, maps, sets and so on. This is not needed in packages based on Java or C++, as containers are part of the latter languages' standard library.

`Swarm` has extensive documentation, as well as numerous well developed pedagogical exercises.

3.8 Repast

`Repast` [23] is a more recent Agent-Based Modeling framework, heavily inspired by `Swarm`. It comes in three different flavours: `RepastJ`, which is a pure Java-based platform; `Repast.Net`, which is implemented in C# using Microsoft's .NET environment; and `RepastPy`, a rapid application development environment based on Python and Java. Both `RepastJ` and `RepastPy` run in a Java Virtual Machine (JVM), whereas `Repast.Net` runs in a .NET virtual machine. It is unclear whether `Repast.Net` can be used in the Mono environment – [24] note the existence of Mono, but also say they expect the

⁸ <http://www.t10.lanl.gov/mdaniels/>

⁹ DCOM is a Microsoft-specific remote procedure call mechanism. Open source equivalents to it exist, such as Mozilla's XPCOM, but these have largely fallen out of favour in recent years in favour of *web services*.

vast majority of .NET code to only be run on the MS-Windows operating system.

RepastPy is meant as a reduced learning curve environment situated somewhere between NetLogo and **Repast** in functionality. Python is an object oriented scripting language that has received a lot interest in the last few years for coding scientific applications. **RepastPy** uses the JPython interpreter, which implements a Python interpreter on JVM with access to the underlying Java class libraries loaded into the JVM. **RepastPy**, in fact, makes considerable reuse of the **RepastJ** class library.

RepastJ is perhaps the most popular agent-based modeling environment in use today. This is in no small part because of the popularity of the Java programming language, but also because it is a pure Java platform (so less complex to use than Java **Swarm**), and also because it has a few years head start on **Mason**, therefore has more comprehensive documentation, and also a larger community of users.

Repast comes with the following functionality: discrete event scheduler; a GUI controller which handles probing and interactively setting model variables, stepping and running the model; a parameter package for specifying model parameters in batch mode and/or managing parametric studies; an analysis package with plotting and histogramming, as well as some basic statistical functionality, and domain specific packages to handle 2D spatial grids, genetic algorithms, neural networks, support for Geographical Information System (GIS) databases and some support for network modeling (classes for representing nodes and edges of a network).

Repast is distributed with the Colt numerical library, which includes an impressive array of random number generators.

The documentation consists a series of ‘how-to’s, relatively informal documents describing how to do one or two specific things. There are a number of example models which are good as starting templates for a new user. It is relatively simple to get the example models running in a GUI interactive mode, but not so easy to find out how to run models in a batch setting. There is a ‘-b’ option that can be passed on the command line to disable the overhead of the GUI simulation controller, however any visualization built into the model will continue to display unless the model has been coded with an explicit parameter that disables graphical output. There is no explicit support for model checkpointing, but since Java natively supports serialization, a competent Java programmer should be able to add this functionality.

3.9 Mason

Like **Repast**, **Mason** is a 100% Java simulation platform that provides the usual array discrete event engines, probes and plotting widgets [19]. Its claimed

strength lies in support from the outset for large scale modeling, with more optimized data structures, support for checkpointing and running of multiple batch runs. In my timing experiments (Sect. 4), **Mason** outperformed **Repast**, which at least backs up that claim. It also has extensive support for 3D calculations, something that is a little weak in **Repast**.

However, documentation is a weakness with **Mason**. It is not immediately obvious how one performs batch experiments, for example. Presumably one has to make specific allowances for this when coding the model, just as in **Swarm** and **Repast**. **Mason** is also a newer platform than **Repast**, hence it hasn't attracted as large a community of users as **Repast**.

3.10 **EcQab**

EcQab originally started life as a special purpose framework for hosting a single model written in C++, the Ecolab model [31]. Over the years it accreted several other similar types of models until by version 4 it had the ability to host an arbitrary C++ coded model [33]. The key feature needed for this was the `Classdesc` preprocessor, which effectively adds reflection to C++ [20]. This allows the *EcQab* framework to supply probing, scripting, checkpointing and even remote visualization of running simulations.

Whilst *EcQab* has been around for while, is reasonably mature software, and reasonably well documented, it has only been used by a small handful of groups.¹⁰ The example models provided with the source code are actually research models, so are not necessarily ideal for learning the system. One of these models is a continuous space agent-based model, which illustrates a number of important ABM techniques.

The lack of pedagogical models is currently being addressed by implementing the 'Stupid Model' of [28] in *EcQab*. The Stupid Model¹¹ has already been implemented in **Swarm**, **Repast**, **Mason** and **NetLogo**, so this is an ideal way of comparing the different environments. The implementation seemed to be about as easy as using **Repast**, and perhaps a little easier than **Swarm**. However, it lacks a special-purpose spatial library – what it does have is something far more powerful (which also means more complex to use) called *Graphcode* [34]. *Graphcode* represents a network of agents (which could be cells of a spatial grid for instance), where the agents can be distributed across a cluster of computers enabling parallel processing. This allows for scaling agent-based models to very large sizes. The jellyfish model provided in the examples has been run with several million jellyfish agents on 4–8 processor clusters.

One important aspect of agent-based modeling is the use of references. When attaching a probe to an agent, the probe object needs to maintain a

¹⁰ <http://ecolab.sourceforge.net>

¹¹ http://www.swarm.org/wiki/Software_templates

reference to its agent. When setting up schedules, lists of references to agents need to be maintained. C++ provides the notion of a ‘static reference’ (reference initialized at construction), and pointers, but the former is too inflexible, and the latter too easily invalidated. *Ecφab* provides an experimental dynamic reference counted reference class that ensures the target object is destroyed once all references to it are. This problem is a non issue in garbage collected languages like Java. Whilst garbage collection receives its share of opprobrium, for scientific modeling its performance impact is restricted to the interactive uses, when model performance is typically less important.

Unlike *Swarm*, *Repast* and *Mason*, *Ecφab* provides a scripting interface. Your C++ model object is linked to a TCL interpreter [24], with the instance variables of your model available as TCL commands. Setting model parameters are simple TCL commands. Complicated initializations can be computed – eg setting the random number seed to a function of the processor ID for instance to ensure independent random streams. The difference between batch processing and interactive processing is the presence of the GUI command, and the presence graphical visualizer commands such as plot or histogram. The net effect is a sort of halfway house between a rapid application development environment, and a fully compiled application, allowing a great deal of flexibility during the experimentation phase.

Ecφab does assume competency with C++, but even though the user needs to program in TCL, not much knowledge of TCL is needed to do most experimental tasks. Sample scripts can be readily adapted by novice TCL programmers. Advanced TCL knowledge is really only needed for novel visualization tools using the Tk canvas widget, for instance.

Whilst *Ecφab* comes with a very elementary random number library, it is interfaced to use the far more comprehensive UNURAN [14] or the GNU Scientific Library [11] random number libraries. With UNURAN in particular, the random generators can be configured by a scripting interface, and this scripting interface is exported to *Ecφab*’s TCL interface.

3.11 The Logos, StarLogo and NetLogo

StarLogo¹² [6] and NetLogo¹³ use the Logo language, which was designed as an elementary teaching language for primary school students. The frameworks are simple and easy to use, so are recommended for users with little or no programming experience. However, the environments are often considered too simple for realistic research models. That said, [29] noted that NetLogo was sufficiently rich for them to be able to implement their pedagogical Stupid Model and that these environments should not be discounted completely for

¹² <http://education.mit.edu/starlogo/>

¹³ <http://ccl.northwestern.edu/netlogo>

scientific research applications [28]. Of the three logo environments, NetLogo is the richest.

Both StarLogo and NetLogo are available for the Java Virtual Machine, and StarLogo has recently been released as a Java open source code version called OpenStarLogo. NetLogo is not open source. In reviewing the logos for this Chapter, I was unable to build OpenStarLogo (on Linux), but both compiled Logos (Star- & Net-) had functional shell scripts for starting the simulator from the unix command line. Nevertheless, Logo is an interpreted code, with the interpreter running inside Java's virtual machine, so the modeling environments will be constrained in terms of performance.

3.12 Cormas

Cormas [5] is an agent-based modeling platform written in Smalltalk that is mature, and has been used to implement a reasonable number of different models. The Smalltalk code comprising **Cormas** is available through an open source license, requiring registration with the **Cormas** development team. A variety of open source and freeware Smalltalk compilers are available, which typically compile to a bytecode interpreted representation. The **Cormas** website recommends the use of the Visual Works Smalltalk compiler from Cincom, which is available for MS-Windows, Mac-OS/X and Linux. It is unclear whether **Cormas** is ANSI Smalltalk standards compliant, or requires specific features of the Visual Works compiler.

To get started with **Cormas** requires downloading a hefty amount of software – the Visualworks environment ISO image is around 600MB. However, once downloaded, the installation of Visualworks, and then **Cormas** on top of that on my Linux workstation was straightforward.

4 Performance Comparisons

Rarely have different agent-based modeling platforms been compared for performance, or ease of use, since re-implementing an existing model is a lot of effort, and people rarely have the cross-platform skills needed to do the task.

However, [28] recently performed a cross-platform study of **Swarm**, **Java Swarm**, **Repast**, **Mason**, and **NetLogo** using a simple pedagogic model (the 'Stupid Model') that is in some way representative of typical agent-based models. They structured their model in the form of a sequence of incremental steps that starts with implementing a number of agents moving around a featureless landscape at random up to a model with predator-prey interactions, and a renewable resource ('grass') that was replenished at different rates at different locations.

The present author has added to this study by implementing the Stupid Model in *EcQab*. The aim of this exercise was to show how *EcQab* could be used for implementing the sorts of models one would use *Repast* and other similar ABM platforms for, to gauge how difficult the task was from a programmer’s perspective and to compare simulation performance.

In order to be as comparable with Railsback’s exercise as possible, the current public *EcQab* release (V4.D21) was used for implementing the models from the Stupid Model specification file. For ease of use, my experience was similar to that reported by [28], in getting the first model working within about 4 hours, and each model after that being a much smaller increment. The first model took as long as it did, as the best way to represent a rectilinear space grid within Graphcode had not been determined. Aside from a specialized space library, no other needed feature was obviously missing. Table 1 compares execution times of several versions of the Stupid Model on different platforms. Versions 10 and 11 were performed in batch mode (no graphical output, no GUI control, *Mason* excepted), version 16 in GUI mode with a plot and histogram. *EcQab*’s field version uses raster rather than canvas for display, and omits the expensive histogram widget.

The stopping criteria as specified by [28] is when the maximum bug size reaches 100. Since bug growth depends on the availability of food, which itself is a function of a random number generator call, and also of the grazing history, this stopping criterion is indeterministic, since the different frameworks will perform object updates in different orderings, and hence draw different sequences of random numbers. For the purposes of inter-framework performance comparisons, the stopping criterion was changed to be a fixed number of bug updates (500).

In version 10 of the Stupid Model, bugs will randomly select a cell within their neighbourhood, and moving to it if the cell is empty, otherwise repeating the selection process. In version 11, all cells in the neighbourhood are iterated over, and the bug moves to the empty cell with the most food.

From version 12, bugs can reproduce and die according to random dynamics, so the amount of work per update step will depend on the number of living bugs. Even though these higher version models are more computationally intensive, run times cannot be compared between different platforms due

Table 1. Execution CPU times (in seconds) for several Stupid Model versions on different platforms

Version	Repast	Mason	Obj-C Swarm	<i>EcQab</i>
10	3.5	3.4	71	3.9
11	32.7	21.3	165	14.9
16	44	40.5	402	1014
field				67

to differences in the order that random numbers are generated. Hence the Stupid 16 measurements reported in Table 1 should be taken with a certain amount of salt. Nevertheless, all models executed for 1000 steps without terminating early, and that the number of Stupid Bugs was roughly the same for each platform (approximately 800–900 after the initial population explosion).

[28] made no attempt to optimize model speed, so for comparison nor was the *EcQab* model optimized. *EcQab* is the only environment that explicitly supports a batch processing mode, and [28] did not provide batch versions of their Stupid Model. Railsback's model code was modified to disable graphical updates, and CPU times used for comparison which eliminates any delay effect from having to launch the run manually with a mouse click. In comparing *EcQab* with *Repast*, *Mason* and *Objective-C Swarm*, *EcQab* was the fastest, with *Mason* and *Repast* not too much slower, but *Swarm* was substantially slower. Furthermore, in GUI mode, *EcQab* was very slow, particularly compared with the Java platforms.

All performance benchmarks were run on a 2GHz Intel Pentium M processor with 1GB memory running Slackware Linux 10.0. The Java version used for *Repast* and *Mason* was SDK 1.4.2 standard edition. The compiler used for *Swarm* and *EcQab* was GCC V3.4.3. I also did a comparison *EcQab* run using the Intel C++ compiler (ICC) 9.0, but this was more than 50% slower than the GCC compiled code. This somewhat surprising result indicates that ICC's strength lies in vectorizing loops that access data contiguously to exploit the inbuilt SSE instructions, but that for more general purpose ABM code, GCC performs better (at least on Linux!).

The source code for *EcQab* Stupid Model is available from the *EcQab* website.¹⁴

This author's observations that the Java platforms performed almost as well as *EcQab*'s C++ based one is broadly in line with other observations that Java implementations tend to be within a factor of 2 of natively compiled applications [4, 18]. The fact that Java code is obtaining comparable performance with native compiled object code indicates that just in time compilation technology has reached a comparable level of maturity compared with native code compilers. The stereotype of virtual machines having poor performance compared with native object code can be laid to rest, at least for the typically integer bound computations often seen in agent-based models. Models requiring linear algebra operations will no doubt continue to perform better with C++ implemented code. Conversely, the poor performance of the GUI mode *EcQab* is due to the graphical operations being implemented with the TCL library, which uses byte code interpretation. There will probably be some substantial wins in integrating *EcQab*'s C++ technology with the *Repast* or *Mason* execution engines.

¹⁴ <http://ecolab.sourceforge.net>

5 Conclusion

Agent-based modeling frameworks have matured a lot since **Swarm** was first released in 1995. Frameworks supply much of the necessary model-independent functionality needed to get a scientific code running, and assist in exploring and debugging the model. Using a framework frees the scientific programmer to spend more time implementing the actual model.

The most important factor discriminating the frameworks reviewed here is the agent implementation language. Usually programmers have more experience in one language more than another, narrowing the choice to environments supporting the language with which they're familiar. If your language is C++, then *EcQab* is a good choice, if Java then **Repast** (although **Mason** has some interesting additional features), if C then Objective-C **Swarm**, and if one is a novice programmer, one of the Logos.

The Java frameworks (**Repast**, **Mason** and **Java Swarm**) are by far the most popular, owing to the popularity of the Java language. Whilst earlier versions of the Java Virtual Machine exhibited performance problems, the most recent versions implementing *just-in-time compilation* can get close to the performance of a well optimized C++ application.

There are very few comparative studies comparing different ABM platforms, and it can be difficult to validly compare different platforms. Programmer familiarity with one programming environment will bias ease-of-use comparisons, and indeterminacy (due to different pseudo random number generators employed) will prohibit valid performance metrics.

The Stupid Model exercise, however, at least indicates the suitability of all the surveyed ABM environments for typical ABM requirements.

Most of the platforms have a range of standard and pedagogical agent-based models implemented, some of which are described in earlier Sections. *EcQab* is the odd one out, in only supplying certain research models. There is a need for *EcQab* versions of the standard models to assist newcomers in building their own models, and to assist in cross-platform comparisons.

References

1. Adami C (1998) *Introduction to Artificial Life*. Springer-Verlag, Berlin.
2. Arthur WB, Holland JH, LeBaron B, Palmer R, Taylor P (1996) Asset pricing under endogenous expectations in an artificial stock market. In: Arthur WB, Durlauf S, Lane DA (eds.) *The Economy as an Evolving, Complex System II*. Addison-Wesley, Menlo Park, CA: 15–44.
3. Bedau MA (2002) Downward causation and the autonomy of weak emergence. *Principia*, 6: 5–50.

4. Boisvert F, Moreira J, Philippsen M, Pozo R (2001) Java and numerical computing. *Computing in Science & Engineering* (see also IEEE Computational Science and Engineering), 3(2): 18–24.
5. Bousquet RF, Bakam I, Proton H, Le Page C (1998) Cormas: common-pool resources and multi-agent systems. In: del Pobil AP, Mira J, Ali M (eds.) *Tasks and Methods in Applied Artificial Intelligence*, Lecture Notes in Artificial Intelligence 1416, Springer-Verlag, Berlin: 826–838.
6. Colella VS, Klopfer E, Resnick M (2001) *Adventures in Modeling*. Teachers College Press, New York, NY.
7. Epstein JM, Axtell RL (1996) *Growing Artificial Societies: Social Science From the Bottom Up*. MIT Press, Cambridge, MA.
8. Free Software Foundation. *The free software definition*. (available online at <http://www.fsf.org/licensing/essays/free-sw.html> – last accessed April 2007).
9. Franklin S, Graesser A (1997) Is it an agent, or just a program?: A taxonomy for autonomous agents. In: Müller JP, Wooldridge MJ, Jennings NR (eds.) *Intelligent Agents III Agent Theories, Architectures, and Languages*, Lecture Notes in Computer Science 1193, Springer-Verlag, Berlin: 21–35.
10. Fromm J (2004) *The Emergence of Complexity*. Kassel University Press, Germany.
11. Galassi M, Davies J, Theiler J, Gough B, Jungman G, Booth M, Rossi F (2005) *GNU Scientific Library Reference Manual (revised 2nd ed.)*. Network Theory Ltd, Bristol, UK.
12. Grimm V (1999) Ten years of individual based modeling in ecology: what have we learned and what could we learn in the future? *Ecological modeling*, 115: 129–148.
13. Holland JH (1997) *Emergence: From Chaos to Order*. Addison Wesley, Reading, MA.
14. Hörmann W, Leydold J, Derflinger G (2004) *Automatic Nonuniform Random Variate Generation* (Statistics and Computing Series). Springer-Verlag, Berlin.
15. Huston M, DeAngelis E, Post W (1988) New computer models unify ecological theory. *Bioscience*, 38(1): 682–691.
16. Kahan W, Darcy JD How java’s floating-point hurts everyone. (available online at <http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf> – last accessed April 2007).
17. Langton CG (1988) Artificial life. In: Langton C (ed.) *Artificial Life*. Addison-Wesley, Reading, MA: 1.
18. Lewis JP, Neumann U (2003) Performance of Java versus C++. (available online at <http://www.idiom.com/~zilla/Computer/javaCbenchmark.html> – last accessed April 2007).
19. Luke S, Cioffi-Revilla C, Panait L, Sullivan K, Balan G (2005) MASON: A multiagent simulation environment. *Simulation*, 81: 517–527.
20. Madina D, Standish RK (2001) A system for reflection in C++. In: *Proc. AUUG2001: Always on and Everywhere*, 26–28 September, Sydney, Australian Unix Users Group Inc., Kensington, NSW: 207.
21. McMullin B (2004) Thirty years of computational autopoiesis: A review. *Artificial Life*, 10: 277–295.
22. Minar N, Burkhart R, Langton CG, Askenazi M (1996) The Swarm simulation system: a toolkit for building multi-agent simulations. *Technical Report WP96-06-042*, Santa Fe Institute. (available online at <http://www.swarm.org> – last accessed April 2007).

23. North MJ, Collier NT, Vos JR (2006) Experiences creating three implementations of the Repast agent modeling toolkit. *ACM Trans. Modeling and Computer Simulation*, 16: 1–25.
24. Ousterhout JK (1994) *TCL and the Tk Toolkit*. Addison Wesley, Reading, MA.
25. Ousterhout JK (1998) Scripting: Higher-level programming for the 21st century. *IEEE Computer*, 31(3): 23–30.
26. Palmer RG, Arthur WB, Holland JH, LeBaron B, Tayler P (1994) Artificial economic life: a simple model of a stock market. *Physica D*, 75: 264–274.
27. Parisi G, Rapuano F (1985) Effects of the random number generator on computer simulations. *Physics Letters B*, 157: 301–302.
28. Railsback SF, Lytinen SL, Jackson SK (2006) Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82: 609–623.
29. Rathgeber HD (1963) Mousetrap model of chain reactions. *American J. Physics*, 31: 62.
30. Ray T (1991) An approach to the synthesis of life. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds.) *Artificial Life II*. Addison Wesley, Reading, MA: 371.
31. Standish RK (1994) Population models with random embryologies as a paradigm for evolution. *Complexity International*, 2.
32. Standish RK (2001) On complexity and emergence. *Complexity International*, 9.
33. Standish RK, Leow R (2003) EcoLab: Agent based modeling for C++ programmers. In: *Proc. SwarmFest 2003*, 13–15 April, Notre Dame, IN (available online at <http://www.nd.edu/~swarm03/Program/program.html> – last accessed April 2007).
34. Standish RK, Madina D (2003) ClassdescMP: Easy MPI programming in C++. In: Sloot PMA, Abramson D, Bogdanov AV, Dongarra JJ, Zomaya AY, Gorbachev YE (eds.), *Computational Science, Lecture Notes in Computer Science 2660*, Springer-Verlag, Berlin: 896.
35. Thearling K, Ray T (1994) Evolving multi-cellular artificial life. In: Brooks RA, Maes P (eds) *Artificial Life IV*, MIT Press, Cambridge, MA: 283–288.
36. Wuensche A (1999) Discrete dynamical networks and their attractor basins. *Complexity International*, 6.

Resources

1 ABM Platforms

Swarm

URL: <http://www.swarm.org>
Model Language: Objective C or Java
Visualizations: Plotting, Histogram, Raster
Scripting: None
Features: 2D Space, Event scheduler, Probes

Repast

URL: <http://repast.sourceforge.net>
Model Language: Java, Python and .Net (C#, etc.)
Visualizations: Plotting, Histogram, Raster
Scripting: Parameter files
Features: 2D Space, Event scheduler, Probes, GIS support

Cormas

URL: <http://cormas.cirad.fr>
Model Language: Smalltalk
Visualizations: Raster, Vector graphics, Plot, Message
Scripting: None
Features: 2D Space, Event scheduler, Probes, GIS support

Mason

URL: <http://cs.gmu.edu/~eclab/projects/mason>
Model Language: Java
Visualizations: Plotting, Histogram, Raster
Scripting: java.util.Properties (parameter files)
Features: 2D & 3D continuous, discrete or network Space, Event scheduler, Probes, Checkpointing

EcoLab

URL: <http://ecolab.sourceforge.net>
Model Language: C++
Visualizations: Plotting, Histogram, Canvas
Scripting: TCL
Features: Network Space (Graphcode), Probes, Checkpointing, Parallel programming

NetLogo

URL: <http://ccl.northwestern.edu/netlogo>
Model Language: Logo
Visualizations: Plotting, Histogram, Raster
Scripting: None
Features: 2D & 3D Space, Probes

StarLogo

URL: <http://education.mit.edu/starlogo>
Model Language: Logo
Visualizations: Plotting, Raster
Scripting: None
Features: Space, Probes

2 Discussion Fora

The forums are not platform-specific; for platform-specific fora, such as asking for programming help or discussing bugs or software improvements, go to the platform-specific web site.

Swarm Modeling:

http://www.swarm.org/wiki/Swarm:_Mailing_lists

Grey Thumb:

<http://www.greythumb.org/wiki/WikiHome>

Planet Agents:

<http://planetagents.org>

Agent-based Computational Economics:

<http://www.econ.iastate.edu/tesfatsi/ace.htm>

SwarmFest (the annual Agent-Based Modeling Conference):

http://www.swarm.org/wiki/Swarm:_SwarmFest

Complexity Science

(general complex systems discussions, occasionally ABM-related):

<http://necsi.net:8100/Lists/complex-science/List.html>

FRIAM (general complex systems discussions, occasionally ABM-related):

<http://www.friam.org/>

http://www.swarm.org/wiki/Software_templates (this page contains links to the Stupid Model specifications, and implementations in various frameworks).

Agent-Based Grid Computing

Minjie Zhang, Jia Tang, and John Fulcher

Intelligent Systems Research Centre, University of Wollongong, NSW 2522,
Australia, minjie@uow.edu.au, jt989@uow.edu.au, john@uow.edu.au

1 Introduction

Hundreds of millions of computers connected to the Internet form a computing resource pool with tremendous computational power and storage, as well as a great variety of services and content. For years, computer scientists have been chasing after the vision of a worldwide computer that can utilize all this resource. Computing Grids, as one of the emerging technologies that aim at making the above vision a reality, have generated enormous computing power for scientific research and have:

“incrementally scaled the deployment of relatively sophisticated services and application, connecting small numbers of sites into collaborations engaged in complex scientific applications” [20].

As the scale of systems increases, Grid computing is now facing and addressing problems relating to high autonomy and heterogeneity, intermittent availability, and dynamic and variable factors, which we call ‘open environments’.¹

Computing Grids have emerged as a major scientific computation platform for scientists and researchers. They have evolved from host-centric supercomputing and centralized cluster computing, and more recently have embraced web services. Computing Grids can provide seamless integration for large numbers of interconnected computers, applications, and users, as an alternative to centralized computing technologies. Grid computing provides Internet-scale resource sharing, selection, and aggregation through its managed, distributed computing resources.

The sheer number of desktop systems today makes the potential advantages of inter-operability between desktops and servers into a single Grid system quite compelling. However, such commodity systems exhibit significantly

¹ <http://www.objs.com/agility/tech-reports/990623-characterizing-the-agent-grid.html>

different properties compared with server-based Grids. They are usually highly autonomous and heterogeneous systems, and their availability varies over time. We refer to such environments as ‘open’.

This Chapter begins with a brief introduction to the development of computing Grids, and introduces prominent Grid architectures, communication protocols, resource allocation and scheduling algorithms. It then reviews the service-oriented Grid computing architecture, its related standards, and highlights five problems associated with the deployment and application of Grids in an open environment. The remainder of the Chapter focuses on two proposed solutions to these problems. The first solution is a hybrid one, which consolidates client-server and peer-to-peer computing architectures. This solution abandons a conventional super-local Grid architecture, and is shown to be more efficient, flexible and robust in open environments.

The latter employs a two-commit scheduling strategy, in which the functionalities of the Grid are implemented as a variety of Web Services. These services are deployed to service containers, which normally run on high-end workstations and servers. One of these services is the job management service, which allows jobs to be scheduled to the back end local scheduler. Super-local schedulers limit the use of conventional (computational) Grids on the Internet due to their centralized scheduling. Our solution, by contrast, is to design each Grid as a container which includes a number of agents. These agents can make decisions regarding job scheduling and load balancing, based on local information and the dynamic status of the open environment.

A multi-purpose task model is introduced to handle state persistence and to assist with task decomposition. Furthermore, flexibility and robustness are strengthened by employing multiple agents to construct the underlying components of the Grid architecture.

Based on this hybrid solution, the second solution improves the task model so that it provides additional support for task decomposition and inter-task communication in a transparent manner. Two framework-based agent technologies are developed for message passing and routing, as well as for resource management. These frameworks, together with various intelligent and evolving mechanisms, promote adaptability and performance.

In summary, this Chapter shows that integrating peer-to-peer computing and multi-agent technologies leads to improved scalability, efficiency, flexibility, and robustness in open environments, compared with conventional Grid computing architectures.

2 Computing Grids

During the past decade, many studies have been undertaken aimed at increasing the performance of parallel systems and host-centric enterprise computing centres. However, these centralized computing technologies have not been able

to fulfil the demand for computational power and distributed collaboration in either scientific or industrial domains.

By exploiting existing centralized and distributed computing technologies to harness distributed heterogeneous computing resources, we can increase the computational capacity available to tackle scientific problems opens up new avenues of feasible computational research (further, the demand for greater computational capacity appears insatiable).

The notion of a computing Grid – ‘grid’ for short – was inspired by the electricity power grid [8, 22]. A computing Grid is

“a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across multiple administrative domains based on their (resources) availability, capability, performance, cost, and users’ quality-of-service requirements” [5].

It is “distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and in some cases, high performance orientation” [23].

The following three-point checklist can be used to determine whether a computing system can be regarded as a Grid [17]:

1. *Coordinates resources that are not subject to centralized control:* a Grid integrates and coordinates resources from different control domains. There is no global centralized structure and the system is totally distributed;
2. *Uses standard, open, general-purpose protocols and interfaces:* a Grid must use multi-purpose protocols and interfaces for communications and operations. These protocols and interfaces must be standard as well as open, so that resource-sharing arrangements can be established dynamically with any interested party. Standards are also important in providing a general-purpose interface between the Grid and clients; and
3. *Delivers a non-trivial quality-of-service:* a Grid coordinates its constituent resources to deliver various qualities-of-service. The utility of the combined system is significantly greater than that of the simple addition of all its parts.

2.1 Development of Computing Grids

The evolution of computer architecture is driven by major technological advances in processors and networks. Up to the 1970s, almost everything was done by mainframe computers. Processing in mainframes quickly became a bottleneck in information systems. Continuous investment in mainframe upgrades could not maintain efficiency under increased processing demands and is thus not cost effective. With the miniaturization of computers and the

emergence of computer networks, the Client-Server (C-S) architecture [31] was initially proposed as an alternative to conventional mainframe systems. Such an architecture shifts the processing burden to the client computer, and therefore improves overall efficiency [2]. Later, we saw the rise of LAN-based cluster computing [41,46] in the 1980s, and WAN-based metacomputing [32,50] in the 1990s. Both of these were derived from the C-S architecture, and aim at further sharing the workload through computer networks. Inspired by the electricity power grid, Grid computing further exploits cluster computing and metacomputing to Internet-scale resource sharing, selection, and aggregation.

2.2 Application-Oriented Metacomputing

Metacomputing is the predecessor of Grid computing. The rise of metacomputing stemmed from the popularity of parallel processing, which was facilitated by two major developments: Massively Parallel Processors (MPPs) and the widespread use of distributed computing. Common to both distributed computing and MPP is the notion of message passing, around which the following two systems were developed: the Parallel Virtual Machine (PVM) [27] and the Message Passing Interface (MPI) [35,44,51].

PVM aimed at exploiting a collection of networked computers of heterogeneous architecture, data format, computational speed, machine load, and network load. From the beginning, it was designed to make programming for a heterogeneous collection of machines straightforward, whereas the MPI standard was not intended to be a complete and self-contained software infrastructure for distributed computing. The main purpose of MPI was to establish a message-passing standard for portability – indeed, it provided MPP vendors with a clearly defined set of routines that they could implement efficiently and/or provide hardware support for [28].

The PVM-based LAN metacomputer at NCSA [50] was the earliest example of a nationwide metacomputing system. The purpose of building metacomputing systems was to solve computational science problems. Table 1

Table 1. Applications of Metacomputing (after [50])

	Theoretical simulation	Instrument/Sensor Control	Data Navigation
Data source	Scientific equations and mathematical model	Scientific instruments and sensors	Database
Advantages	no time or space constraints	high-speed, real-time processing	ability to handle very large databases
Example	molecular Virtual Reality, thunderstorm simulation	interactive imaging of atomic surfaces	simulation of cosmic structures

summarizes applications of metacomputing in computational science, which cut across three fundamental areas [50]:

1. *Theoretical simulation*, which can be described as using high-performance computing to solve scientific problems numerically by using scientific equations and mathematical models.
2. *Instrument/Sensor control*, in which a metacomputing system is used to manipulate real-time and interactive visualization from raw data supplied by scientific instruments and sensors.
3. *Data Navigation*, the method through which most computational science is carried out. This involves exploring large databases, and translating numerical data into human sensory input.

Condor [15] and **Legion** [33,34] are early successful general-purpose metacomputing systems. A general-purpose metacomputing system needs to be responsible for:

- transparently scheduling application components on processors;
- managing data migration, caching, transfer and coercion;
- detecting and managing faults; and
- providing adequate protection to users' data and physical resources.

Other general-purpose metacomputing systems include **Charlotte** [4], **Javelin** [10,42], **WebFlow** [1], **Gateway** [25], **CX** [7], and an early version of **Globus** [21]. A comprehensive description of existing metacomputing systems is provided in [3].

2.3 Service-Oriented Grid Computing

As already observed, the inspiration for computing Grids came from electricity power grids. It evolved from cluster computing, but with one notable distinction – namely the manner of resource management [5]. In the case of Grid computing, there is no global (centralized) structure; the system is totally distributed. In a cluster environment, by contrast, resource allocation is performed in a centralized fashion; a master/slave relationship always exists, where the master node acts as a load balance proxy or task scheduler.

The first definition of computing Grids was given in [22]. In a subsequent article, the authors stated that

“Grid computing is concerned with coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations (VOs)” [24].

The key concept is to exploit synergies that result from cooperation – the ability to share and aggregate resources among these VOs, and then to use the resulting resource pool for a specific purpose. This notion was further

developed in Open Grid Services Architecture (OGSA) [23], where a Grid was viewed as an extensible set of Grid services that may be aggregated in various ways to meet the needs of the VOs.

2.4 Convergence of Grids and Peer-to-Peer Computing

Today, the sheer number of desktop systems make the potential advantages of inter-operability between desktops and servers into a single Grid system quite compelling. Peer-to-Peer (P2P) computing [2], as another emerging computing architecture, is tackling a set of problems which overlap with Grid computing [39]. The differences between Grid computing and P2P computing originate from their usage. Computing Grids were first used for scientific computation, while P2P computing gained prominence in the context of multimedia file exchange. **Globus** [21] – the *de facto* Grid standard – was initially an umbrella project. It was designed to federate underlying workload management systems to work for collaborations. This objective destined its role to be that of middleware and a super-local architecture. On the other hand, P2P computing aims at the collaboration of massive commodity computing devices. There are no such constraints on its architecture, as with **Globus**, which makes P2P computing more flexible (and scalable). In fact, it uses the computing power at the edge of a connection rather than within a network. The Client-Server architecture does not exist in a P2P system. Instead, peer nodes act as *both* clients *and* servers – their roles are determined by the characteristics of the tasks and the system status. This architecture minimizes the workload of each node, and maximizes utilization of the overall computing resources within the network.

In contrast to the application of Grid computing in a scientific research context, P2P computing primarily offers file sharing (for example, **Napster** and **BitTorrent** [11]), distributed computation (for instance, **SETI@home** [37]), and anonymity (for example, **Publius** [53]). Although the two types of computing architectures have both conceptual and concrete distinctions, their convergence is significant:

“the vision that motivates both Grid and P2P computing – that of a worldwide computer within which access to resources and services can be negotiated as and when needed – will come to pass only if we are successful in developing a technology that combines elements of what we today call both P2P and Grid computing” [20].

Despite the aspirations of the scientific research community for Grid computing to contribute to computational science compute capacity, its server-based architecture in a local context and middleware nature in a global context limits its application in *open environments*, where the computing nodes are highly autonomous and heterogeneous, and their availability varies from time to time. An example of an open environment is the Internet, where

enormous idle resources exist, which are normally not organized in terms of providing computing power for a certain purpose.

2.5 Research Questions of Grid Computing

Research questions concerning Grid computing in open environments include:

- *What is the best way to support task decomposition, inter-task communication, and state persistence?*

These three features are essential to the Grid. With task decomposition support, a computational task – which consists of parallel subtasks – can be decomposed automatically to achieve parallelism, and therefore leads to better performance and efficiency than a sequential computing model. This support of inter-task communication and state persistence will save a great deal of time for application developers, as they will not need to write their own frameworks to support the two features.

- *What is the best strategy for resource management and task scheduling?*

Open environments are markedly different from the application domain of conventional Grids. In order to find the best way to manage resources and schedule tasks, the properties of open environments must be carefully considered. The autonomy, heterogeneity, intermittent participation and highly variable behavior of the constituents of open environments are major concerns from the perspective of resource management and scheduling.

- *How to provide compatibility and inter-operability?*

Today, hundreds of production Grids exist all over the world, on which thousands of applications are currently running. Any new Grid system must take into consideration compatibility and inter-operability with existing Grids, Grid clients and applications. With the standardization of Grid computing and the embrace of Web Services standards, it is easier to achieve compatibility and inter-operability in any new Grid, as long as it follows these standards.

This Chapter addresses these questions by:

1. Proposing multi-purpose programming models to support task decomposition, inter-task communication, and state persistence.
2. Proposing resource management and scheduling strategies to solve resource discovery, selection, allocation (and release), load balance, task execution, task monitoring, fault-tolerance, and the storage of data. We divide these issues into two areas: (i) computing architecture and (ii) resource management framework. The former probes the dispatch of tasks and service requests, while the latter works on how to record resources and match them with service requests.
3. Keeping compatibility and inter-operability in mind. The solutions presented later (Sects. 4 and 5) provide good compatibility and inter-operability with existing Grids, Grid clients and applications.

3 Grid Computing in Open Environments

As Web Services provide a standard means for communication and object invocation between clients and service providers, embracing Web Services increases Grid inter-operability. The super-local scheduling strategy is also successful in high-end computational environments, because of its flexibility in the face of various widely accepted local schedulers, such as *Condor* [15]. However in order to implement and deploy a Grid in an open environment, the autonomous, heterogeneous, and highly dynamic nature of such an environment must be carefully considered. These properties further lead to the following problems with conventional Grids:

1. Web Services Resource Framework (WSRF) was developed to complement Web Services (WS) in order to make stateless Web Services stateful. However, it can result in significant overheads on network traffic and object invocations due to transmission of WS-Resources between the client and the service host, and conversions between internal states of a service and their WS-Resource equivalents.
2. Current service-oriented architectures have poor adaptability in terms of performance, availability and scalability, as no facility is provided by current Grid systems to allow the automatic deployment of services according to both client requests and the load currently carried by the Grid.
3. Dependence on local schedulers increases the complexity of application programming in the Grid environment, as it is difficult to provide various local schedulers with a uniform programming interface that supports task decomposition, state persistence and inter-task communication.
4. The super-local resource management and scheduling strategy relies intensively on the underlying local schedulers. This two-level process leads to more complex handling of resource discovery, selection and allocation, compared with a one-level process. The lack of direct management of the computing nodes can cause unsuitable selection of resources and unbalanced loads, and therefore limits the overall performance. In addition, as new computing nodes can only join local schedulers, instead of joining the Grid directly, the scalability of local schedulers greatly affects the overall scalability of the Grid.
5. It is not feasible to introduce local schedulers into our targeted environment, as local schedulers require a relatively static and non-autonomous environment.

4 SmartGrid – A Hybrid Solution to Grid Computing in Open Environments

Five problems have been outlined in Sect.3 in relation to Grid computing in open environments. Aiming at solving these problems, in this Section we propose a hybrid solution using multiple intelligent agents [40] combined with

server-based computing architecture and P2P computing architectures. We call this service-oriented, microkernel, agent-based, rational, and transparent Grid solution ‘**smartGRID**’.

We first present a description of the overall architecture of **smartGRID** and describe the essential components and adaptive mechanisms that make it flexible and robust (Sect. 4.1). Then we introduce the novel **smartGRID** task model and proceed to demonstrate how it can not only support state persistence, but also assist the scheduling process (Sect. 4.2).

Next, we focus on the scheduling process and evolving mechanisms of **smartGRID**. Coloured Petri Nets (CPNs) [36] are extensively used to describe agent interactions and communication protocols. All evolving mechanisms are described in detail, followed by an explanation of how they can render **smartGRID** self-contained.

Finally, in Sect. 4.4 we discuss compatibility and inter-operability issues. We explain how **smartGRID** is compatible with existing Grid clients. Two methods are described in regard to the preservation of states, as well as how to use one of these to achieve task decomposition. Lastly, we discuss promising approaches to the inter-operability of **smartGRID** and existing local schedulers.

4.1 Overall Architecture and Core Components

There are three tiers in **smartGRID**: *clients*, *trackers*, and *computing nodes*, defined as follows:

Definition 1 A ‘*client*’ is a generic computing device that seeks services from the Grid using Web Services Standards.

Definition 2 A ‘*tracker*’ is a computer which performs task scheduling operations in its managed local area network (LAN).

Definition 3 A ‘*computing node*’ is the place where tasks are executed and computing occurs. A client or a tracker can act as a computing node at the same time.

Figure 1 shows the tiers in **smartGRID**. We assume that the tiers discussed in the following sections are in the same LAN.

A *tracker* maintains the following information:

1. the available computing resources (called *profile*) on each of the nodes in the *tracker*’s LAN;
2. all tasks submitted to the *tracker* (including the running tasks, and the tasks in its waiting queue);
3. the overall load of the *tracker*’s LAN; and
4. the contact information of a limited number of other *trackers*.

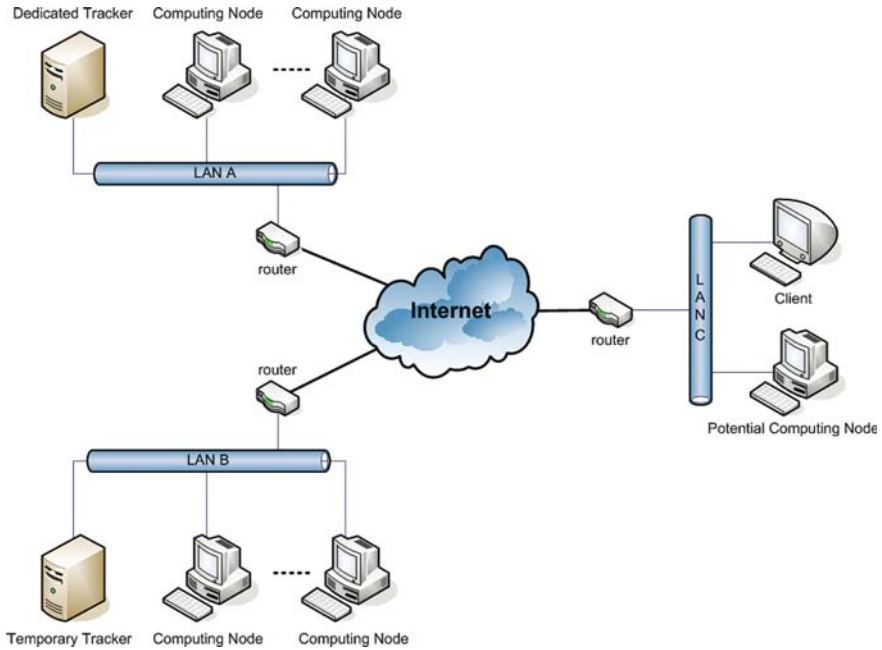


Fig. 1. smartGRID tiers

Multiple *trackers* can exist in the same LAN for performance and/or fault-tolerance consideration.

Details of the self-organizing process are provided in Fig. 2. This process allows new *computing nodes* to join smartGRID, and enables it to expand dynamically, which is essential for scalability. It also works as one of the evolution mechanisms that dynamically optimizes the configuration of smartGRID by selecting the most suitable temporary *tracker* to handle the LAN-based operations, so that the *computing nodes* can contribute their computing power to the fullest extent. The following principles cover the basic communication rules within smartGRID:

- A number of computers which have high availability, good connectivity, and good performance are selected as the ‘top-level *trackers*’ when the Grid is constructed.
- Any other computer becomes a *tracker* by registering itself to an existing *tracker*; the existing *tracker* is called the ‘parent’ of the new *tracker*. Any *tracker* therefore has at least one parent, except for the top-level *trackers*.
- *Trackers* such as the top-level *trackers* that can guarantee their availability and serviceability are called ‘dedicated *trackers*’. To become a dedicated *tracker*, a computer must register itself to an existing dedicated *tracker*, except for the top-level *trackers*.
- A *tracker* can communicate with other *trackers* for scheduling purposes.

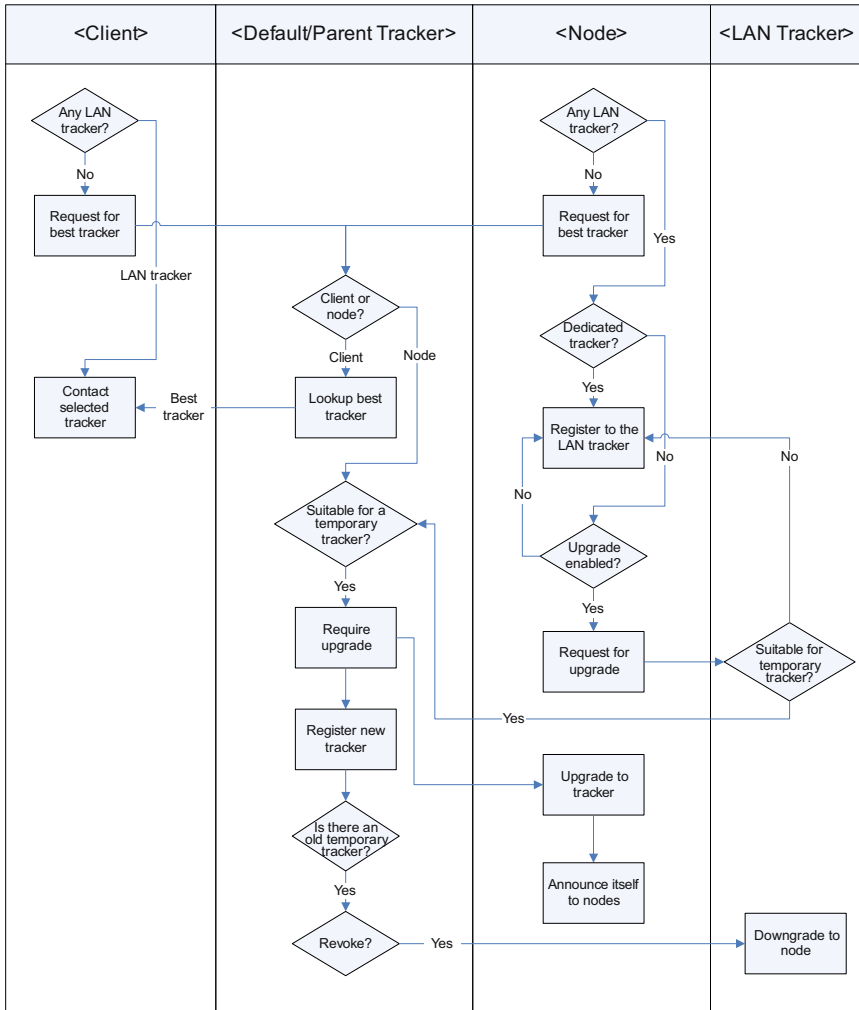


Fig. 2. The self-organizing process in smartGRID

- The *clients* or *computing nodes* only communicate with a *tracker* in the same LAN, as long as such a *tracker* exists. In case there is no existing *tracker*, a process called ‘self-organizing’ is triggered, so that the most suitable *tracker* can be returned to the *client* or the *computing nodes*.
- When a new *computing node* joins smartGRID and no *tracker* exists in its LAN, the node will be upgraded to a ‘temporary *tracker*’ as a result of the self-organizing process. A new *computing node* can also become a temporary *tracker* attributed to the self-organizing process, if the process selects it as a replacement for an existing temporary *tracker* in its LAN.

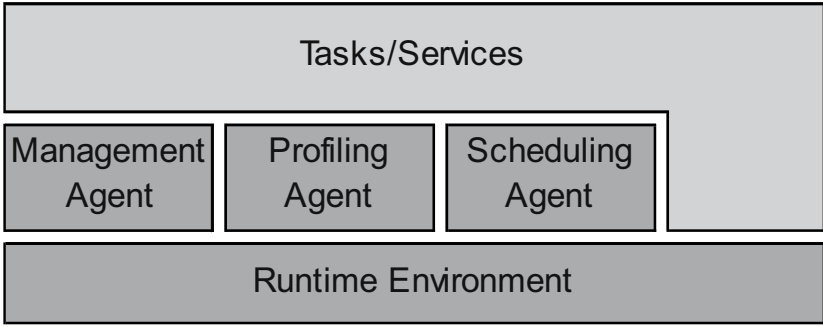


Fig. 3. Schematic view of the smartGRID container

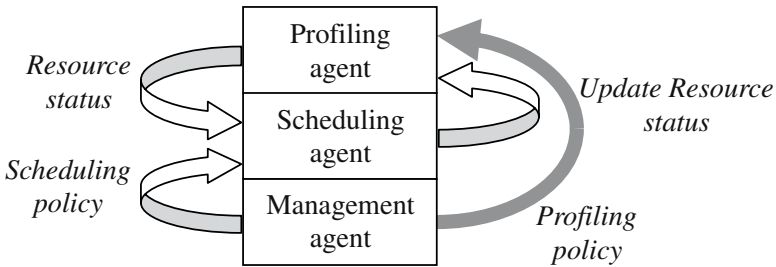


Fig. 4. Agent interactions within smartGRID

A *microkernel Grid container* runs on every *computing node* and *tracker*. The *container* serves as the runtime and managerial environment for the tasks. The *smartGRID container* consists of four components: the *Runtime Environment (RT)*, the *Management Agent (MA)*, the *Profiling Agent (PA)*, and the *Scheduling Agent (SA)*, as indicated in Fig. 3.

The *Runtime Environment* provides the runtime libraries and software components for both agents and tasks. For example, the XML parsing libraries, and implementations of the Web Services standards [57], such as Simple Object Access Protocol (SOAP), are included in the *Runtime Environment*. The *Management Agent* provides the service and managerial interface within the Grid and for the *client*. The policies and configurations are managed by the MA as well. The *Profiling Agent* gathers the status of the network, the *trackers*, the *computing nodes* and the running tasks, and provides dynamic and optimized configurations for the *Scheduling Agent*. The *Scheduling Agent* is responsible for the scheduling and management of tasks. It manages the task life cycle, and provides scheduling, fault-tolerance, and load balance services for the Grid. Figure 4 depicts agent interactions within a *smartGRID container*.

4.2 The Task/Service Model

smartGRID has a service-oriented architecture regarding its *clients*, and conforms to the Web Services (WS) standards [57]. The adoption of Web Services gives **smartGRID** good inter-operability with WS-compatible *clients* and other WS-compatible Grids. However, in order to support state persistence and task decomposition, **smartGRID** incorporates a novel task model – called Task/Service (TS) – which is a hybrid of the conventional task and service models (Fig. 5).

A TS comprises the following five components:

- TS description (TSD)
- executables
- the data
- serialization
- checkpoints

The serialization and checkpoints are automatically generated and managed by **smartGRID** when the TS is re-scheduled (in other words, when a running task is suspended). A TS without the serialization and checkpoints is called a ‘Raw TS’ (RTS). Figure 5 shows the composition of the **smartGRID** Task/Service.

The TSD has two sections: the ‘task’ section and the ‘service’ section.

Figure 6 displays the task section of the TSD, which comprises the following three sub-sections:

- The *dependencies* subsection defines the runtime components and the services that the TS depends on.
- The *scheduling policies* subsection defines:
 1. the instance policies (the minimum number of active instances, the maximum number of active instances, the minimum number of standby instances, the maximum number of standby instances) (discussed in Sect. 4.3);

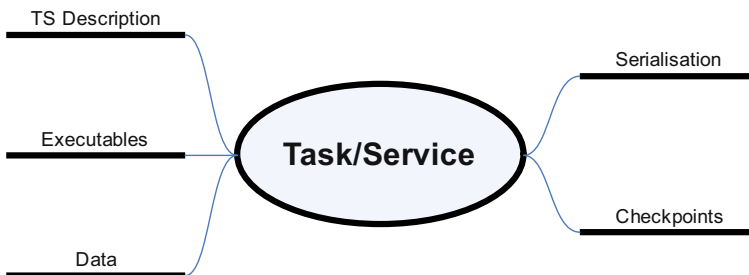


Fig. 5. Task/Service model of **smartGRID**

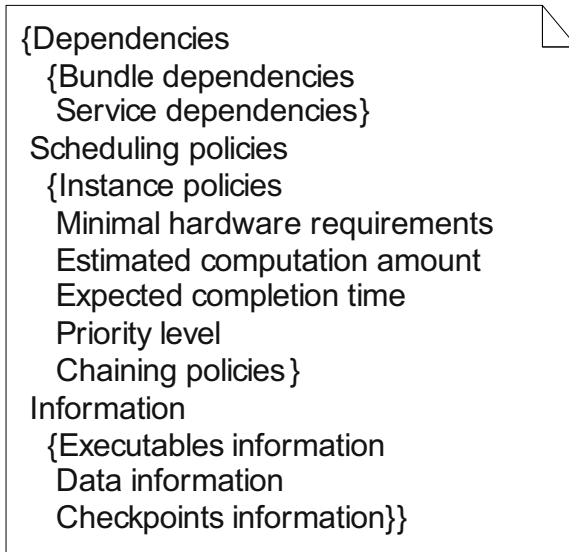


Fig. 6. Task section of the task/service description

2. the minimum hardware requirements for machine type, processor type, the amount of cycles contributed, the amount of memory contributed, and the amount of storage contributed;
 3. the estimated amount of computation;
 4. the expected completion time;
 5. the priority level; and
 6. the chaining policies (discussed in Sect. 4.3).
- The *information* subsection defines information about the executables, the data, and the checkpoints.

The service section of the TSD uses the Web Service Description Language (WSDL) [9] and WS-Resource [19] specifications to define the service interfaces and the related stateful information.

The executables are currently Java byte code files or .NET executables. The data is optional, and may come from multiple sources that are defined in the data information section of the TSD. The serialization is equivalent to the object serialization of Java [52]. It stores the runtime dynamics of any suspended TS. *smartGRID* also supports checkpoints. As not all runtime states can be preserved through the serialization process, the checkpoint mechanism is provided to give the TS a chance to save its additional runtime states as checkpoints when the TS is suspended. When re-scheduled, the TS is deserialized, and then resumed, so that the TS is able to restore its states from previous checkpoints. Checkpoints are also useful if a TS wants to roll back to its previous states.

4.3 The smartGRID Scheduling Process

The scheduling process in **smartGRID** mainly involves coordinating the agents' actions within and between the Grid containers, and constructing a self-organized evolving computing network. More specifically, there are two separate processes – to schedule the TSs to suitable computing nodes, and to balance requests and schedule the corresponding TSs to the computing nodes to serve these requests.

It has been established that Colored Petri Nets (CPNs) [36] are one of the best ways to model agent interaction protocols [12, 13, 43, 47]. In the CPN model of an agent interaction protocol, the protocol structure and the interaction policies are a network of components. The states of an agent interaction are represented by CPN places. Each place has an associated type determining what kind of data the place may contain. Data exchanged between agents are represented by tokens, whose colours indicate the value of the representing data. The interaction policies of a protocol are carried by CPN transitions and their associated arcs. A transition is enabled if all of its input places have tokens, and the colours of these tokens can satisfy the constraints that are specified on the arcs. A transition can be fired, which means the actions of this transition can occur when this transition is enabled. When a transition occurs, it consumes the input tokens as the parameters, conducts the conversation policy and adds the new tokens into all of its output places. After a transition occurs, the state of a protocol is changed. A protocol is in its *TERMINATED* state when there is no enabled or fired transition.

Task/Service Life Cycle

Figure 7 shows the states of a TS in its life cycle. When a Raw TS is submitted by a client via a *tracker's* MA, the MA checks the TS's validity. If the TS is valid, it enters the *SUBMITTED* state. A set of pre-schedule operations are then applied to the TS by the MA and SA of the *tracker*. These operations include making a backup of the submitted TS, and allocating and initializing the internal resources for the purpose of scheduling that TS, and so forth. If all operations succeed, the TS enters the *READY* state.

The *READY* state means that the TS is ready to be scheduled. In this state, the SA of the tracker uses a 'best-match' algorithm to determine whether the managed *computing nodes* of the *tracker* are suitable for the TS. If a suitable computing node is found, a schedule operation is applied. Otherwise, the SA (called *chaining source*) extracts the TSD from the TS, and passes it to the SAs of other known *trackers*. Every time the TSD passes by a *tracker*, the TTL (Time-to-Live) specified in the chaining policies of the TSD decreases by 1. If one of the *trackers* happens to be able to consume the TS according to the best-match algorithm, it contacts the source SA to transfer the TS to it. If the tracker is not able to consume the TS, it keeps passing on the TSD until the TTL equals 0. The above process is called 'chaining'.

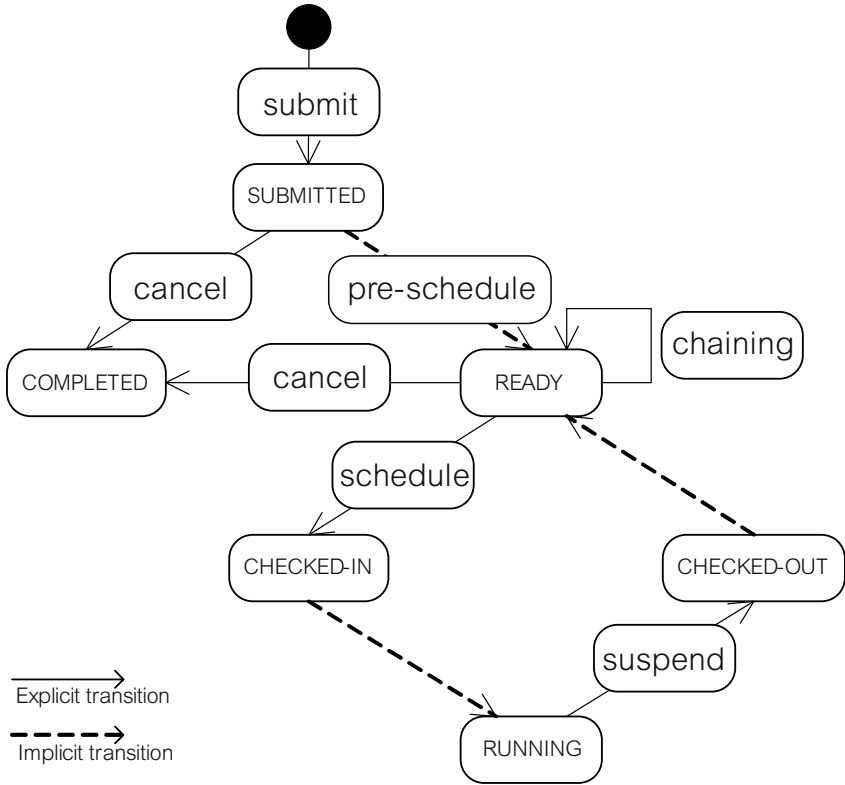


Fig. 7. States of a task/service in smartGRID

After chaining, the TS remains in the *READY* state. Chaining is the core mechanism in smartGRID to balance the loads and requests globally.

The TS enters the *CHECKED-IN* state after the schedule operation, which means that the TS is scheduled to a computing node, the executables are resolved by the runtime environment of the computing node, and the runtime dynamics and checkpoint have been restored for a suspended TS. The TS then automatically enters the *RUNNING* state until the suspend operation is applied, where the TS is serialized and suspended, and enters the *CHECKED-OUT* state. Following this, the TS is automatically transferred to the *tracker*, where the computing node registers for rescheduling. A special situation is that if the TS exits, it fires the suspend operation itself and stores the computing result whilst suspended.

Task-related Scheduling

In the smartGRID scheduling strategy, the TSs, requests, and profiles of the *trackers* and computing nodes are represented as three kinds of *tokens*. The

transition rules of these tokens are different when the *tokens* are placed in different places. The agents in *smartGRID* are responsible for allocating the *tokens* and modifying them after the transitions are fired.

The task-related scheduling process can be described as three sub-processes: scheduling within a *tracker*, scheduling between the *tracker* and the *computing nodes*, and scheduling among the *trackers*.

Scheduling within a Tracker

Figure 8 demonstrates the scheduling process with a *tracker* modelled by a CPN. There are four types of *places* defined in the CPN: Task-related places, operation places, the profile/load place, and the simulated synapse place. They are described as follows:

1. The *Raw TS* place holds the Raw TS *token*, which is received from the client.
2. The *Rejected TS* place holds the Raw TS *tokens*, which are rejected by the Check transition.
3. The *Legal TS* place holds the Raw TS *token*, which is asserted as legal by the Check transition. The legal Raw TS token may also come from the *tracker* itself, due to a re-schedule operation.

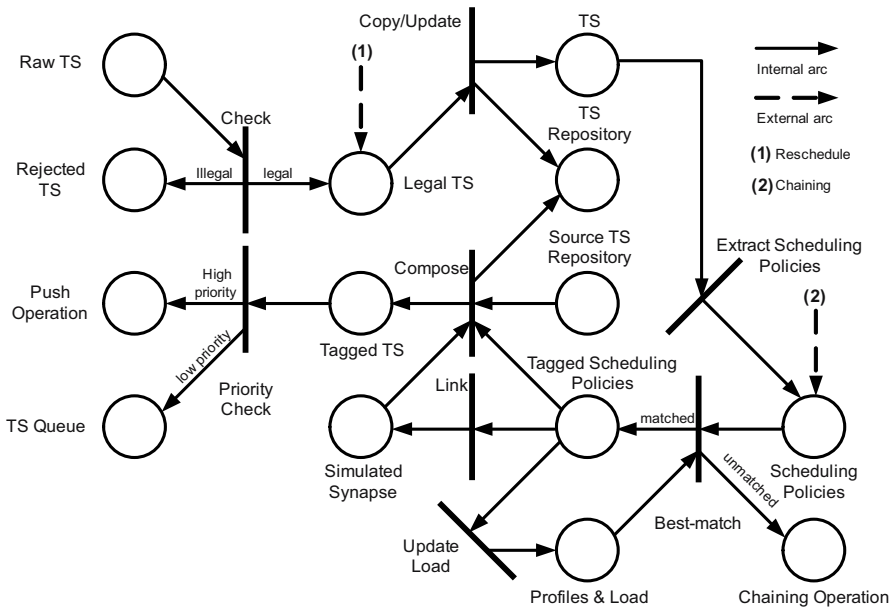


Fig. 8. Scheduling process within a tracker

4. The *TS Repository* place holds the backup *TS tokens*. A backup *TS token* is removed when the corresponding *TS* exits or moves to another *tracker* through the chaining process. A backup *TS token* is updated when the corresponding *TS* is re-scheduled.
5. The *TS* place holds the *TS token*, which is produced by the Copy/Update transition.
6. The *Scheduling Policies* place holds the scheduling policies token, which is extracted from its corresponding *TS token*. The scheduling policies token may also come from another *tracker* through the chaining process.
7. The *Profiles and Load* place holds the profile *tokens* and load *token*. Each profile token contains the information and status (called *profile*) of its corresponding *computing node*. The load *token* contains the status of the overall load of its corresponding *computing nodes*. Figure 9 depicts the scheme represented by the profile *token* and the load *token*.
8. The *Chaining Operation* place holds the unmatched scheduling policies token, which is consumed by the chaining process.
9. The *Tagged Scheduling Policies* place holds the Tagged Scheduling Policies token, which is produced by the best-match transition. The tagged *token* has *winner* tags, which contain the identifiers of the best suitable nodes ('winners').
10. The *Synapse* place holds the synapse *token*, which represents the link between the destination and the source of a chaining process.
11. The *Source TS Repository* place holds the corresponding *TS token* of the scheduling policies *token*, which is passed through the chaining process.
12. The *Tagged TS* place holds the Tagged *TS token*, which comprises the *TS token* and the Tagged Scheduling Policies *token*.
13. The *Push Operation* place holds the Tagged *TS token*, which will be 'pushed' to its corresponding *computing node*.
14. The *TS Queue* place holds the Tagged *TS tokens*, which will be 'pulled' by any of the winner *nodes*.

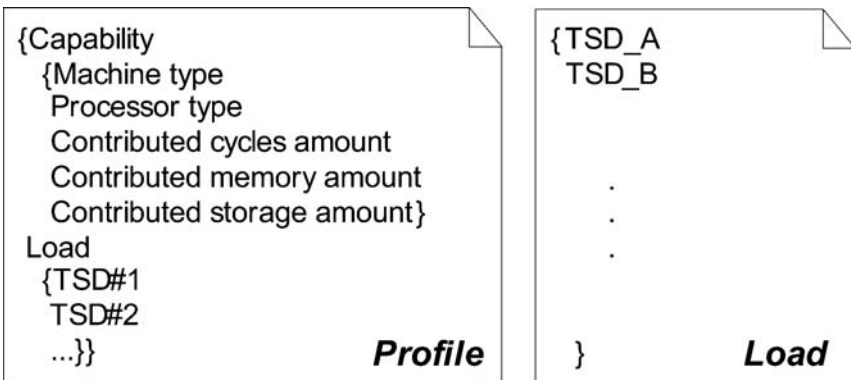


Fig. 9. Profile and Load

There are eight *transitions*, which represent eight operations. They are:

1. The *Check* transition checks the syntax of the TSD of the Raw TS *token*. It also checks whether the dependent bundles and services exist, and whether the services defined by the Raw TS conflict with the existing services (for instance, conflict due to the same service name). In addition, the Check transition converts the Raw TS *token* into the TS *token*.
2. The *Copy/Update* transition either duplicates the TS token, or updates the TS token in the TS repository place.
3. The *Extract Scheduling Policies* transition extracts the scheduling policies from the TSD.
4. The *Best-match* transition performs the best-match algorithm. Figure 10 explains the algorithm (the Profile-Aware Eager Scheduling will be discussed later).
5. The *Update Load* transition converts the scheduling policies into the computing load, and adds the load to the overall load of the *tracker*.
6. The *Link* transition connects the two end points of a chaining process. Scheduling from one *node* to another *node* within the same LAN is a special case, as a tracker is always linked with itself.
7. The *Compose* transition transfers the TS *token* from the source TS repository, updates the local TS repository, and composes the Tagged TS *token* from the TS *token* and the tagged scheduling policies *token*.
8. The *Priority Check* transition compares the priority of the tagged TS *token* with the current loads of the winners, to determine whether the *token* is ‘pushed’ to its corresponding *computing node*, or stored in a queue for the ‘pull’ operation.

Scheduling between a Tracker and its Nodes

smartGRID uses a scheduling algorithm called Profile-Aware Eager Scheduling (PAES), which is derived from eager scheduling, to schedule the TSs from the *trackers* to their managed *computing nodes*.

The eager scheduling algorithm was first introduced in *Charlotte* [4]. Its basic idea is that faster *computing nodes* will be allocated tasks more often, and if any task is left uncompleted by a slow *node* (a failed *node* is infinitely slow), that task will be reassigned to a fast *node*. In other words, it uses a ‘keep the faster nodes busy’ strategy. It also guarantees fault tolerance by using a redundant task cache with a time-out mechanism. The PAES algorithm takes the profiles of the *computing nodes* provided by the profile agent and the scheduling policies provided by the TSs into consideration when performing scheduling. In contrast to eager scheduling, it allows bidirectional scheduling operations – namely ‘pull’ and ‘push’. Figure 11 demonstrates the two operations.

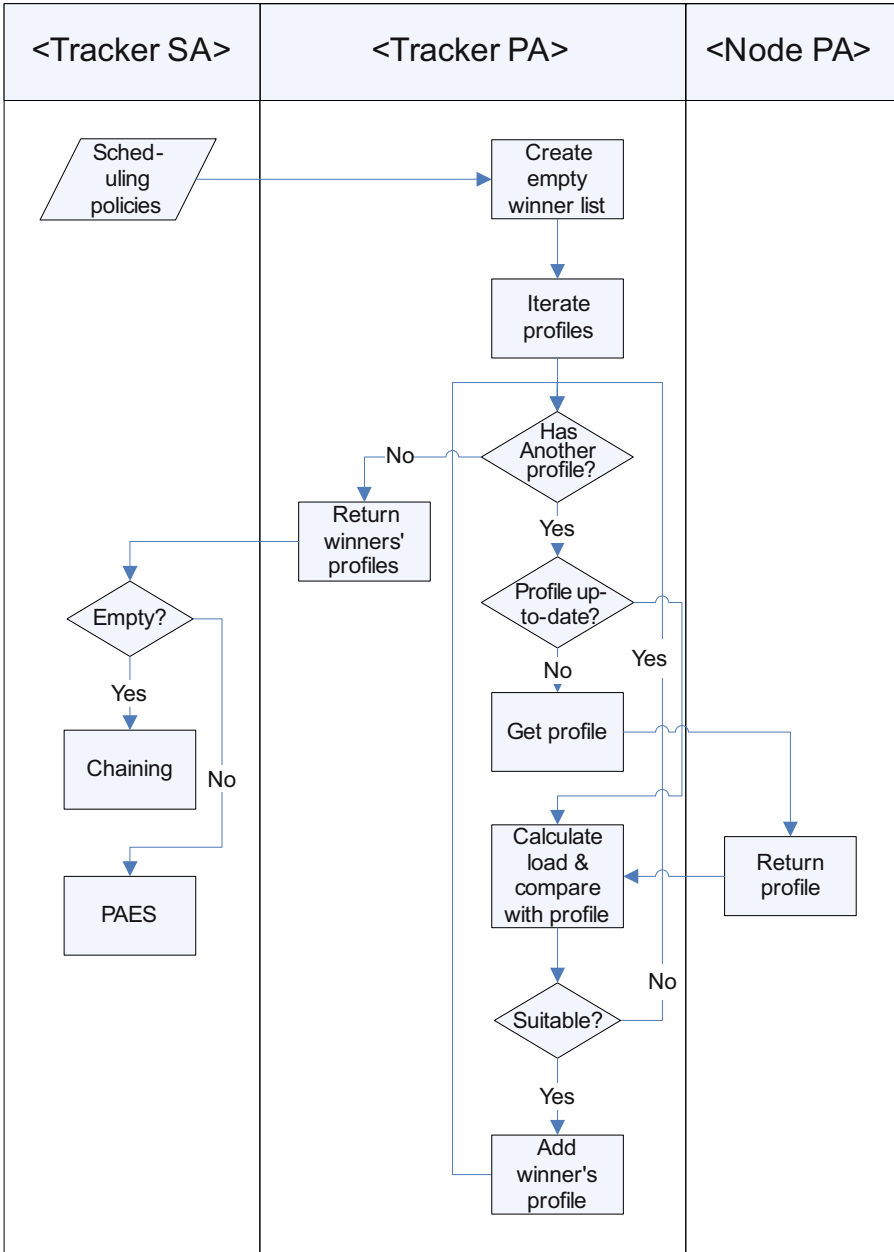


Fig. 10. The best-match algorithm

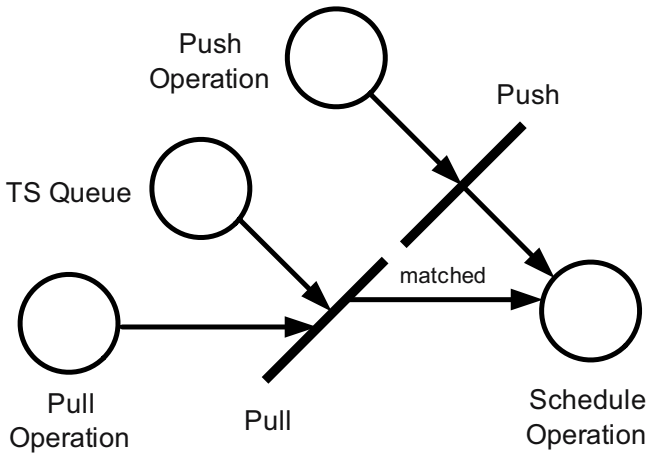


Fig. 11. *Push* and *pull* operations

The Schedule Operation place holds the TS *token*, which is scheduled to the corresponding *computing node*. The Push Operation is straightforward. The *Push* transition represents the push operation – that is it assigns the TS to one of the winners. The Pull Operation place holds the requests from the *computing nodes*. Whenever the scheduling agent of a *node* determines that it is able to run a new task, it sends a request to the *tracker*. The *Pull* transition represents the pull operation – in other words, the scheduling agent matches the *computing node* requesting the TSs with the tagged TS *tokens*. If the *node* is the winner, the TS is assigned to that particular *node*.

Scheduling among the Trackers

Trackers are linked by the chaining process, which is the core of the scheduling process among the *trackers*.

Figure 12 shows the basic chaining mechanism. There are two transitions, in the chaining mechanism, *Check* and *Send*, which check the TTL in the scheduling policies token first. If it is greater than 0, the TTL decreases by 1, and the scheduling policies with the new TTL is sent to all known *trackers*. If the TTL equals 0, the scheduling policies *token* is discarded.

Recalling Fig. 8, there is a link transition, which makes two chained *trackers* learn (that is, if *tracker-A* successfully schedules the chained TS of *tracker-B*, A and B are chained), and preserve each other's information for future chaining processes. However, if the links exist permanently, the performance of the chaining process will gradually decrease as time goes by because of the explosive numbers of links. A link must therefore be able to

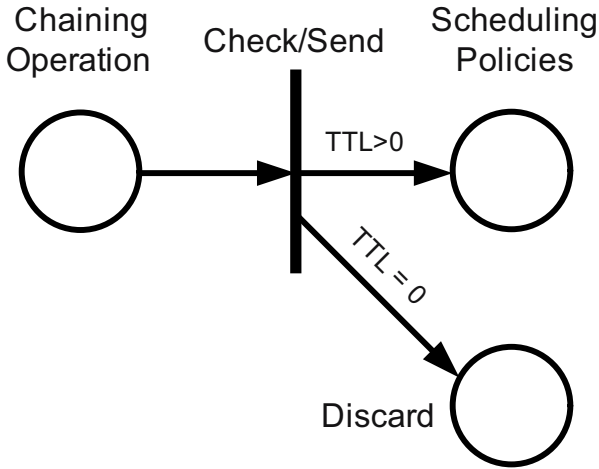


Fig. 12. The basic chaining mechanism

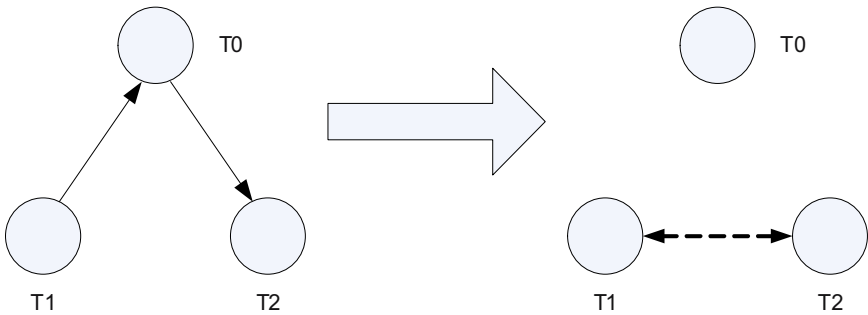


Fig. 13. Formation of the simulated synapse

be strengthened and weakened. Such a link is called a ‘simulated synapse’. Figure 13 shows the formation of the simulated synapse.

The underlying algorithm used to strengthen and weaken the link can be defined in the chaining policies. One of the simplest algorithms is the aging algorithm, in which every simulated synapse has an associated weight. A weight is a numerical value between 0 and 1, which is used to evaluate the strength of its associated chain (1 representing the strongest link, and 0 representing no link). The weight is calculated based on the frequency of communication occurring on its associated chain. When a simulated synapse is created, an initial weight is specified. Then for each interval I , the weight is squared. If the resulting weight is less than the threshold θ , the simulated synapse is removed. On the other hand, each time the *Link* transition is fired, the square root of the weight is calculated. Below is a pseudo implementation of the aging algorithm which utilizes a monitor.

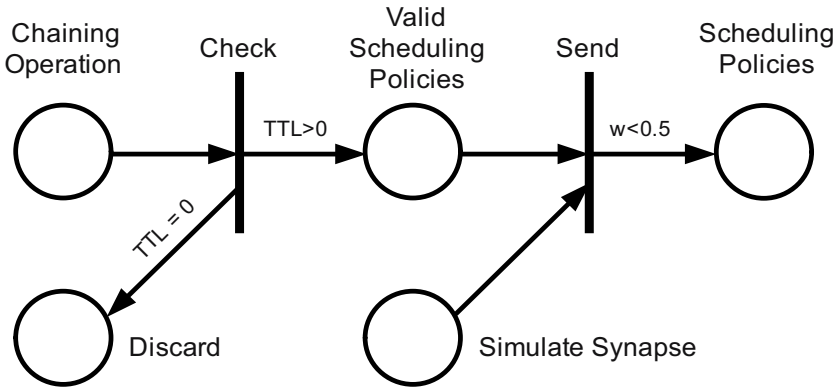


Fig. 14. Advanced chaining mechanism example

To take advantage of the simulated synapse, the chaining process must take the strength of the simulated synapse into consideration. Figure 14 demonstrates an example of the advanced chaining mechanism.

Algorithm 1 Aging algorithm using a Monitor

```

/* Global Area */
DEFINE MONITOR M /* monitor */
DEFINE CONSTANT  $\theta$  /* threshold */
DEFINE CONSTANT I /* interval */
DEFINE OBJECT synapse /* simulated synapse */

/* Link transition thread */
synchronised(M) {
if synapse.weight = 0 then
  INITIALISE synapse.weight
else
  synapse.weight = SQRT(synapse.weight)
  NOTIFY();
}

/* Background daemon thread */
synchronised(M)
{
while synapse.weight >  $\theta$  do
  WAIT(I)
  synapse.weight = synapse.weight*synapse.weight
  synapse.weight = 0
}

```

Request-related Scheduling

As the TSs are allowed to register services in `smartGRID`, one of the functions of scheduling is to balance requests and schedule the corresponding TSs to the computing nodes to serve these requests. In fact, the only difference between task-related scheduling and request-related scheduling is that objects are *actually* scheduled. In the former, the object is the TS or the scheduling policies extracted from the TS; in the latter, the object is the service request. As the requests have no common characteristic in terms of the potential load they may bring in, it is hard for the scheduling components to make rational decisions. However, `smartGRID` still provides two ways to help services achieve high throughput.

Recalling the TSD, there is a subsection called *instance policies*, which defines the Minimum number of Active Instances (MINAI), the Maximum number of Active Instances (MAXAI), the Minimum number of Standby Instances (MINSI), and the Maximum number of Standby Instances (MAXSI). When a service TS (a TS that defines services) is scheduled, the instance policies are used to guide the scheduling components to retain a proper number of service instances. Then, when a client attempts to invoke these services, it uses the Web Services standards to discover the service instances. It is at that time that client requests are distributed to the pre-allocated service instances, so that these requests are balanced.

Another way to balance service requests is to let the service providers themselves manage the requests, as only they know about the internals of the requests and the best way to handle them. The multi-agent architecture of `smartGRID` allows the service TSs to use the underlying APIs to provide their own scheduling strategies, and schedule the requests themselves.

5 A Peer-to-Peer Solution to Grid Computing in Open Environments

In this Section, we propose our second hybrid solution by focusing solely on a P2P architecture; we call this pure P2P solution `smartGRID2`.

5.1 Overall Architecture and Core Components of `smartGRID2`

`smartGRID2` consists of three major components: *an improved task model*, which derives from the task model of `smartGRID`; *a P2P computing architecture*, which develops the chaining mechanism and simulated synapse into a message passing and routing framework; and *a resource management framework*, which uses profiles to match computing resources with requests, and provides up-to-date information about matched resources. In this Section, we present an overview of these components, and discuss each component in turn.

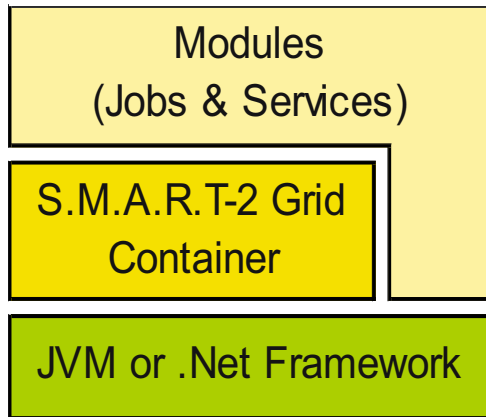


Fig. 15. Components within a `smartGRID2` computing node

There are two tiers in `smartGRID2`: *clients* and *computing nodes* (or peers). A *microkernel Grid container* runs on every *computing node*. These containers serve as the runtime and managerial environment for the tasks. A task (such as a job or service) is described as a group of linked *modules* in `smartGRID2` – a *module* being the fundamental unit that can be scheduled among peers. All *modules* run on peers, or more specifically, within the `smartGRID2` containers. Figure 15 shows the relationship between the *modules* and the container.

The `smartGRID2` container allows modules to register to the service portal as Web Services. The service portal conforms to Web Services standards [57], and allows clients to interact with the Grid using SOAP messages. Figure 16 demonstrates the overall architecture of the `smartGRID2` container.

Inside the container, there are four components: the *Runtime Environment (RT)*, the *Management Agent (MA)*, the *Profiling Agent (PA)*, and the *Computing Agent (CA)*. The *Runtime Environment* provides fundamental routines and runtime libraries for both agents and *modules*. For example, XML parsing libraries, and implementations of Web Services standards [57], such as Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) [9], are included in the *Runtime Environment*; the service portal is also part of the *Runtime Environment*. The *Management Agent* provides the managerial interface between the container and the Grid Management Service. It manages the container, the policies and the configurations as well. The *Profiling Agent* gathers the status of the network, the peers and the running *modules*, and provides optimized dynamic configurations for the *Computing Agent*. The *Computing Agent* is responsible for managing the *module* life cycles, locating resources and *modules*, discovering services, and scheduling *modules* and service invocations among peers, while providing fault tolerance and load balancing. Figure 17 shows agent interactions within the Grid container.

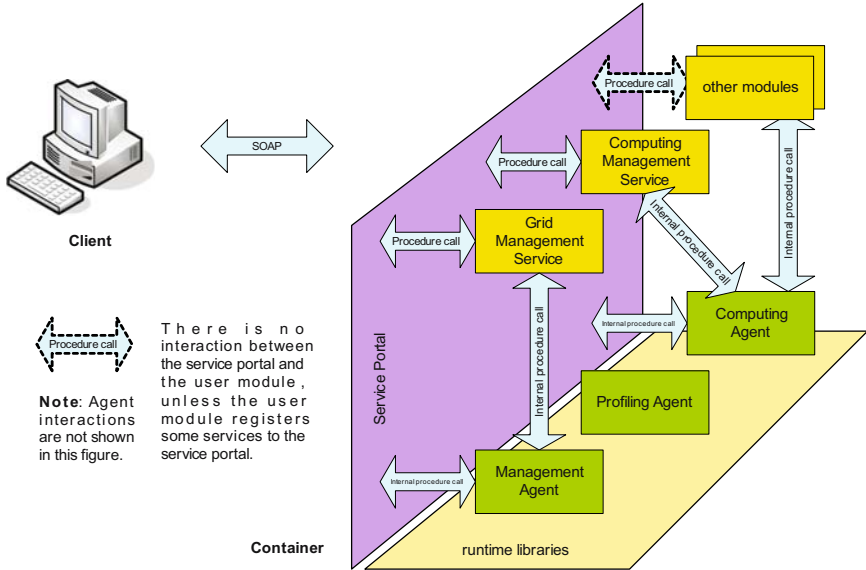


Fig. 16. Schematic view of smartGRID2 container

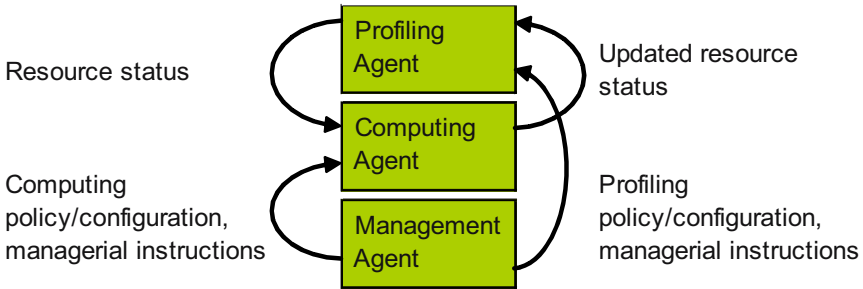


Fig. 17. Agent interactions in a smartGRID2 container

Besides these components, there are two predefined *modules* which register as *Grid Management Service (GMS)* and *Computing Management Service (CMS)*, respectively. GMS allows users who have certain privileges to manage the Grid – for instance, specifying the computing policy/configuration, and monitoring the Grid status. CMS provides interfaces for clients to manage the computing resources. In smartGRID2, all objects involved in the computing process are regarded as resources. These resources include *module* executables, service descriptions registered by the *modules*, data files, storage, computational cycles, and similar.

5.2 Module – An Improved Task Model

As mentioned in the preceding Section, *smartGRID2* uses *modules* to describe tasks. A module consists of the module description, executables, serialization and any module-owned files. Figure 18 displays the *module* composition.

The Module Description (MD) has two sections: ‘task’ and ‘service’. Figure 19 shows the MD task section which defines the task-related information. It consists of two subsections, these being:

- The *deployment description* subsection defines information about a *module’s* executables (for example, what the entry point is if it is a startup *module*), as well as any *module* dependencies. A *module’s* dependency is another *module* or a service that the *module* depends on.
- The *computing policy* subsection defines a *module’s* (a) minimum hardware requirements on a peer’s machine type, processor type, and contributed cycle/memory/storage; (b) the estimated amount of computation; (c) the expected completion time; (d) the priority level; and (e) relay policies (see Section 5.3).

The MD service section is optional and is only needed if the *module* registers one or more services to the Grid; it uses WSDL to define the service interfaces.

The executables are Java byte code files or .NET executables. When a running *module* is suspended by a user, or if it is relocated (see Sect. 5.4),

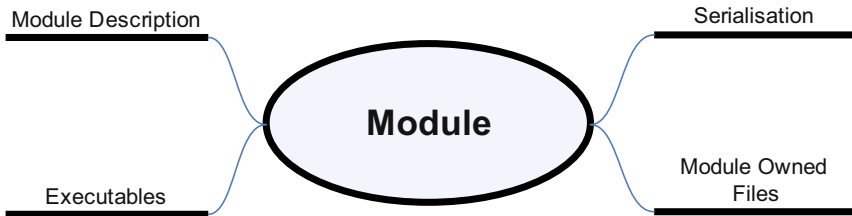


Fig. 18. *smartGRID2* module

```

{Deployment description
 {Executable description
  Module dependencies
  Service dependencies}
Computing policies
 {Minimal hardware requirements
  Estimated computation amount
  Expected completion time
  Priority level
  Relay policies }}
  
```

Fig. 19. Task section of the module description

it will be serialized. This process is equivalent to object serialization in Java [52]. It allows the Grid container to store the *module* runtime dynamics, and restore them when execution of the *module* is resumed. The *module*-owned files (MOFs) are files that tightly bind to the *module*. These files are regarded as part of the *module*, and migrate, together with the *module*'s description, executables and serialization.

A group of linked *modules* consists of a complete task. Each *module* implements a fraction of the overall task. As these *modules* can be executed at the same time on different peers, load balance and parallelism are achieved. Each task has a startup *module*. After all the task *modules* have been deployed to the Grid, the client can start the task through CMS. CMS then uses the `create` method of the `IModuleContext` interface to create an instance of the startup *module*. Once the startup module is instantiated and runs, it can start instances of other modules by using the same interface. Figure 20 depicts the hierarchy of *module* instances in `smartGRID2`.

When a *module* is instantiated, it gains access to the `IModuleContext` interface, which is provided by the *Computing Agent*. This interface defines three kinds of methods, which respectively allow a *module*'s instance (a) to create instances of other *modules*, (b) to perform procedure calls (in other words, invoke methods of other *modules*), and (c) to delete instances which are not in use in order to release their occupied resources. Figure 21 lists the `IModuleContext` interface. The internals of the creation process, the subsequent procedure calls, and the deletion process are all discussed in Sect. 5.4.

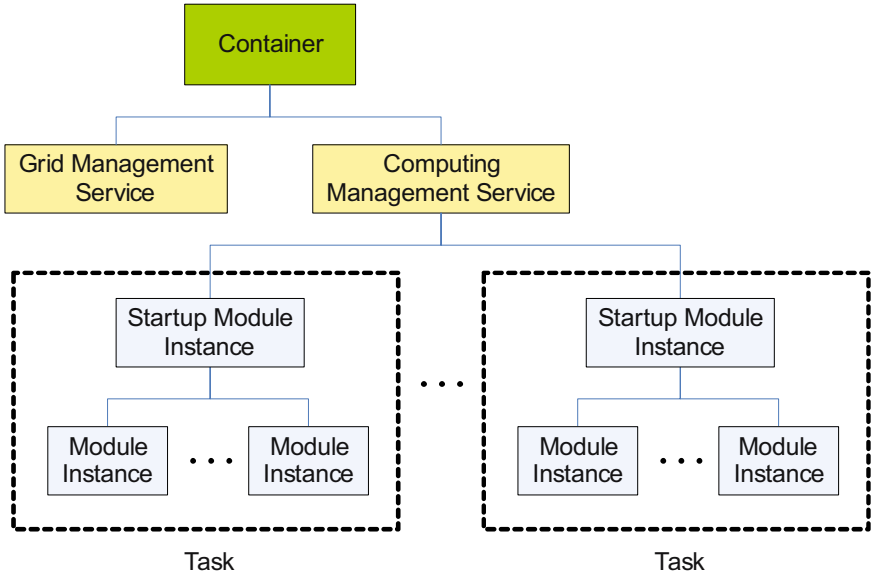


Fig. 20. Hierarchy of module instances in `smartGRID2`

```

public interface IModuleContext
{
    public ModuleInstance create(String moduleName,
                                Object ... args)
                                throws ModuleException;

    public Object invoke(ModuleInstance moduleInstance,
                          String method,
                          Object ... args)
                          throws ModuleException;

    public void delete(ModuleInstance moduleInstance)
                     throws ModuleException;

    /**
     * For static method only
     */
    public Object invoke(String moduleName,
                          String method,
                          Object ... args)
                          throws ModuleException;
}

```

Fig. 21. IModuleContext interface

5.3 Peer-to-Peer Computing Architecture

A number of interconnected peers comprise **smartGRID2**. The notion of ‘connection’ in **smartGRID2** is defined as follows:

Definition 4 A ‘connection’ represents a message passing route from one peer to another, and is not equivalent to a network connection. A connection from peer *A* to peer *B* means peer *A* has the information to send messages to peer *B* successfully, where *A* is the source of the connection, and *B* is the destination of the connection.

Definition 5 A connection is ‘directional’ – that is, ‘peer *A* connects to peer *B*’ does not presume ‘peer *B* connects to peer *A*’. ‘Peer *A* connects to peer *B*’ is represented as $A \rightarrow B$. If peer *B* also connects to peer *A*, then *A* and *B* have a two-way connection, which is represented as $A \leftrightarrow B$.

Definition 6 A peer’s connections are the connections whose source is the ‘peer’. When recording these connections, only the destination peers (destinations for short) are recorded.

The peers which have a relatively large number of connections are called *hubs*. When the Grid is constructed, a number of *computing nodes* which have high availability, good connectivity and good performance are selected as the ‘backbone’ of the Grid. Each of them permanently has at least two two-way connections with the others. As new *nodes* appear, they register to at least one of the backbone *nodes*, so that a two-way connection can be established between them.

In *smartGRID2*, the connections of a peer are recorded in a hash table, where the destinations of the connections are the keys, and the objects representing the strength of the connections (called ‘simulated synapses’) are the values. Figure 22 shows a typical implementation of a simulated synapse (synapse for short). Definitions of the above fields are described as follows:

```

public class Synapse {
    public double strength;
    public double deathThreshold;
    public double activateThreshold;
    public double permThreshold;

    public static Synapse createPermSynapse () {
        Synapse synapse = new Synapse ();
        synapse.strength = 1;
        return synapse;
    }
    public static Synapse createTempSynapse () {
        Synapse synapse = new Synapse ();
        synapse.deathThreshold =
            SynapseManager.deathThreshold +
            deathRange * random.nextDouble ();
        synapse.permThreshold =
            SynapseManager.permThreshold +
            permRange * random.nextDouble ();
        synapse.activateThreshold =
            synapse.permThreshold -
            (synapse.permThreshold -
            synapse.deathThreshold) * GOLDEN_SECTION;
        synapse.strength =
            Math.pow(synapse.activateThreshold, 2);
        return synapse;
    }
}

```

Fig. 22. Sample implementation of a simulated synapse

Definition 7 *strength*, whose range is $(0, 1]$, represents the current strength of the connection. A value '1' means that the connection is a permanent connection. A random initial value which is less than *activeThreshold* is given to *strength* when a connection is created.

Definition 8 *deathThreshold*, whose value is randomly selected from a user-configured range when a connection is created. When *strength* is less than *deathThreshold*, the connection is removed from the hash table, which means the connection breaks up.

Definition 9 *activateThreshold* – when a connection is created, a random value is selected from a user configured range as *lstinline activateThreshold*. At that stage, the connection is inactive. Afterwards, if *strength* grows to a value greater than *activateThreshold*, the connection becomes active, and the *activateThreshold* is set to 0.

Definition 10 *permThreshold*. If an active connection's *strength* continues growing to a value greater than *permThreshold*, then *strength* is set to 1 and the connection becomes a permanent one.

Two operations can be applied to a synapse: *grow*, which increases the strength of the connection; and *decay*, which decreases the connection strength. Figure 23 shows the internals of these operations.

There are three kinds of computing operations in *smartGRID2*, namely for deploying, locating, and utilizing resources. In order to achieve load balance, and allocate the most suitable peer to perform a computing operation (or series of operations), or to locate certain resources, various messages are generated by the peer which receives the client's instruction, and then delivered to other peers before performing this operation(s). These messages and the reply messages are encapsulated into impulses, and transmitted among the peers. This process is called *relay*. The definition of 'Impulse' is as follows:

```
public class Impulse {
    public int type_ttl;
    public long serial;
    public Peer from;
    public Message message;
}
```

Assume that *O* represents the peer which generates the message, and *R* represents any peer which replies to *O*. An impulse transmitted from *O* to *R* is called an *outbound impulse*; an impulse transmitted from *R* to *O* is called an *inbound impulse*. For any outbound message, the value of *type_ttl* indicates the Time-To-Live (TTL) of the impulse, and is set by *O* when *O* creates the

```

private static Hashtable<Peer, Synapse> synapses;

public static void grow(Peer peer) {
    Synapse synapse = synapses.get(peer);
    if(synapse == null)
        synapses.put(peer, Synapse.createTempSynapse());
    else {
        if(synapse.strength == 1)
            return;
        if(synapse.activateThreshold == 0) {
            synapse.strength =
                Math.pow(synapse.strength, 0.5);
            if(synapse.strength > synapse.permThreshold)
                synapse.strength = 1;
        }
        else {
            synapse.strength +=
                Math.pow(synapse.activateThreshold, 2);
            if(synapse.strength >
                synapse.activateThreshold) {
                synapse.strength =
                    synapse.deathThreshold +
                    (synapse.permThreshold -
                     synapse.deathThreshold) *
                    GOLDEN_SECTION;
                synapse.activateThreshold = 0;
            }
        }
    }
}

public static void decay(Peer peer) {
    Synapse synapse = synapses.get(peer);
    if(synapse.strength == 1)
        return;
    if(synapse.activateThreshold == 0)
        synapse.strength=Math.pow(synapse.strength, 2);
    else
        synapse.strength -=
            Math.pow(synapse.activateThreshold, 2);
    if(synapse.strength < synapse.deathThreshold)
        synapses.remove(peer);
}

```

Fig. 23. Operations on the simulated synapse

impulse; the `serial` field contains a unique number generated by *O*; the `from` field is set to *O*; and the `message` field contains the actual message carried by the impulse. When *R* replies to *O*, it resets `type_ttl` to -1 to indicate that the impulse carries a replied message; `serial` is not changed; `from` is reset to *R*; and `message` is set to the replied message.

When a peer starts, a fixed-size queue, which is used to cache the impulses relayed by the peer, is created. `Hashtable<Long,List<Impulse>>` impulses is also created to store the inbound impulses, where the key (whose type is `Long`) denotes the sequence number of the impulse, and the value (which is a list of `Impulse`) denotes the inbound impulses. When a relay process starts, an outbound impulse is created by *O* with its fields being set, and an empty list created and inserted into the hash table. Next, *O* transmits the impulse to all of its active connections. When any peer receives the impulse, it checks whether the impulse is already in its queue. If it is, it discards the impulse; otherwise it decreases the TTL by one, and then checks whether this is 0. If it is, the impulse is discarded; otherwise the peer appends the impulse to the end of the queue, and relays it to all its active connections. Finally, it checks whether it is able to respond to the message carried by the impulse. If it can, an outbound impulse will be generated and transmitted directly to *O*. Figure 24 demonstrates the *relay* process.

After *O* transmits the impulse, it suspends the calling thread for a period of time specified before the transmission or until the number of replies reaches a threshold. Whenever a reply comes back from *R* to *O*, and there exists a corresponding list in the hash table, it is added to the list, and the *grow* operation is performed on the connection to *R*. When the thread is resumed, the replies are retrieved from the corresponding list in the hash table. Then *O* goes through all its connections, and performs the *decay* operation on the connections without a reply. Afterwards, all replies are returned to the thread for selection. Figure 25 shows the `getReplies` method, which is implemented in CA.

With the *selection* process (Sect. 5.4), the *relay* process enables load balancing and the election of the most suitable peer for a certain payload (that is, the message). In the long run, connections between peers are optimized according to the payload characteristics. New hubs are also developed, so that the Grid will gain better connectivity and a higher ratio of resource utilization, as well as work more efficiently.

5.4 Resource Management and Scheduling Mechanisms

`smartGRID2` uses resource matrices to track the status of computing resources. Table 2 displays a sample matrix. It defines the type of resource, where the resource resides, and the resource's status (called 'profile') or the resource description. A peer's local resources are registered by the profiling agent

```

impulse = CA.receiveImpulse ();

if(impulse.isReply()) {
    List<Impulse> list =
        impulses.get(impulse.getSerial());
    if(list != null) {
        list.add(impulse);
        grow(impulse.getFrom());
    }
}
else if(queue.indexOf(impulse) == -1) {
    Handler handler =
        Container.getHandler(impulse.getMessage());
    if(--type_ttl > 0) {
        appends impulse to the queue
        CA.relay(impulse);
    }
    if(handler != null) {
        Message reply =
            handler.handle(impulse.getMessage());
        impulse.type_ttl = -1;
        Peer dest = impulse.from;
        impulse.from = Container.getLocal();
        impulse.message = reply;
        CA.send(impulse, dest);
    }
}
/**
 * else
 *     Discard impulse
 */

impulse.destroy();

```

Fig. 24. Relay process

when the peer starts. The profiling agent also updates the local resource profiles when they change. Figure 26 shows a representative processor profile definition.

When a module requires a resource, its container *C* first tries to match the required resource with those in the resource matrix. If none of them matches the requirement, the container starts a *relay* process. Alternatively, the container may start the *relay* process immediately upon receiving the module's request. How the container behaves is determined by the resource type. For example, local service resources have precedence over remote service resources, but there is no such discrimination in terms of processor resources.

```

public static List<Impulse> getReplies(Impulse impulse)
{
    List<Impulse> list =
        impulses.remove(impulse.getSerial());
    ArrayList<Peer> peers =
        new ArrayList<Peer>(synapses.size());
    peers.addAll(synapses.keySet());
    for(Impulse i : list)
        peers.remove(i.getFrom());
    for(Peer peer : peers)
        decay(peer);
    return list;
}

```

Fig. 25. getReplies method

Table 2. Resource matrix of peer 192.168.2.1

Resource Type	Residing Peer	Resource Profile/Description	References
Processor	192.168.2.1	Pentium-4; 1.8GB; fully contributed; no running module	n/a
Storage	192.168.2.1	1024MB free space; transfer speed 160(120)Mbps (R/W)	n/a
Module Instance	192.168.2.1	name = decrypt; ID = 1234	192.168.2.7 192.168.2.8
File	192.168.2.2	/modules/decrypt.mar	192.168.2.1
Service	192.168.2.4	/unix-encrypt; Module Description with Service Section	192.168.2.1

```

{Capability
  {Machine type
   Processor type
   Contributed cycles amount
   Contributed memory amount
   Contributed storage amount}
  Load
  {Module#1
   Module#2
   ...}}

```

Fig. 26. Sample processor profile definition

During the *relay* process, the participating peers look up the required resource in their resource matrices. If matching resources exist, references to these resources are returned to *C*. If multiple replies exist, *C* starts a resource selection process to determine which resource is most suitable. The outcome is then returned to the module for its subsequent operations. Moreover, if the type of resource located has local precedence, it will be cached in *C*'s resource matrix. A resource matrix only caches a limited number of references. Each time a cached reference is retrieved, it is regarded as 'updated'. The least updated entry will be removed if the cache is full and a new reference comes in.

Once the reference to a resource is obtained, it is accessible to the module through `smartGRID2`. Each time a resource is accessed, its reference is quoted and passed to the resource's residing peer *R*. *R* will then perform the actual operations and send the results back to the module. When the module finishes using the resource, it notifies *R* so that the resource can be released.

A peer also keeps records of other peers which have cached references to its local file, service, and module instance resources, so that references can be updated when the actual resources migrate to other peers.

In `smartGRID2`, files can be uploaded to the backbone nodes through CMS. Unlike other resources, local files never appear in the resource matrix. When a file is located and used, it can be cached by the peer that uses the file, if there is sufficient storage therein.

Module executables are regarded as files, and need to be uploaded to the Grid before execution. `smartGRID2` has a two-stage scheduling mechanism. Once a module is uploaded, its residing peer *O* will trigger a *relay* process, informing other peers of the potential workload. Other peers will reply to *O* if they can execute the module. *O* then determines the suitability of these peers (including *O* itself). The module will be moved to the winner if the winner is a backbone node; otherwise it is transferred to the winner and cached there. During the second stage, when the module is about to be created, a *relay* process will be started to locate the module. Once it is located, it will be scheduled and executed by its residing peer. The following illustrates a reference to the instance of the module used in the procedure calls:

```
public class Resource
{
    private Peer peer;
}
public final class ModuleInstance extends Resource
implements Serializable
{
    public String name;
    public String id;
}
```

The only difference in the execution process of a service is that it has to be discovered *before* its module execution process. Figure 27 demonstrates this process.

5.5 Compatibility and Inter-Operability

Recalling the task model (Sect. 4.2), it is easy to see that the new model enables modeling of both conventional (stateless) services and stateful tasks. A *module* is allowed to register its own services to the service portal using Web Services standards. Hence, any WS-compatible client is capable of accessing these services through `smartGRID2`.

There are two means by which stateful information for a service in `smartGRID2` can be maintained. The client and the service can use agreed methods, such as WS-Resource, to exchange stateful information. `smartGRID2` supports WS-Resource standards, hence a WS-Resource based client needs no modification to work with `smartGRID2`, as long as the service interface is not changed. Another way to preserve the states throughout different service invocations is to create a transaction-specific service module. In this case, a *token* representing a certain transaction is passed during service invocations. When a new transaction starts, the startup *module* of the service creates a new service *module* to serve the transaction. Stateful information is maintained by the service *modules*. The *tokens* act as identifiers for the startup *module* to dispatch service invocations to an appropriate service *module*. Once the transaction is completed, the client implicitly notifies the service's startup *module*, so that the startup *module* can delete the corresponding service *module* and release resources.

Since `smartGRID2` conforms to Web Services and WS-Resource standards, any *module* in `smartGRID2` is able to operate on the services provided by other WS-compatible Grids using these standards. However, being different in its architecture and programming model, `smartGRID2` has neither binary nor source code compatibility with programs running on existing Grids.

6 Conclusion and Further Work

The primary objective of this Chapter was to solve fundamental issues relating to the architecture of Grid computing in open environments.

We investigated conventional Grid computing architectures and introduced two architectures, `smartGRID`—a client-server architecture—and `smartGRID2`—a peer-to-peer (P2P) architecture—which combined intelligent agent and colored Petri Net technologies in order to handle the problems of job balance, distributed resource management and dynamic task scheduling. As a result, we are able to provide approaches for solving some fundamental

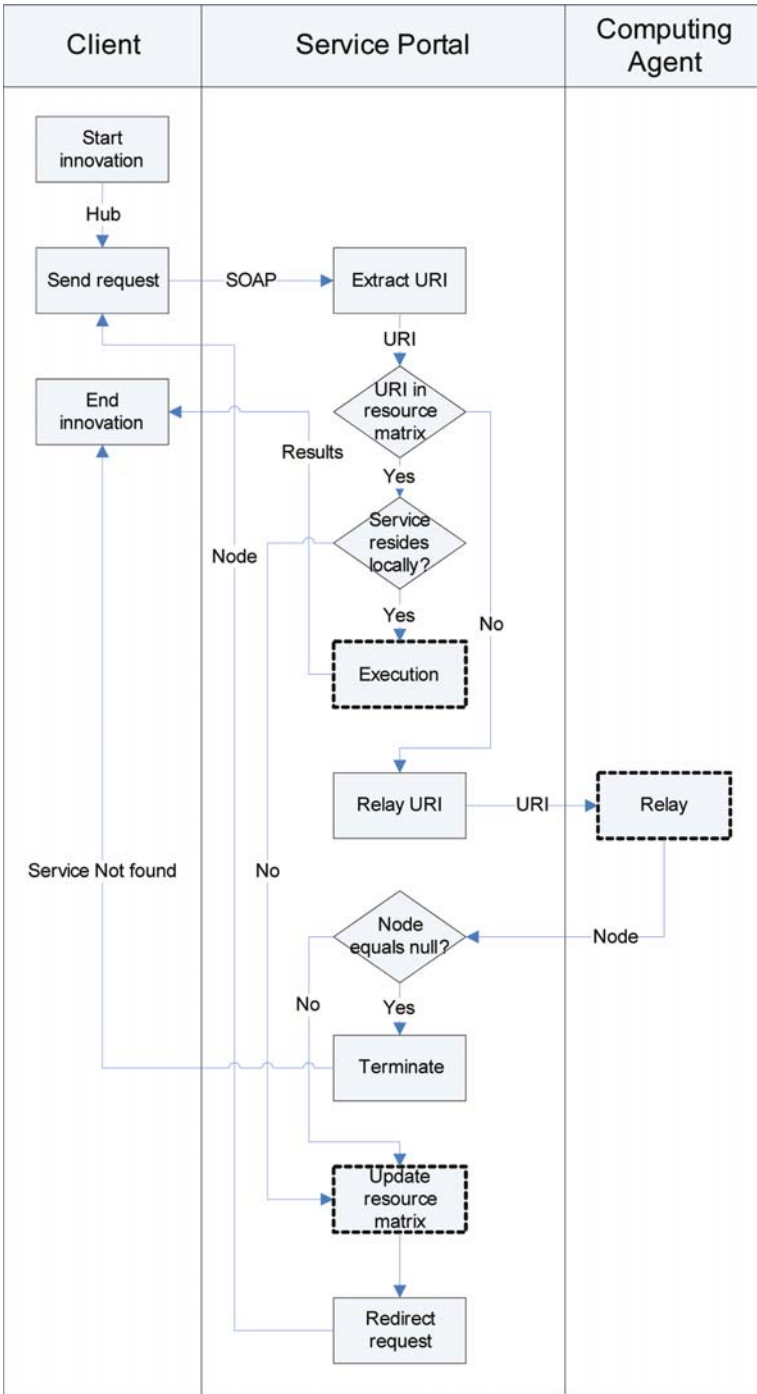


Fig. 27. Service invocation process

problems inherent in Grids, and moreover also support Grid applications in open environments.

Our proposed architectures have various advantages over conventional Grids, more specifically:

- Both architectures provide direct management of *computing nodes*, with load balance being guaranteed by various mechanisms,
- Adaptability of services can be easily achieved by spawning subtasks to serve increased requests,
- Both architectures are resilient to faults by keeping redundant copies of tasks and their intermediate results, and
- The P2P architecture transparently supports inter-task communication through its in-built interface.

More research is needed into the organization and management of physically distributed computing resources, how to solve issues of authentication and authorization, as well as encryption of data and communications among *nodes* in general-purpose Grid systems.

Acknowledgement

The authors would like to acknowledge the financial support of the Intelligent Systems Research Centre at the University of Wollongong.

References

1. Akarsu E, Fox GC, Furmanski W, Haupt T (1998) Webflow: High-level programming environment and visual authoring toolkit for high performance distributed computing. In: *Proc. 1998 ACM/IEEE Conf. Supercomputing*, San Jose, CA. IEEE Computer Society Press, Los Alamitos, CA: 1–7.
2. Alfred WL (2002) The future of peer-to-peer computing. *Communications ACM*, 46: 56–61.
3. Baker M, Buyya R, Laforenza D (2002) Grids and grid technologies for wide-area distributed computing. *Software: Practice and Experience*, 32: 1437–1466.
4. Baratloo A, Karaul M, Kedem Z, Wyckoff P (1996) Charlotte: Metacomputing on the web. In: *Proc. 9th Conf. Parallel and Distributed Computing Systems (PDCS-96)*, September, Dijon, France: 181–188.
5. Buyya R (2002) Grid Computing Info Centre: Frequently Asked Questions (FAQ) (available online at <http://www.gridcomputing.com/gridfaq.html> – last accessed May 2007).
6. Buyya R, Abramson D, Giddy J (2000) Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In: *Proc. 4th Intl. Conf. High Performance Computing in Asia-Pacific Region (HPC ASIA '2000)*, 14–17 May, Beijing, China. IEEE Computer Society Press, Los Alamitos, CA. 1: 283–289.

7. Cappello P, Mourloukos D (2001) A scalable, robust network for parallel computing. In: *Proc. 2001 Joint ACM-ISCOPE Conf. Java*, 2–4 June, Stanford, CA. ACM Press, New York, NY: 78–86.
8. Chetty M, Buyya R (2002) Weaving computational grids: How analogous are they with electrical grids. *Computing in Science and Engineering*, 4: 61–71.
9. Christensen E, Curbera F, Meredith G, Weerawarana S (2001) Web Services Description Language (WSDL) 1.1. (available online at <http://www.w3.org/TR/wsdl> – last accessed May 2007).
10. Christiansen BO, Cappello P, Ionescu MF, Neary MO, Schauser KE, Wu D (1997) Javelin: Internet-based parallel computing using Java. *Concurrency: Practice and Experience*, 9: 1139–1160.
11. Cohen B (2003) Incentives build robustness in BitTorrent. (available online at <http://www.bittorrent.com/bittorrentecon.pdf> – last accessed May 2007).
12. Cost RS, Chen Y, Finin T, Labrov Y, Peng Y (1999) Modeling agent conversations with coloured petri nets. In: *Proc. Workshop on Specifying and Implementing Conversation Policies*, May, Seattle, WA: 59–66.
13. Cranefield S, Purvis M, Nowostawski M, Hwang P (2002) Ontology for interaction protocols. In: *Proc. 2nd Intl. Workshop Ontologies in Agent Systems (OAS'02 at AAMAS'02)*, 15–19 July, Bologna, Italy. ACM Press, New York, NY: 15–19.
14. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S, (1998) A resource management architecture for metacomputing systems. In: Feitelson DG, Rudolph L (eds.) *Proc. IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, 30 March, Orlando, FL. Lecture Notes in Computer Science 1459, Springer-Verlag, Berlin: 62–82.
15. Epema DHJ, Livny M, van Dantzig R, Evers X, Pruyne J (1996) A worldwide flock of condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12: 53–65.
16. Fitzgerald S, Foster I, Kesselman C, Laszewski GV, Smith W, Tuecke S (1997) A directory service for configuring high-performance distributed computations. In: *Proc. 6th IEEE Symp. High Performance Distributed Computing*, 5–8 August, Portland, OR. IEEE Computer Society Press, Los Alamitos, CA: 365–375.
17. Foster I (2002) What is the grid? a three point checklist. *Grid Today*, 1 (available online at <http://www.gridtoday.com/02/0722/100136.html> – last accessed May 2007).
18. Foster I (2005) A Globus Toolkit Primer. (available online at http://www-unix.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf – last accessed May 2007).
19. Foster I, Czajkowski K, Ferguson D, Frey J, Graham S, Maguire T, Snelling D, Tuecke S (2005) Modeling and managing state in distributed systems: the role of OGSi and WSRF. *Proc. IEEE*, 93: 604–612.
20. Foster I, Iamnitchi A (2003) On death, taxes, and the convergence of peer-to-peer. In: *Proc. 2nd Intl. Workshop Peer-to-Peer Systems (IPTPS 2003)*, 20–21 February, Berkeley, CA. Lecture Notes in Computer Science 2735, Springer-Verlag, Berlin: 118–128.
21. Foster I, Kesselman C (1997) Globus: A metacomputing infrastructure toolkit. *Intl. J. Supercomputer Applications and High Performance Computing*, 11: 115–128.
22. Foster I, Kesselman C (1999) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffman, San Francisco, CA.

23. Foster I, Kesselman C, Nick JM, Tuecke S (2002) The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. (available online at <http://www.globus.org/research/papers/ogsa.pdf> – last accessed May 2007).
24. Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid: Enabling scalable virtual organizations. *Intl. J. High Performance Computing Applications*, 15: 200–222.
25. Fox G, Haupt T, Akarsu E, Kalinichenko A, Kim KS, Sheethalnath P, Youn CH (1999) The gateway system: Uniform web based access to remote resources. In: *Proc. 1999 ACM Conf. Java*, June, San Francisco, CA. ACM Press, New York, NY: 1–7.
26. Frey J, Tannenbaum T, Foster I, Livny M, Tuecke S (2001) Condor-g: A computation management agent for multi-institutional grids. In: *Proc. 10th IEEE Symp. High Performance Distributed Computing (HPDC10)*, 7–9 August, San Francisco, CA. IEEE Computer Society Press, Los Alamitos, CA: 55–63.
27. Geist A, Beguelin A, Dongarra JJ, Jiang W, Manchek R, Sunderam V (1994) *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA.
28. Geist GA, Kohl JA, Papadopoulos PM (1996) PVM and MPI: a comparison of features. *Calculateurs Paralleles*, 8: 137–150.
29. Global Grid Forum Open grid services infrastructure (OGSI) version 1.0 (2003) available online at http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf – last accessed May 2007.
30. Globus Alliance Globus Toolkit 4.0 (GT4) (2005) available online at <http://www-unix.globus.org/toolkit/docs/4.0/GT4Facts/> – last accessed May 2007.
31. Goldman J, Rawles P, Mariga J (1999) *Client/Server Information Systems*. Wiley, Hoboken, NJ.
32. Grimshaw A, Ferrari A, Lindahl G, Holcomb K (1998) Metasystems. *Communications ACM*, 41: 46–55.
33. Grimshaw AS, Wulf WA (1996) Legion: Flexible support for wide-area computing. In: *Proc. 7th ACM SIGOPS European Workshop*, 9–11 September, Connemara, Ireland. ACM Press, New York, NY: 205–212.
34. Grimshaw AS, Wulf WA (1997) Corporate: The Legion vision of a worldwide virtual computer. *Communications ACM*, 40: 39–45.
35. Gropp W, Lusk E, Skjellum A (1994) *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, Cambridge, MA.
36. Jensen K (1992) *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Springer-Verlag, Berlin.
37. Koepela E (2001) Seti@home: Massively distributed computing for SETI. *Computing in Science and Engineering*, 3: 78–83.
38. Laszewski GV, Gawor J, Pena CJ, Foster I (2002) Infogram: A grid service that supports both information queries and job execution. In: *Proc. 11th IEEE Intl. Symp. High Performance Distributed Computing (HPDC'02)*, July, Edinburgh, Scotland. IEEE Computer Society Press, Los Alamitos, CA: 333–342.
39. Ledlie J, Shneidman J, Seltzer M, Huth J (2003) Scooped, again. In: *Proc. 2nd Intl. Workshop Peer-to-Peer Systems (IPTPS 2003)*, February, Berkeley, CA. Lecture Notes in Computer Science 2735, Springer-Verlag, Berlin: 129–138.
40. Lesser V (1999) Cooperative multiagent systems: A personal view of the state of the art. *IEEE Trans. Knowledge and Data Engineering*, 11: 133–142.

41. Marcus E, Stern H (2000) *Blueprints for High Availability: Designing Resilient Distributed Systems*. Wiley, New York, NY.
42. Neary MO, Christiansen BO, Cappello P (1999) Javelin: Parallel computing on the internet. *Future Generation Computer Systems*, 15: 659–674.
43. Nowostawski M, Purvis M, Cranefield S (2001) A layered approach for modeling agent conversations. In: *Proc. 2nd Intl. Workshop Infrastructure for Agents, MAS, and Scalable MAS*, 28 May, Montreal, Canada: 163–170.
44. Pacheco PS (1997) *Parallel Programming with MOI*. Morgan Kaufman, San Francisco, CA.
45. Peterson J (1981) *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, NJ.
46. Pfister G (1997) *In Search of Clusters (2nd ed)*. Prentice Hall, Englewood Cliffs, NJ.
47. Poutakidis D, Padgham L, Winikoff M (2002) Debugging multi-agent system using design artefacts: The case of interaction protocols. In: *Proc. 1st Intl. Joint Conf. Autonomous Agents and Multi Agent Systems*, 15–19 July, Bologna, Italy: 960–967.
48. Roehrig M, Ziegler W, Wieder P (2002) Grid Scheduling Dictionary of Terms and Keywords. *Global Grid Forum*, (available online at <http://www.ggf.org/documents/GWD-I-E/GFD-I.011.pdf> – last accessed May 2007).
49. Schopf J (2001) The actions when superscheduling (available online at <http://www.ggf.org/documents/GFD/GFD-I.4.pdf> – last accessed May 2007).
50. Smarr L, Catlett CE (1992) Metacomputing. *Communications ACM*, 35: 44–52.
51. Snir M, Otto S, Huss-Lederman S, Walker D, Dongarra J (1996) *MPI: The Complete Reference*. MIT Press, Cambridge, MA.
52. Sun Microsystems Inc. (2004) Java Object Serialization Specification. (available online at <http://java.sun.com/j2se/1.5/pdf/serial-1.5.0.pdf> – last accessed May 2007).
53. Waldman M, Rubin AD, Cranor LF (2000) Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In: *Proc. 9th USENIX Security Symp*, 14–17 August, Denver, CO: 59–72.
54. Welch V, Siebenlist F, Foster I, Bresnahan J, Czajkowski K, Gawor J, Kesselman C, Meder S, Pearlman L, Tuecke S (2003) Security for grid services. In: *Proc. 12th Intl. Symp. Performance Distributed Computing (HPDC-12)*, June, Seattle, WA. IEEE Computer Society Press, Los Alamitos, CA: 48–57.
55. W3C (2003) HTTP - Hypertext Transfer Protocol. (available online at <http://www.w3.org/Protocols/> – last accessed May 2007).
56. W3C (2003) Simple Object Access Protocol. (available online at <http://www.w3.org/TR/soap/> – last accessed May 2007).
57. W3C (2002) Web Services. (available online at <http://www.w3.org/2002/ws/> – last accessed May 2007).
58. W3C (2003) Web Services Architecture. (available online at <http://www.w3.org/TR/ws-arch/> – last accessed May 2007).

Resources

1 Key Books

Baraki D (2002) *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*. Intel Press, Palo Alto, CA.

Bradshaw J (ed.) (1997) *Software Agents*. AAAI/MIT Press, Cambridge, MA.

Buyya R (ed.) (1999) *High Performance Cluster Computing: Architectures and Systems (Vols.1 and 2)*. Prentice Hall, Englewood Cliffs, NJ.

Foster I, Kesselman C (1999) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffman, San Francisco, CA.

Gropp W, Lusk E, Sterling TT (2003) *Beowulf Cluster Computing with Linux (2nd ed)*. MIT Press, Cambridge, MA.

Jensen K (1997) *Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use (Vols.1 and 2)*. Springer-Verlag, Berlin.

Subrayanian R, Goodman BD (eds.) (2005) *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*. IGI Global, Hershey, PA.

2 Key Survey/Review Articles

Baker M, Buyya R, Laforenza D (2002) Grids and Grid Technologies for Wide area Distributed Computing. *Software: Practice and Experience*, 32(15): 1437–1466.

Foster I, Kesselman C (1997) Globus: A Metacomputing Infrastructure Toolkit. *Intl. J. Supercomputer Applications and High Performance Computing*, 11(2): 115–128.

Foster I (2002) What is the Grid? A Three Point Checklist. *Grid Today*, 1(6) (available online at <http://www.gridtoday.com/02/0722/100136.html> – last accessed July 2007).

Karnik N, Tripathi A (1998) Design issues in mobile agent programming systems. *IEEE Concurrency*, 5(3): 52–61.

Shoham Y (1993) Agent-oriented programming. *Artificial Intelligence*, 60(1): 51–92.

3 Journal

Multiagent and Grid Systems (IOS Press)

4 Key International Conferences/Workshops

IEEE/ACM International Conference on Grid Computing (GridXY)

IEEE International Symposium on Cluster Computing and the Grid (Agent-Based Grid Computing at)

IEEE International Conference on High-Performance Computing (HPC)

IEEE/WIC/ACM Intl. Conf. on Intelligent Agent Technology (IAT)

Intl. Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS)

5 Web Resources

FIPA (<http://fipa.org/>) – the standards organization for agents and multi-agent systems – was officially accepted by IEEE at its eleventh Standards Committee on 8th June 2005

Agent Builder
<http://agentbuilders.com/>

Echelon

<http://www.geocities.com/echelongrid/>

Globus Alliance. Globus Toolkit 4.0 (GT4)

<http://www-unix.globus.org/toolkit/docs/4.0/GT4Facts/>

Open Grid Services Infrastructure (OGSI)

<http://www-unix.globus.org/toolkit/>

Repast

<http://repast.sourceforge.net/repastpy/GridAgent.html>

Decentralized Multi-Agent Clustering in Scale-free Sensor Networks

Mahendra Piraveenan, Mikhail Prokopenko, Peter Wang, and Astrid Zeman

CSIRO Information and Communication Technologies Centre, Sydney, Australia*,
Mahendra.Piraveenan@csiro.au, Mikhail.Prokopenko@csiro.au,
Peter.Wang@csiro.au, Astrid.Zeman@csiro.au

1 Introduction

1.1 Multi-Agent Systems and Self-organization

Many interaction processes in complex adaptive systems occur in groups, and in order to organize knowledge, collaboration and a proper distribution of functions and tasks, there is a need to analyze, model and develop computational systems in which several autonomous units interact, adapt and work together in a common open environment, combining individual strategies into overall behavior. The approach to engineering a desired system-level behavior, adopted in this work, is based on a multi-agent system [11], in which the preferred responses emerge as a result of inter-agent interactions.

Multi-agent systems (MAS) represent a new technology to engineer complex adaptive systems. Informally, a MAS is composed of multiple interacting units (agents). Each individual agent can have individual actions, plans, and so on, while all agents work together towards a common goal. It is important to distinguish between agent characteristics and MAS properties. An agent may be described in terms of the following qualities [4, 27, 28]:

- *situatedness* – an agent can receive sensory input from its environment and can perform actions which change the environment in some way; no single agent has access to what everyone else is doing;
- *autonomy* – an agent has control over its own actions and internal state without direct external intervention, and the agents are expected to self-organize and survive on the basis of local, rather than global, information;
- *temporal continuity* – an agent is a continuously running process rather than a function with fixed inputs and outputs;

* Author list is in alphabetical order.

- *adaptability* – an agent makes decisions in accordance with various rules and modifies the rules on the basis of new information;
- *communication* – agents frequently engage in communication with users and each other;
- *multi-platform functionality* – some agents run on low-end platforms, some on high-end platforms.

Some key concepts of MAS are as follows:

- each agent has incomplete capabilities to solve the global problem addressed by the MAS;
- there is no global system control or external coordination;
- data processed by the system is decentralized;
- computation within the system is asynchronous;
- robustness – the system is able to deal with unexpected changes in the environment, and recover from its own and users' errors;
- scalability – the system can be easily extended without a major redesign of its individual parts, in other words, the effort required to extend the system does not increase exponentially with the growth in the number of agents;
- solutions obtained at the system level are not explicitly programmed, and can be interpreted as emergent behavior.

Multi-agent interactions often lead to emergent patterns in overall system behavior [for example, 28]. The emergence of system-level behavior out of agent-level interactions is a distinguishing feature of complex multi-agent systems, making them very different from other complicated multi-component systems, where multiple links among the components may achieve efficient interaction and control with fairly predictable and often pre-optimized properties. However, the concept of emergence is a matter of considerable debate [6]. In particular, emergence is an expected (but not guaranteed) property of self-organization, while the latter is typically defined as the evolution of a system into an organized form in the absence of external pressures. For example, [5] described self-organization as:

“a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components. The rules specifying the interactions among the systems constituent units are executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system by an external ordering influence.”

Despite the huge potential offered by self-organization, a solution based on MAS technology is warranted only when the problem domain is non-trivial, and can be characterized by the following three properties:

- *dynamism* – the problem itself changes concurrently with the problem-solving processes, forcing the latter to adapt;
- *decentralization* – the system’s computational architecture is spatially distributed;
- *computational complexity* – the dimension of the full solution search space is exponential in the dimension of the problem representation, for example, optimization problems such as the Travelling Salesman Problem [8] and the Minimum Energy Broadcast Problem [45].

Informally, the problem has ‘depth’ in three dimensions: time, space, and computation. In the absence of at least one such requirement, it is quite likely that a more conventional approach would be more appropriate. For instance, if a problem is NP-hard and spatially distributed, but static, then it might make sense to establish a predefined hierarchy of problem-solvers that process and channel data to a single point where the global solution is integrated. If a problem is NP-hard and changes in time, but is spatially localized, then again a powerful incremental problem-solver located in a single place is the preferred choice. Finally, if a problem can be solved in polynomial time, but is dynamic and spatially distributed, then a dynamic hierarchy of problem-solvers may be considered. A study of typical trade-offs is described in [34].

1.2 Multi-Agent Networks

A well-known instance of multi-agent systems is a multi-agent network, in particular, a sensor network. A sensor network interconnects (often, wirelessly) multiple spatially distributed autonomous devices (nodes or agents), each capable of sensing, computation and communication, requiring limited memory and power. Typically, a multi-agent network is decentralised. The following summary proposed by [9], captures this requirement with three constraints:

- there is no single central information fusion or coordination centre; no node should be central to the successful operation of the network;
- there is no common communication facility; nodes cannot broadcast results and communication must be kept on a strictly node-to-node basis (although a broadcast medium is often a good model of real communication networks);
- sensor nodes do not have any global knowledge of the sensor network topology; nodes should only know about connections in their own neighbourhood.

The last constraint distinguishes between decentralized systems where each agent still has global information about the group and decentralized systems where an agent has access only to local information from a small subset [22].

Sensor networks may be utilized in various tasks, for instance, monitoring physical or environmental conditions at different locations, search, surveillance, target tracking, mapping and exploration [22]. When networked nodes

are controllable and/or have actuators, one may call such network an active sensor network [21]. Typically, control complexity of large multi-agent networks grows rapidly with the number of agents [37], as well as the number of simultaneous and spatiotemporally distributed real-time events. Thus, self-organizing networks are ideally suited to implementing large sensor networks, being both robust to failures of individual nodes and scalable in terms of the number of detectable events, network size, and so on. Since the overall network behavior is spread over multiple reconfigurable communication paths and interactions, an incremental loss of a portion of the multi-agent network will lead to an incremental loss in quality, rather than a catastrophic failure.

In general, a self-organizing multi-agent network is expected to be:

- *sentient* – relying on perception through sensing (but not necessarily conscious);
- *active* – interrogating/probing the environment, and self-inspecting both locally and globally [29, 35];
- *reconfigurable* – reacting in real time, robust to external and internal fluctuations [32], and adapting to significant change through updating sensor layouts, communication protocols, and power consumption modes;
- *coordinated* – behaving coherently as a dynamical system [36]; fusing the data of individual agents into a joint shared model [9, 30];
- *symbiotic* – recognizing and forming relationships of mutual benefit or dependence among various types of agents (for example, nodes in a sensor network monitoring environment may assist in navigation of multi-robot teams, while being powered by the robots when required) [13].

These desiderata call for an efficient network structure or topology. It is obvious that a fixed topology is not likely to meet such objectives as re-configurability, symbiosis, and the like. Instead, one may consider self-organizing formation of links between nodes, leading to adaptive topologies.

1.3 Adaptive Topologies and Dynamic Hierarchies

Dynamic creation and maintenance of optimal topologies in large dynamic networks is a well-recognized challenge. It appears in many different contexts, for example, as dynamic hierarchies in Artificial Life [32, 38], coalition formation in Agent-based Systems [40], decentralized clustering in Multi-Agent Systems [24], dynamic cluster formation in Mobile Ad Hoc Networks [20], decentralized sensor arrays [25, 26, 31], reconfigurable sensor networks [12, 33], and similar. In this Chapter, we consider a sub-problem from this class: dynamic cluster formation in a sensor and communication network without centralized controllers.

There is a distinction between sensor networks and sensor grids, as pointed out in the recent literature, for instance:

“Whereas the design of a sensor network addresses the logical and physical connectivity of the sensors, the focus of constructing a sensor grid is on the issues relating to the data management, computation management, information management and knowledge discovery management associated with the sensors and the data they generate.” [14]

Dynamic sensor-data clustering is a significant issue addressed by sensor grids. The clustering process is aimed at grouping entities with similar characteristics together so that main trends or unusual patterns may be discovered. In the absence of centralized controllers, this process can be described as *self-organization* of dynamic hierarchies, with multiple cluster-heads emerging as a result of inter-agent communications.

Decentralized clustering algorithms deployed in multi-agent networks are hard to evaluate precisely for the reason of the diminished predictability brought about by self-organization. In particular, it is hard to predict when the cluster formation will converge to a stable configuration. The results presented by [31] identified a predictor for the convergence time of dynamic cluster formation in a specific topology (a rectilinear grid), based on the traffic volume of asynchronous inter-agent communications. The work presented here is an extension of the method to scale-free (sensor) grids/networks. In a scale-free network, some nodes are highly connected in comparison to the rest of the nodes in the network. Properties of scale-free networks have been extensively studied in recent times, since a lot of real world networks seem to fall into this category [1, 10, 17, 46, 47]; we shall briefly introduce this important class of networks in the next Section.

The simple predictor mentioned above is implemented at two levels:

- *the global level*, where full information on nodes’ states and their inter-connections is available, and
- *the local level*, where only partial information is obtained within a small selected subset of nodes.

Quantitative measures of multi-agent dynamics can be used as feedback for evolving agent behaviors [36]. Such measures can use either full information on agent states and their inter-connections, or work with partial information, obtained locally: *localisable measures* [34]. Of course localisable measures can be embedded in the agents themselves and be accessible to selected nodes (for example, hubs), controlling agent behaviors during run-time via adaptive feedback. In general, however, the communication role of a hub should not be confused with its possible control role – in some applications, the only information available at the hub is the number of transiting messages and not their content, and the main decision expected from the hub is a decision on whether to interrupt current multi-agent dynamics without knowing specific details of the exchanged messages.

Our immediate goal is predicting when the cluster formation will converge to a stable configuration. In achieving this goal, we consider an underlying time series, the traffic volume of inter-agent communications, and relate its irregularity during an initial interval to the eventual convergence time. Clearly, the shorter the initial interval, the more efficient is the prediction: for instance, when a predicted value exceeds a threshold, agents may adjust parameters and heuristics used in the clustering process.

A simplified version of a decentralized adaptive clustering algorithm operating within a scale-free network, developed for evaluation purposes, is described in the next Section. The proposed predictor for the convergence time of cluster formation is then described, followed by a discussion of the obtained results.

2 Dynamic Cluster Formation Algorithm

The dynamic cluster formation algorithm has been implemented in a scale-free topology. In a scale-free topology, it is not uncommon to find nodes with a degree (the number of connections from a node) that is much higher than the average degree of that network. These highly connected nodes are called *hubs* and can play specific roles in their networks, depending on the network domain. Hubs are often formed by a growth model that shows preferential attachment. That is, when a new node is attached to the network, it is more likely to be attached to a node with a higher degree. A general model of this type of growth is that the probability of a new node being attached to a particular existing node is proportional to the number of connections from that existing node [3]. According to this model, starting from m_0 vertices (that are typically fully connected), at each iteration a new vertex with $m \leq m_0$ edges is attached to old vertices in such a way that the probability of being connected to the existing vertex i is proportional to the degree k_i , and is set to $\frac{k_i}{\sum k_i}$, where the sum is computed over all nodes. If the parameter $m = 1$, then the growth results in a scale-free *tree graph*; otherwise, if $m > 1$, then a scale-free *network* is produced.

A degree distribution (sometimes called *vertex degree distribution*) is the probability distribution function describing the total number of vertices in a network with a given degree. The degree distribution of a scale-free network follows a power law, in the form of $p(x) \approx k^{-\gamma}$, where k is the degree. The power index γ is usually between 2.1 and 3.0 for most biological, social and technological networks [42]. Some scale-free networks may have rapid cut-offs after a certain degree, so that the degree distribution takes the form of $p(x) \approx k^{-\gamma} \phi(k/\xi)$, where $\phi(k/\xi)$ is the step function [42] which introduces a cut-off at some characteristic scale ξ . When ξ is very small, $p(x) \approx k^{-\gamma} \phi(k/\xi)$ and the degree distribution is single-scaled. As ξ grows, a power law with a sharp cut-off is obtained, while scale-free nets are observed for large ξ .

Scale-free networks generally display the *small world* phenomenon, in that the average distance between any two vertices is very small, compared to a regular or randomly connected network. Scale-free networks also tend to have higher clustering coefficients. The clustering coefficient C can be defined as the probability of two nodes individually connected to a particular third node being connected to each other [23]. Formally,

$$C = 3 \left(\frac{\text{number of triangles in the graph}}{\text{number of connected triples of vertices in the graph}} \right) \quad (1)$$

where the multiplier 3 indicates that one triangle accounts for three individual nodes that are each connected to two other nodes.

Scale-free networks are generally robust against random attacks, but highly vulnerable against targeted attacks. In other words, removal of random nodes will only slightly affect functionality of the networks, whereas removal of a hub will *drastically* affect network functionality [2]. A sensor grid node within a network communicates only with immediate neighbours: all data are processed locally, and only information relevant to other regions of the grid is communicated as a multi-hop message. A cluster-head may be dynamically selected among the set of nodes and become a local coordinator of transmissions within the cluster. The intersection of cluster-heads and hubs of the scale-free network may be empty – in other words, a cluster-head does not need to have many network connections. On the other hand, a non-hub cluster-head would generate more intense communication traffic. Clusters may adapt, that is, re-form when new data is obtained on the basis of local sensor signals. Importantly, a cluster formation algorithm should be robust to such changes, failures of individual nodes, communication losses, and the like.

As pointed out earlier, our main goal is an analysis of a representative clustering technique in a dynamic and decentralized multi-agent setting, deployed within a scale-free sensor grid, *in terms of the predictability of its convergence time*. We represent a node’s sensory reading with a single aggregated value, define ‘differences’ between cells in terms of this value, and cluster the nodes while minimizing these ‘differences’.

The algorithm input is a series of events detected at different times and locations, while the output is a set of non-overlapping clusters, each with a dedicated cluster-head (network node) and a cluster map of its followers in terms of their sensor-data and relative grid coordinates. The algorithm is described in Appendix-A, and involves a number of inter-agent messages notifying agents about their sensory data, together with changes in their relationships and actions. For example, an agent may send a *recruit* message to another agent, delegate the role of cluster-head to another agent, or declare ‘independence’ by initiating a new cluster.

Most of these and similar decisions are based on the clustering heuristic described by [24], and a dynamic offset range introduced by [26]. This heuristic

determines if a cluster should be split in two, as well as the location of this split. Each cluster-head (initially, each agent) broadcasts its recruit message periodically, with a broadcasting period, affecting all agents with values within a particular dynamic offset of the sensor reading detected by this agent. Every recruit message contains the sensor data of all current followers of the cluster-head with their relative coordinates (a cluster map). Under certain conditions, an agent (which is not a follower in any cluster) receiving a *recruit* message becomes a follower, stops broadcasting its own *recruit* messages and sends its information to its new cluster-head indicating its relative coordinates and the sensor reading. However, there are situations when the receiving agent is already a follower in some cluster and cannot accept a recruit message by itself – a recruit disagreement. In this case, this agent forwards the received recruiting request to its present cluster-head. Every cluster-head waits for a certain period, collecting all such *forward* messages, at the end of which the clustering heuristic is invoked on the union set of present followers and all agents who forwarded their new requests [26, 31].

Firstly, all n agents in the combined list are sorted in decreasing order according to their sensor reading value x . Then, a series of all possible divisions in the ordered set of agents is generated. That is, the first ordering is a cluster with all agents in it; the second ordering has the agent with the largest value in the first cluster and all other agents in the second cluster; and so forth (the n th division has only the last n th agent in the second cluster). For each of these divisions, the quality of clustering is measured by the total squared error:

$$E_j^2 = \sum_{i=1}^z \sum_{x \in A_{i,j}} \|x - m_{i,j}\|^2 \quad (2)$$

where z is a number of considered clusters ($z = 2$ when only one split is considered), $A_{i,j}$ are the clusters resulting from a particular division, and $m_{i,j}$ is the mean value of the cluster $A_{i,j}$. We divide E^2 values by their maximum to get a series of normalized values. Then we approximate the second derivative of the normalized errors per division:

$$f''(E_j^2) = \frac{(E_{j+1}^2 + E_{j-1}^2 - 2E_j^2)}{h^2} \quad (3)$$

where $h = 1/n$.

If the peak of the second derivative is greater than some threshold for the division j , we split the set accordingly; otherwise, the set will remain as one cluster. When the clustering heuristic is applied, it may produce either one or two clusters as a result. If there are two clusters, the offset of each new cluster-head is modified. It is adjusted in such a way that the cluster-head of the ‘smaller’ agents (henceforth, references like ‘larger’ or ‘smaller’ are relative to the value x) can now reach up to, but not including, the ‘smallest’ agent in the cluster of ‘larger’ agents. Similarly, the cluster-head of ‘larger’ agents can

now reach down to, but not including, the ‘largest’ agent (the cluster-head) of the cluster of ‘smaller’ agents. These adjusted offsets are sent to the new cluster-heads along with their cluster maps.

The cluster-head which invoked the heuristic notifies new cluster-heads about their appointment, and sends their cluster maps to them: a *cluster-information* message. There are other auxiliary messages involved in the algorithm but importantly, the cluster formation is driven by three types: *recruit*, *cluster-information*, and *forward* messages. The first two types are periodic, while the latter type depends only on the degree of disagreements among cluster-heads. On the one hand, if there are no disagreements in the clustering (for instance, if a clustering heuristic resulted in optimal splits even with incomplete data), then there is no need to forward messages. On the other hand, when cluster-heads frequently disagree on formed clusters, the forward messages are common. In short, it is precisely the number of forward messages traced in time – the traffic volume of inter-agent communications – that we hope may provide an underlying time series $\{v(t)\}$ for our prognostic analysis, as it exhibits both periodic and chaotic features.

The quality of clustering is measured by the weighted average cluster diameter [49]. The average pair-wise distance D for a cluster C with points $\{x_1, x_2, \dots, x_m\}$ is given by

$$D = \frac{\sum_{i=1}^m \sum_{j=1}^m d(x_i, x_j)}{m(m-1)/2} \quad (4)$$

where $d(x_i, x_j)$ is the Euclidean distance between points x_i and x_j . The weighted average cluster diameter for k clusters is given by

$$\bar{D} = \frac{\sum_{i=1}^k m_i(m_i - 1)D_i}{\sum_{i=1}^k m_i(m_i - 1)} \quad (5)$$

where m_i is the number of elements in the cluster C_i with pair-wise distance D_i . This metric is known to scale well with the size of data points and number of clusters in a particular clustering. It does not, however, account for singleton clusters, while at the same time favouring small clusters.

As pointed out by [26], the algorithm does not guarantee a convergence minimizing this criterion. In fact, it may give different clusterings for the same set of agent values, depending on the relative node locations within the network. The reason is a different communication flow affecting the adjustment of the offsets. Each time the clustering heuristic is executed in an agent, its offsets are either left alone or reduced. The scope of agents involved in the clustering heuristic depends on the order of message passing, which in turn

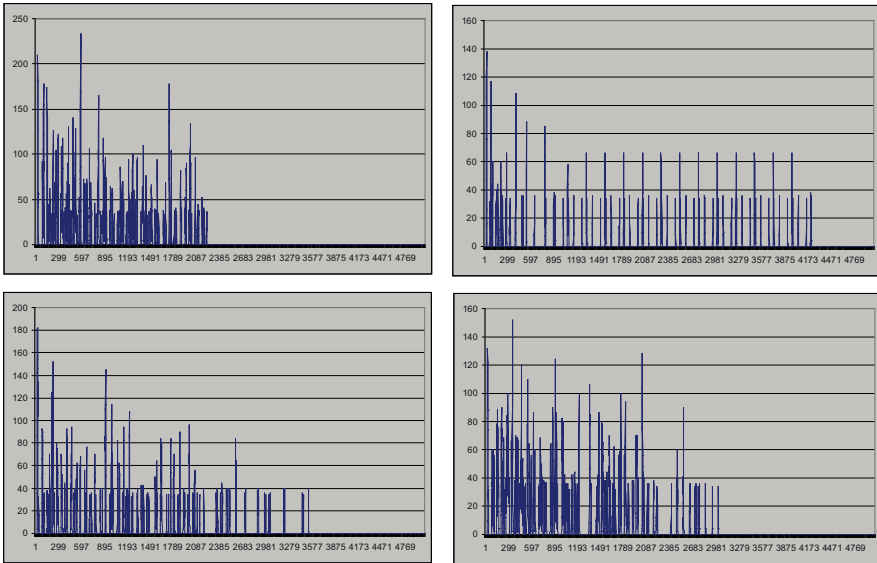


Fig. 1. Varying convergence times T for different experiments tracing the whole communication space

depends on the relative node locations. The adjusted offsets determine which agents can be reached by a cluster-head, and this will affect the result of clustering. Therefore, for any set of agent values, there are certain sequences of events which yield better clustering results than others.

We conducted extensive simulations to determine whether the algorithm is robust and scales well in terms of the quality of clustering and convergence, as measured by the number of times the clustering heuristic was invoked before stability is achieved with each data set – both for rectilinear grids [26] and scale-free networks. While the simulation results show that the algorithm converges and scales well in all cases, and in addition, is robust to dynamics of the sensor data flux, the convergence time varies significantly (Figs. 1 and 2), without obvious indicative patterns – highlighting the need for its better prediction.

3 Regularity of Multi-Agent Communication-Volume

In this Section, we focus on our main objective: prediction of the convergence time T , based on regularity of an initial segment $0, \dots, \Omega$ of the ‘communication-volume’ series $\{v(t)\}$, where $\Omega < T$ and $v(t)$ is the number of forward messages at time t . The series $\{v(t)\}$ may be obtained by monitoring the whole communication space, or by monitoring the communication messages through only selected nodes. Given the role played within the scale-free

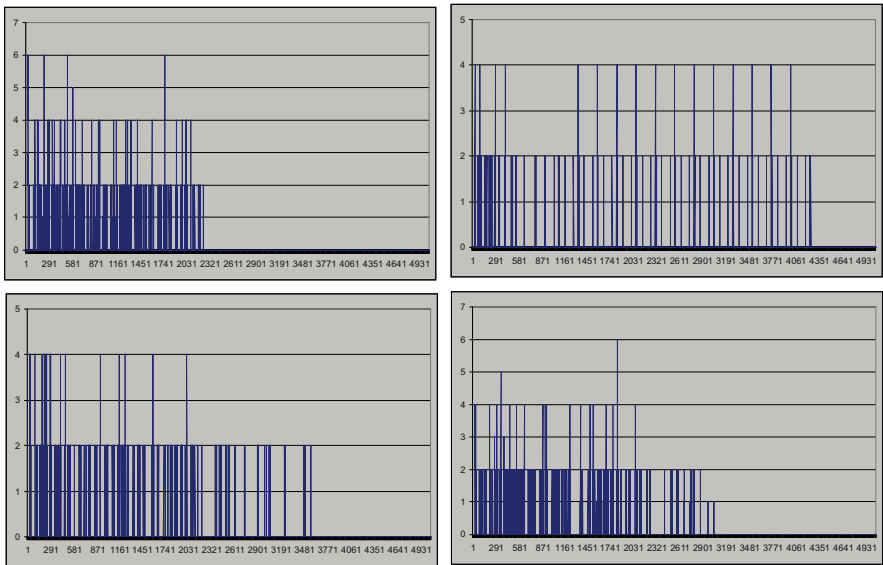


Fig. 2. Varying convergence times T for the same experiments as shown in Fig. 1, but tracing communication traffic through the highest-ranked hub only

network by hubs, we rank the nodes by the number of their connections, and monitor every node, producing multiple series $\{v_h(t)\}, 1 < h < H$, where H is the total number of nodes in the network. Thus, each series $\{v_h(t)\}$ traces the number of messages passed through the first h nodes ranked by the number of connections – for instance, $\{v_1(t)\}$ traces the number of messages passed through the hub with the most connections; $\{v_2(t)\}$ combines the number of messages passed through the first two hubs with the highest number of connections; and $\{v_H(t)\}$ is identical to $\{v(t)\}$, as it traces the messages through all the nodes in the network. The idea is then to determine whether monitoring only a subset of nodes, ideally with h being small, is almost as good as monitoring the whole network.

It is known that in many experiments, time series often exhibit irregular behavior during an initial interval before finally settling into an asymptotic state which is non-chaotic [7] – in our case, eventually converging to a fixed-point ($v(T) = 0$; henceforth, we shall drop the subscript h if it does not matter which series is being considered). The irregular initial part of the series may, nevertheless, contain valuable information: this is particularly true when the underlying dynamics are deterministic and exhibit ‘transient chaos’ [7, 16]. It was conjectured and empirically verified [31] that the described algorithm for dynamic cluster formation creates multi-agent transient chaotic dynamics.

[31] used the Kolmogorov-Sinai entropy K , also known as metric entropy [19, 41], and its generalization to the order- q Rényi entropy K_q [39]. The

entropy K or K_q is an entropy per unit time, or an ‘entropy rate’, and is a measure of the rate at which information about the state of the system is lost in the course of time. In particular, the predictor estimated the ‘correlation entropy’ K_2 using the algorithm of [15]. The predictor based on K_2 uses the initial segment of length Ω of the observed time series $\{v(t)\}$ in ‘converting’ or ‘reconstructing’ the dynamical information in one-dimensional data to spatial information in the τ -dimensional embedding space [43], and also depends on the length Ω and the embedding dimension τ . The method for computing the K_2 predictor is described in Appendix-B.

The predictor based on K_2 was used in predicting convergence of cluster formation within rectilinear grids. Here we apply this method to cluster formation in scale-free sensor networks.

For each experiment s , we

- (a) select an initial segment of length Ω of the time series; and
- (b) compute the regularity predictor: the correlation entropy $K_2(d, r, \Omega)$ for a range of embedded dimensions d and a suitable precision r (see Appendix-B for details).

Then,

- (c) given the estimates $K_2(d, r, \Omega)$ for all the experiments, we correlate them with the observed convergence times T_s by linear regression $T = a + bK_2$ and the corresponding correlation coefficient $\rho(d, r, \Omega)$ between the series T_s and $K_2(d, r, \Omega)_s$;
- (d) we determine the embedding dimension \hat{d} and the distance \hat{r} which provide the best fit: the maximum of $\rho(d, r, \Omega)$.

This allows us to predict the time T of convergence to $\nu(T) = 0$, as

$$T = a(\hat{d}, \hat{r}, \Omega) + b(\hat{d}, \hat{r}, \Omega)K_2(d, r, \Omega) \quad (6)$$

for any real-time run that produced the predictor value $K_2(d, r, \Omega)$.

4 Experimental Results

The experiments included multiple scenarios, each of which was defined by a specific scale-free tree graph, Φ (more precisely, a specific degree distribution in the corresponding graph); the number of nodes in the network, H ; and the number of events sensed by network, N . For example, two different scale-free tree graphs Φ_1 and Φ_2 , both with 400 nodes and 100 events, may be compared (Figs. 3 and 4). In particular, we considered different values for N (for example, N was set to $1/8$, $1/4$, $1/2$ of H), given the same network Φ with fixed H . This approach evaluated the predictor with respect to the dynamics in the communication space brought about by multiple events. In addition,

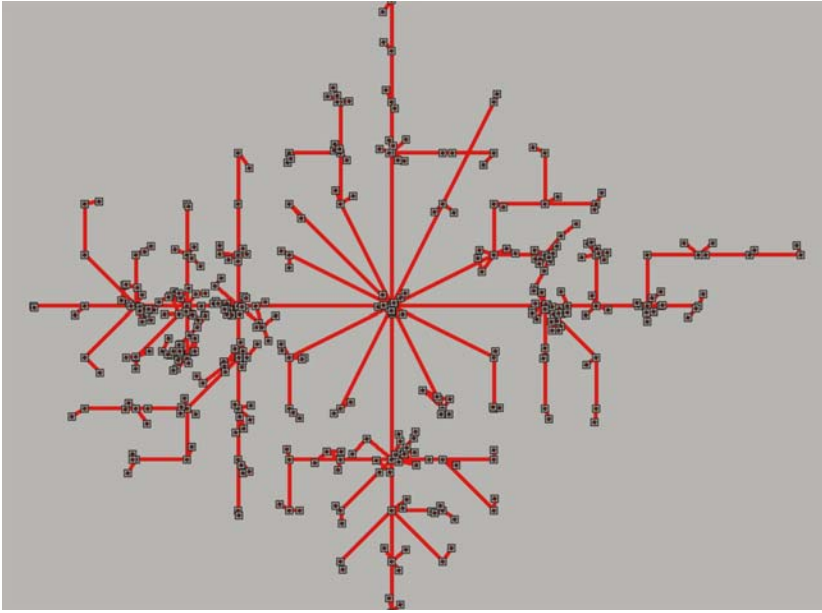


Fig. 3. A scale-free tree graph Φ_1 used in experiments

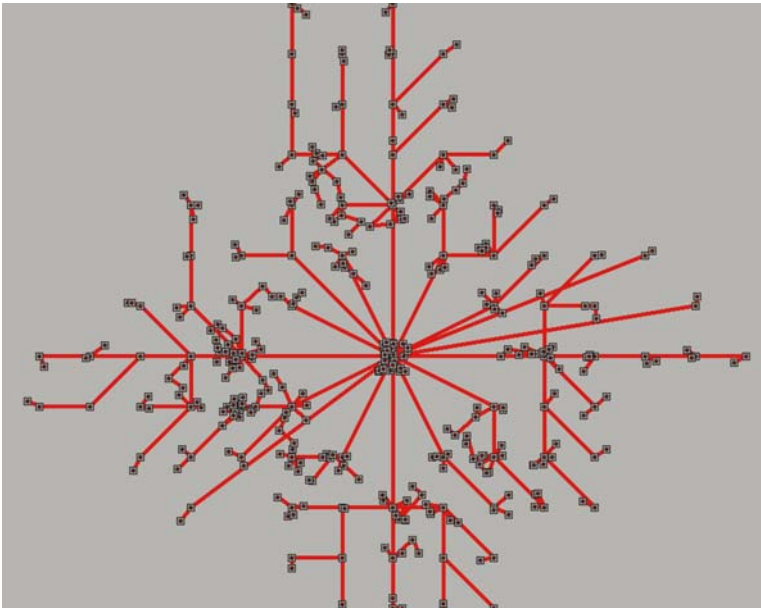


Fig. 4. A scale-free tree graph Φ_2 used in experiments

the predictor was verified by varying degree distributions Φ for a fixed number H – in other words, by considering a different network of the same size. Again, the number of events N was increased as a proportion of H . Finally, we verified the results by increasing the network size H .

In order to estimate the statistical significance of the results, each scenario (fixed Φ, H, N) included 100 runs of the clustering algorithm on a scale-free tree graph, where every run involved N events in random network locations, tracing the communication-volume time series $\{v(t)\}$, as well as multiple series $\{v_h(t)\}, 1 < h < H$. We then selected an initial segment $\Omega = 1500$ (while the longest run is 5000) and carried out the steps b), c) and d) described previously in Sect. 3. These runs produced a 2-dimensional array $K_2(d, r, \Omega)$ for varying dimensions d and precisions r and each run $s (s = 1, \dots, 100)$. Given the array, the correlation coefficient $\rho(d, r, \Omega)$ between the actual convergence time T_s (standardized series) and the auto-correlation predictor $K_2(d, r, \Omega)$ (standardized series) was determined for the ranges of d and r . The higher the maximal correlation coefficient $\rho(d, r, \Omega)$ is, the more predictive power is contained in the predictor $K_2(d, r, \Omega)$.

The correlation coefficient $\rho(d, r, \Omega)$ is obviously decreased to $\rho_h(d, r, \Omega)$ as the series $\{v(t)\}$ is replaced with the series $\{v_h(t)\}$, if $h < H$. We observed, however, that the difference between $\rho(d, r, \Omega)$ and $\rho_h(d, r, \Omega)$ is insignificant when $h \geq 1$, for at least one embedding dimension. In other words, monitoring a single highest-ranked hub (or two highest-ranked hubs) is sufficient in order to predict the convergence of cluster formation in a scale-free tree graph. This result holds for all the considered scenarios, and supports our conjecture that a localisable predictor is feasible, although in general it is harder to maintain predictability when the number of events is large.

Figure 5 plots the correlation coefficients $\rho(d, r, \Omega)$ for a range of dimensions d and different precisions r in the scenario ($\Phi = \Phi_1, H = 400, N = 50$). The precision $r = 10$ (messages) yields the highest predictive power (shown as the plot with squares). Evaluation of the precision $r = 10$ was continued with higher dimensions d . Figures 6, 8 and 9 plot the correlation coefficients $\rho(d, r, \Omega)$ for a range of dimensions d (and the best precision r) in the scenarios ($\Phi = \Phi_1, H = 400, N = 50$), ($\Phi = \Phi_1, H = 400, N = 100$), and ($\Phi = \Phi_1, H = 400, N = 200$), respectively – based on the series $\{v(t)\}$ tracing the whole communication-space and the series $\{v_1(t)\}$ at the highest-ranked hub only. All results are statistically significant (significance levels of 0.999), and are within reasonable confidence limits (Fig. 7). The results with the alternative network $\Phi = \Phi_2$, as well as different network sizes H , are analogous, and also support the localized predictor based on the series $\{v_1(t)\}$.

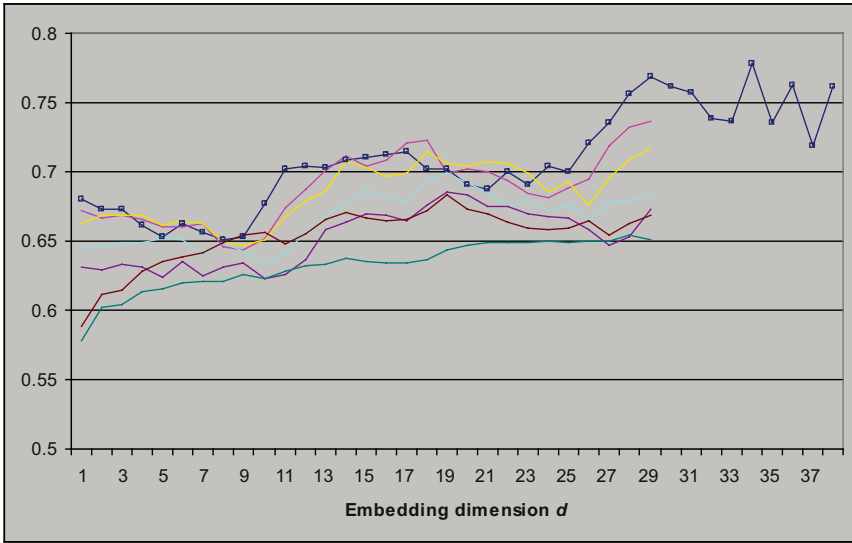


Fig. 5. Correlation coefficient $\rho(d, r, \Omega)$ between series T_s and predictor $K_2(d, r, \Omega)$, for the scenario $(\Phi = \Phi_1, H = 400, N = 50)$, based on series $\{v(t)\}$ tracing the whole communication-space, for a range of precisions r . The precision $r = 10$ (messages) yields the highest predictive power (shown as the plot with squares). Evaluation of the precision $r = 10$ was continued with higher dimensions d

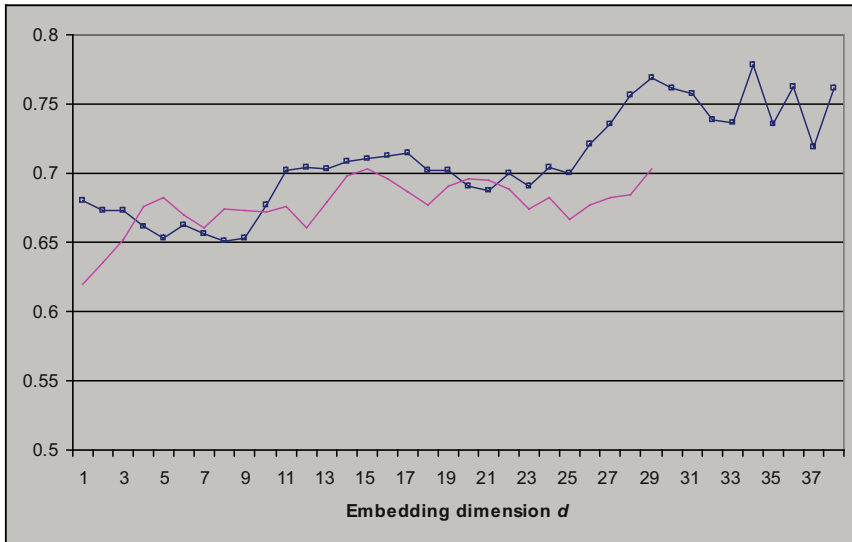


Fig. 6. Correlation coefficient $\rho(d, r, \Omega)$ between series T_s and predictor $K_2(d, r, \Omega)$, for the scenario $(\Phi = \Phi_1, H = 400, N = 50)$, based on series $\{v(t)\}$ tracing the whole communication-space, shown as the plot with squares ($r = 10$), and series $\{v_1(t)\}$ traced at the highest-ranked hub ($r = 2$)

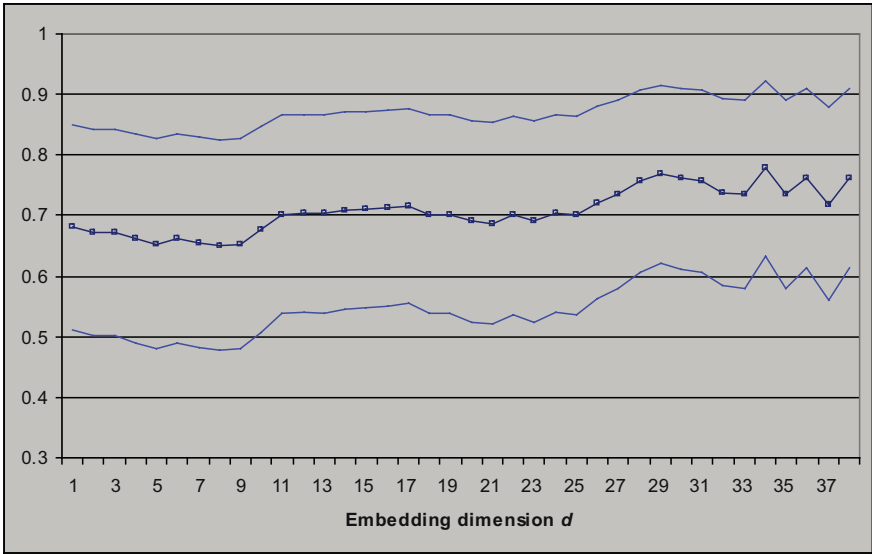


Fig. 7. Confidence limits of the correlation coefficient $\rho(d, r, \Omega)$ between the series T_s and predictor $K_2(d, r, \Omega)$, for the scenario $(\Phi = \Phi_1, H = 400, N = 50)$, based on the series $\{v(t)\}$ tracing the whole communication-space, shown as the plot with squares ($r = 10$)

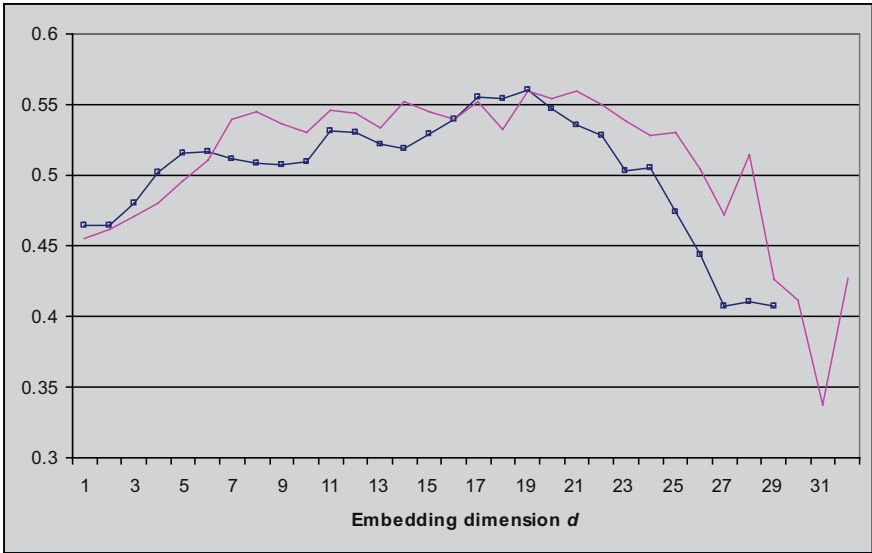


Fig. 8. Correlation coefficient $\rho(d, r, \Omega)$ between series T_s and predictor $K_2(d, r, \Omega)$, for the scenario $(\Phi = \Phi_1, H = 400, N = 100)$, based on series $\{v(t)\}$ tracing the whole communication-space (the plot with squares; precision $r = 20$), and series $\{v_1(t)\}$ traced at the highest-ranked hub ($r = 2$)

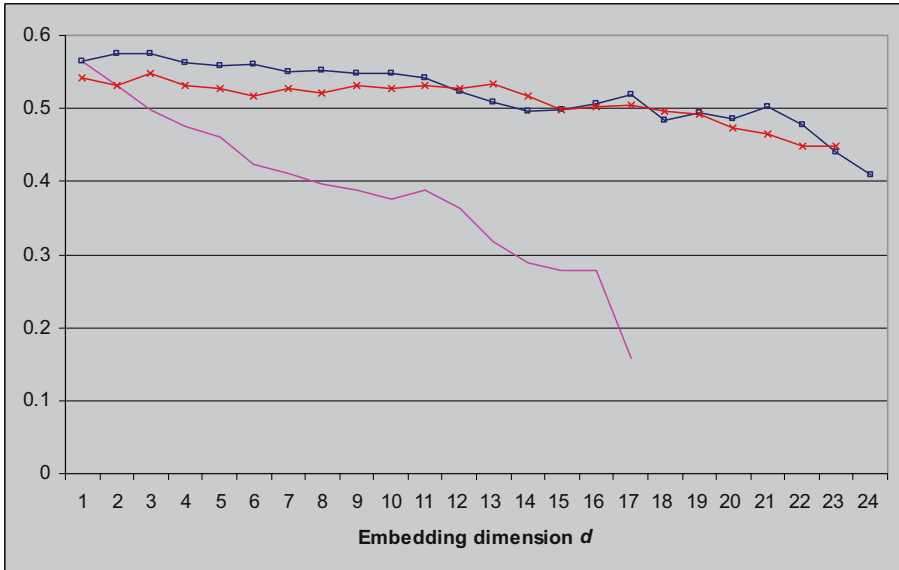


Fig. 9. Correlation coefficient $\rho(d, r, \Omega)$ between series T_s and predictor $K_2(d, r, \Omega)$, for the scenario ($\Phi = \Phi_1, H = 400, N = 200$), based on series $\{v(t)\}$ (squares: precision $r = 40$), the series $\{v_1(t)\}$ traced at the highest-ranked hub ($r = 2$), and series $\{v_2(t)\}$ traced at two highest-ranked hubs (crosses: $r = 4$)

5 An Application Scenario – Distributed Energy Management and Control

Distributed energy refers to the generation of power (for heating and cooling) within close proximity to the point of use. CSIRO is developing a range of small scale distributed energy technologies [18] based on both renewable and fossil fuels (mainly natural gas). A major part of this program is identifying the most efficient ways of integrating large numbers of small generation plants (including solar and wind) into the existing power networks to deliver maximum efficiency with minimum environmental impacts.

A decentralized approach to the problem of power load management is described [48], using modeling of direct load management as a computational market. A load, in this context, is any device that consumes electric energy, such as a water heater or an electric motor. Load management involves controlling the loads at the demand side to achieve a better use of energy: better for the utility, the customer or both. [48] define *direct load management* as a process when the utility determines what loads are to be connected, reduced, or disconnected at specific occasions, and contrast it with *indirect load management* when the utility sends some signal to customers, such as price information, and expects them to adjust to this signal. Decentralized multi-agent algorithms become usable in power load management due to the

inherent parallelism of the underlying network increasing the computational power. In addition, according to [48]:

“from the energy utility point of view it is desirable to have a system that hides most details of the different loads while still providing enough information for energy optimization. The system should be able respond to high level control commands for – for example, reduction of the current load in the distribution system by a certain amount”.

The US Department of Energy’s recent report to Congress on demand response [44] notes that:

“if you take a system balancing and reliability perspective, active demand gives you another set of tools, another resource on which you can call to enable system balancing to avoid triggering capacity constraints and involuntary interruptions. Furthermore, double-sided markets have important economic efficiency benefits as well as system reliability benefits”.

Typically, an efficient demand response is provided by a multi-agent coalition (a cluster of agents), comprising both generators and loads, sharing/fusing some information while trying to cooperatively solve a distributed decentralized problem. We believe that the efficient methods of decentralized dynamic clustering deployed in scale-free power grids will increase the efficiency of the overall solution.

6 Conclusions

We considered decentralized and dynamic cluster formation in scale-free multi-agent sensor grids, and described and experimentally evaluated a predictor for the convergence time of cluster formation. The new predictor estimates regularity of the inter-agent communication space via the ‘correlation entropy’ (the order-2 Rényi entropy) K_2 , and was observed to be well correlated with the time of cluster formation.

The predictor is implemented and experimentally evaluated at the global level, where full information on nodes’ states and their inter-connections is available, as well as at the local level, using only partial information obtained within a small selected subset of nodes. In other words, the predictor K_2 does not have to employ full information on nodes’ states and their inter-connections – instead it may use only partial information obtained within a small selected subset of nodes. Thus, the analysis and presented results support the deployment of localisable predictors monitoring a small part of the inter-agent communication space – the part contained within the most connected hubs of the scale-free sensor network.

Efficient and reliable algorithms for cluster formation in sensor grids may include a convergence predictor, such as predictor K_2 , as a feedback to the algorithms, and this is a subject for future research. Another direction is an evaluation of other decentralized algorithms in terms of multi-agent dynamics within the communication spaces.

References

1. Albert R, Jeong H, Barabási A-L (1999) Diameter of the world-wide web. *Nature*, 401: 130–131.
2. Albert R, Jeong H, Barabási A-L (2000) Error and attack tolerance of complex networks. *Nature*, 406: 378–382.
3. Albert R, Barabási A-L (2002) Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1): 47–97.
4. Butler M, Prokopenko M, Howard T (2001) Flexible synchronisation within RoboCup environment: a comparative analysis. In: Stone P, Balch TR, Kraetzschmar GK (eds) *RoboCup 2000: Robot Soccer World Cup IV (Proc. 4th RoboCup-2000 Workshop, 31 August – 1 September, Melbourne, Australia)*, Lecture Notes in Computer Science 2019: 119–128, Springer-Verlag, Berlin.
5. Bonabeau E, Theraulaz G, Deneubourg J-L, Camazine S (1997) Self-organisation in social insects. *Trends in Ecology and Evolution*, 12(5): 188–193.
6. Boschetti F, Prokopenko M, Macreadie I, Grisogono A-M (2005) Defining and detecting emergence in complex networks. In: Khosla R, Howlett RJ, Jain LC (eds) *Proc. 9th Intl. Conf. Knowledge-Based Intelligent Information and Engineering Systems – KES 2005*, 14–16 September, Melbourne, Australia, Lecture Notes in Computer Science 3684: 573–580, Springer-Verlag, Berlin.
7. Dhamala M, Lai YC, Kostelich EJ (2001) Analyses of transient chaotic time series. *Physical Review E*, 64: 1–9.
8. Dorigo M, Gambardella LM (1997) Ant colonies for the Traveling Salesman Problem. *BioSystems*, 43: 73–81.
9. Durrant-Whyte, HF, Stevens M (2001) Data fusion in decentralised sensing networks. *Proc. 4th Intl. Conf. Information Fusion*, 7–10 August, Montreal, Canada, International Society of Information Fusion, Sunnyvale, CA.
10. Faloutsos M, Faloutsos P, Faloutsos C (1999) On power-law relationships of the internet topology. *Computer Communication Review*, 29: 251–262.
11. Ferber J (1999) *Multi-Agent Systems*. Addison Wesley Professional, Reading, MA.
12. Foreman M, Prokopenko M, Wang P (2003) Phase transitions in self-organising sensor networks. In: Banzhaf W, Christaller T, Dittrich P, Kim JT, Ziegler J (eds) *Advances in Artificial Life (Proc. 7th European Conf. Artificial Life – ECAL2003)*, 14–17 September, Dortmund, Germany, Lecture Notes in Artificial Intelligence 2801: 781–791, Springer-Verlag, Berlin.
13. Gerasimov V, Healy G, Prokopenko M, Wang P, Zeman A (2006) Symbiotic sensor networks in complex underwater terrains: a simulation framework. In: Gabrys B, Howlett RJ, Jain LC (eds) *Proc. 10th Intl. Knowledge-Based Intelligent Information and Engineering Systems Conf. – KES 2006*, 9–11 October,

- Bournemouth, UK, Lecture Notes in Artificial Intelligence 4253(III): 315–323, Springer-Verlag, Berlin.
14. Ghanem M, Guo Y, Hassard J, Osmond M, Richards M (2004) Sensor grid for air pollution monitoring. In: Cox SJ (ed) *Proc. 3rd UK e-Science All-hands Conf. – AHM 2004*, 31 August – 3 September, Nottingham, UK, Engineering and Physical Sciences Research Council (EPSRC), UK: 106–113.
 15. Grassberger P, Procaccia I (1983) Estimation of the Kolmogorov entropy from a chaotic signal. *Physical Review A*, 28(4): 2591–2593.
 16. Jánosi IM, Tél T (1994) Time series analysis of transient chaos. *Physical Review E*, 49(4): 2756–2763.
 17. Jeong H, Tombor B, Albert R, Oltvai ZN, Barabási, A-L (2000) The large-scale organization of metabolic networks. *Nature*, 407: 651–654.
 18. Jones T, James G (2005) The management and control of distributed energy resources (extended version). In: *Proc. 18th Intl. Conf. and Exhibition on Electricity Distribution – CIRED*, June 2005, Turin, Italy, IEE, London, UK: 987–998.
 19. Kolmogorov AN (1959) Entropy per unit time as a metric invariant of automorphisms (in Russian). *Doklady Akademii Nauk SSSR*, 124: 754–755.
 20. Lin R, Gerla M (1997) Adaptive clustering for mobile wireless networks. *IEEE J. Selected Areas in Communications*, September: 1265–1275.
 21. Makarenko A, Kaupp T, Grocholsky B, Durrant-Whyte HF (2003) Human-robot interactions in active server networks. In: *Computational Intelligence in Robotics and Automation for the New Millennium* (Proc. 2003 IEEE Intl. Symposium Computational Intelligence in Robotics and Automation), 16–20 July, Kobe, Japan, IEEE Computer Society Press, Piscataway, NJ, 1: 247–252.
 22. Mathews GM, Durrant-Whyte HF, Prokopenko M (2006) Scalable decentralised decision making and optimisation in heterogeneous teams. In: *Proc. IEEE Intl. Conf. Multisensor Fusion and Integration for Intelligent Systems – MFI2006*, 3–6 September, Heidelberg, Germany, IEEE Computer Society Press, Piscataway, NJ: 383–388.
 23. Newman MEJ (2002) The structure and function of networks. In: Hossfeld F, Binder E (eds) *Proc. Europhysics Conf. Computational Physics – CCP2001*, 5–8 September, 2001, Aachen, Germany, Elsevier Science, Amsterdam, 147(1–2): 40–45.
 24. Ogston E, Overeinder B, Van Steen M, Brazier F (2003) A Method for decentralized clustering in large multi-agent systems. In: Rosenschein JS, Sandholm T, Wooldridge M, Yokoo M (eds) *Proc. 2nd Intl. Joint Conf. Autonomous Agents and Multi-Agent Systems*, 14–18 July, Melbourne, Australia, ACM Press, New York, NY: 798–796.
 25. Olsson L, Nehaniv CL, Polani D (2004) Sensory channel grouping and structure from uninterpreted sensor data. In: Zebulum RS, Gwaltney D, Hornby G, Keymeulen D, Lohn J, Stoica A (eds) *Proc. NASA/DoD Conf. Evolvable Hardware – EH’04*, 24–26 June, Seattle, WA, IEEE Computer Society Press, Los Alamitos, CA: 153–160.
 26. Piraveenan M, Prokopenko M, Wang P, Price DC (2005) Towards adaptive clustering in self-monitoring multi-agent networks. In: Khosla R, Howlett RJ, Jain LC (eds) *Proc. 9th Intl. Conf. Knowledge-Based Intelligent Information and Engineering Systems – KES’2005*, 14–16 September, Melbourne, Australia. Lecture Notes in Computer Science 3682(II), Springer-Verlag, Berlin: 796–805.

27. Prokopenko M (1999) On situated reasoning in multi-agent systems. In: *Hybrid Systems and AI: Modeling, Analysis and Control of Discrete and Continuous Systems*, AAAI Technical Report SS-99-05, March, AAAI Press, Menlo Park, CA: 158–163.
28. Prokopenko M, Butler M, Howard T (2001) On emergence of scalable tactical and strategic behavior. In: Stone P, Balch TR, Kraetzschmar GK (eds) *RoboCup 2000: Robot Soccer World Cup IV (Proc. 4th RoboCup-2000 Workshop)*, 31 August – 1 September, Melbourne, Australia, Lecture Notes in Computer Science 2019, Springer-Verlag, Berlin: 357–366.
29. Prokopenko M, Wang P (2004) On self-referential shape replication in robust aerospace vehicles. In: Pollack J, Bedau MA, Husbands P, Ikegami T, Watson RA (eds) *Artificial Life IX (Proc. 9th Intl. Conf. Simulation and Synthesis of Living Systems)*, 12–15 September, Boston, MA, MIT Press, Cambridge, MA: 27–32.
30. Prokopenko M, Wang P (2004) Evaluating team performance at the edge of chaos. In: Polani D, Browning B, Bonarini A, Yoshida K (eds) *RoboCup 2003: Robot Soccer World Cup VII (Proc. 7th RoboCup-2003 Springer-Verlag, Berlin:Symposium)*, Padua, July, Lecture Notes in Computer Science 3020: 89–101.
31. Prokopenko M, Piraveenan M, Wang P (2005) On convergence of dynamic cluster formation in multi-agent networks. In: Capcarrère MS, Freitas AA, Bentley PJ, Johnson CG, Timmis J (eds) *Advances in Artificial Life (Proc. 8th European Conference – ECAL 2005)*, 5–9 September, Canterbury, UK. Lecture Notes in Computer Science 3630, Springer-Verlag, Berlin: 884–894.
32. Prokopenko M, Wang P, Price DC, Valencia P, Foreman M, Farmer AJ (2005) Self-organising hierarchies in sensor and communication networks. *Artificial Life* (Special issue on Dynamic Hierarchies), 11(4): 407–426.
33. Prokopenko M, Wang P, Foreman M, Valencia P, Price DC, Poulton G (2005) On connectivity of reconfigurable impact networks in ageless aerospace vehicles. *J. Robotics and Autonomous Systems*, 53: 36–58.
34. Prokopenko M, Wang P, Price DC (2005) Complexity metrics for self-monitoring impact sensing networks. In: Lohn J, Gwaltney D, Hornby G, Zebulum R, Keymeulen D, Stoica A (eds) *Proc. NASA/DoD Conf. Evolvable Hardware – EH-05*, 29 June – 1 July, Washington, DC, IEEE Computer Society Press, Los Alamitos, CA: 239–246.
35. Prokopenko M, Poulton GT, Price DC, Wang P, Valencia P, Hoschke N, Farmer AJ, Hedley M, Lewis C, Scott DA (2006) Self-organising impact sensing networks in robust aerospace vehicles. In: Fulcher, J (ed) *Advances in Applied Artificial Intelligence*. Idea Group, Hershey, PA: 186–223.
36. Prokopenko M, Gerasimov V, Tanev I (2006) Evolving spatiotemporal coordination in a modular robotic system. In: Nolfi S, Baldassarre G, Calabretta R, Hallam JCT, Marocco D, Meyer J-A, Miglino O, Parisi D (eds) *From Animals to Animats 9 (Proc. 9th Intl. Conf. Simulation of Adaptive Behavior – SAB2006)*, 25–29 September, Rome, Italy, Lecture Notes in Computer Science 4095, Springer-Verlag, Berlin: 558–569.
37. Pynadath DV, Tambe M (2002) Multiagent teamwork: analyzing the optimality and complexity of key theories and models. In: Castelfranchi C, Johnson WL (eds) *Proc. 1st Intl. Joint Conf. Autonomous Agents and Multiagent Systems – AAMAS2002*, 15–19 July, Bologna, Italy, ACM Press, New York, NY: 873–880.

38. Rasmussen S, Baas NA, Mayer B, Nilsson M, Olesen MW (2001) Ansatz for dynamical hierarchies. *Artificial Life*, 7(4): 329–353.
39. Rényi, A (1970) *Probability theory*. North-Holland, Amsterdam, The Netherlands.
40. Sandholm T, Lesser V (1995) Coalition formation among bounded rational agents. In: Mellish C (ed) *Proc. 14th Intl. Joint Conf. Artificial Intelligence – IJCAI95*, 20–25 August, Montréal, Québec, Canada, Morgan Kaufmann, San Francisco, CA: 662–671.
41. Sinai YG (1959) On the concept of entropy of a dynamical system (in Russian). *Doklady Akademii Nauk SSSR*, 124: 768–771.
42. Solé RV, Ferrer-Cancho R, Montoya JM, Valverde S (2002) Selection, tinkering and emergence in complex networks – crossing the land of tinkering. *Complexity*, 8(1): 20–33.
43. Takens F (1981) Detecting strange attractors in turbulence. *Dynamical systems and Turbulence*, Lecture Notes in Mathematics 898, Springer-Verlag, Berlin: 366–381.
44. US Department of Energy (2006) Benefits of Demand Response in Electricity Markets and Recommendations for Achieving Them. *A Report to the United States Congress Pursuant to Section 1252 of the Energy Policy Act of 2005*, February.
45. Wieselthier JE, Nguyen GD, Ephremides A (2000) On the construction of energy-efficient broadcast and multicast trees in wireless networks. *Proc. 19th Annual Joint Conf. IEEE Computer and Communications Societies – INFOCOM2000*, 26–30 March, Tel Aviv, Israel: 585–594.
46. White JG, Southgate E, Thompson JN, Brenner S (1986) The structure of the nervous system of the nematode *C. elegans*. *Philosophical Transactions of the Royal Society of London – Series B: Biological Sciences*, 314(1165): 1–340.
47. Williams, RJ, Martinez ND (2000) Simple rules yield complex food webs. *Nature*, 404: 180–183.
48. Ygge F, Akkermans JM (1996) Power load management as a computational market. In: Tokoro M (ed) *Proc. 2nd Intl. Conf. Multi-Agent Systems – ICMAS’96*, 9–13 December, Kyoto, Japan, AAAI Press, Menlo Park, CA: 393–400.
49. Zhang T, Ramakrishnan R, Livny M (1997) BIRCH: a new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2): 141–182.

Decentralised Clustering Algorithm

This Appendix reproduces the description of the clustering algorithm proposed by [26].

Each agent is initially a follower to itself, and its followers' list will contain only itself. Each agent is also a cluster-head initially (a singleton cluster). The communication messages (shown in italics) are 'flooding' broadcasts. The algorithm involves the following steps carried out by each cell (agent) which sensed the value x (henceforth, references like 'larger' or 'smaller' are relative to this value):

1. It keeps broadcasting its *recruit* message initially (*recruit* messages will always contain the followers' list of an agent). This broadcasting is done periodically, with a broadcasting-period P , affecting all agents with values within a particular offset of the value x of this agent – in other words, with values between $x - \varepsilon$ and $x + \varepsilon$. The offset ε is initially set to a proportion α of its agent value: $\varepsilon = \alpha x$;
2. If an agent in a singleton cluster receives a *recruit* message from a 'smaller' agent, it ignores it;
3. If an agent p in a singleton cluster receives a *recruit* message from a 'larger' agent q in a singleton cluster, it becomes its follower, stops broadcasting its own *recruit* messages and sends its information to its new cluster-head q : an *acceptance* message with its relative coordinates and the agent-value x . It also stores details of the cluster-head q : the agent-value x_q and relative coordinates;
4. If an agent p in a singleton cluster receives a *recruit* message from a 'larger' agent q which does have other followers, it ignores the message: simply because the 'larger' agent q would also receive and handle a *recruit* message from p itself (see step 6);
5. If an agent receives an *acceptance* message from some potential follower agent, it adds the agent involved in its followers' list;

6. If a member of a non-singleton cluster, either the head or a follower, receives a *recruit* message (either from a ‘larger’, ‘smaller’ or ‘equal’ agent), it *forwards* it to its present cluster-head;
7. After forwarding a *recruit* message to its cluster-head, a follower ignores further *recruit* messages until the identity of its head has been re-asserted (as a result of the clustering heuristic being invoked somewhere);
8. The cluster-head waits for a certain period W , collecting all such *forward* messages (the period W , called heuristic-period, should be greater than $2P$). At the end of the heuristic-period, the clustering heuristic is invoked by the cluster-head on the union set of followers and all agents who *forwarded* the messages. The ‘largest’ agent in any resulting cluster is appointed as its cluster-head;
9. The cluster-head which invoked the heuristic notifies new cluster-heads about their appointment, and sends their cluster maps to them: a *cluster-information* message;
10. A cluster-head stops sending its *recruit* messages P cycles before it invokes the clustering heuristic. If it is re-appointed as a cluster-head, it resumes sending *recruit* messages;
11. If an agent receives *cluster-information* message it becomes a cluster-head. If it was already a cluster-head with a cluster map, it erases that cluster map and accepts the new cluster map. It also *notifies* all its new followers;
12. A follower will periodically get *recruit* messages from its cluster-head. If this does not happen for a while, then it means that this follower is no longer in the followers’ list of its cluster-head. Then it will make itself a cluster-head and start sending its own *recruit* messages. The offset of these *recruit* messages will be determined by the offsets it had when it was a cluster-head the last time (not necessarily the same as ε).

Because of the unpredictable timing of the clustering heuristics being invoked in various agents, it is possible that a cluster-head keeps a particular agent as its follower even after its offset ε has changed and this particular agent is now out of range. To counter this, the cluster-head checks its followers’ list periodically and removes agents with values out of range. It is also possible that a node detects a new sensor reading, possibly increasing the agent-value by a large amount. If this agent was a follower, it immediately becomes a cluster-head and *updates* its former cluster-head. The former cluster-head will delete it from its followers’ list.

Depending on the nature of the set of agent values, the offset ε may be initially too small to reach any other agent. To counter this, an agent periodically (with a period δ) increases its offsets exponentially until a certain limit: $\varepsilon_{k+1} = \max(2\varepsilon_k, \beta x)$, where $\varepsilon_0 = \varepsilon$ initially, and β is the limit proportion (for example, the initial ε_0 may be $0.01x$ and after 5 periods the offset would become $\varepsilon_5 = 0.32x$). Alternatively, the increase will stop when the offsets of an agent have been reset by the clustering heuristic. When the clustering heuristic is applied, it may produce either one or two clusters as a

result. If there are two clusters, the offset of each new cluster-heads is modified. It is adjusted in such a way that the cluster-head of the 'smaller' agents can now reach up to, but not including, the 'smallest' agent in the cluster of 'larger' agents. Similarly, the cluster-head of 'larger' agents can now reach down to, but not including, the 'largest' agent (the cluster-head) of the cluster of 'smaller' agents. These adjusted offsets are sent to the new cluster-heads along with their cluster maps.

Predictor K_2

This Appendix describes the estimation method adopted by [34].

Suppose that the d -dimensional phase space is partitioned into boxes of size r^d . Let $P_{i_0, \dots, i_{d-1}}$ be the joint probability that a trajectory is in box i_0 at time 0, in box i_1 at time $\Delta t, \dots$, and in box i_{d-1} at time $(d-1)\Delta t$, where Δt is the time interval between measurements on the state of the system (in our case, we may assume $\Delta t = 1$, and omit the limit $\Delta t \rightarrow 0$ in the following definitions). The order-2 Rényi entropy K_2 is defined as

$$K_2 = \lim_{\Delta t \rightarrow 0} \lim_{r \rightarrow 0} \lim_{d \rightarrow \infty} \frac{1}{d \Delta t} \ln \sum_{i_0 \dots i_{d-1}} P_{i_0 \dots i_{d-1}}^2 \quad (1)$$

It is well-known that KS Entropy $K = 0$ in an ordered system, K is infinite in a random system, and K is a positive constant in a deterministic chaotic system. Grassberger and Procaccia (1983) considered the ‘correlation entropy’ K_2 in particular, and capitalised on the fact $K \geq K_2$ in establishing a sufficient condition for chaos $K_2 > 0$. The Grassberger and Procaccia algorithm estimates the entropy K_2 as follows

$$K_2 = \lim_{r \rightarrow 0} \lim_{d \rightarrow \infty} \lim_{N \rightarrow \infty} \ln \frac{C_d(N, r)}{C_{d+1}(N, r)} \quad (2)$$

where $C_d(N, r)$ is the correlation integral

$$C_d(N, r) = \frac{1}{N(N-1)} \sum_{i=1}^N \sum_{j=1}^N \theta(r - \|V_i - V_j\|) \quad (3)$$

Here θ is the Heaviside function (equal to 0 for negative argument and 1 otherwise), and the vectors V_i and V_j contain elements of the observed time series $\{v(t)\}$, ‘converting’ or ‘reconstructing’ the dynamical information in one-dimensional data to spatial information in the d -dimensional embedding space: $V_k = (v_k, v_{k+1}, v_{k+2}, \dots, v_{k+d-1})$ [28]. The norm $\|V_i - V_j\|$ is the

distance (sometimes called precision) between the vectors in the d -dimensional space, for example, the maximum norm

$$\|V_i - V_j\| = \max_{\tau=0}^{d-1} (\nu_{i+\tau} - \nu_{j+\tau}) \quad (4)$$

Put simply, $C_d(N, r)$ computes the fraction of pairs of vectors in the d -dimensional embedding space that are separated by a distance less than or equal to r . Since we consider only an initial segment of the times series, we simply set $N = \Omega$, estimating the entropy as

$$K_2(d, r, \Omega) = \ln \left(\frac{C_d(\Omega, r)}{C_{d+1}(\Omega, r)} \right) \quad (5)$$

Resources

1 Key Books

Barabási A-L (2002) *Linked: The New Science of Networks*. Perseus Publishing, Cambridge, MA

Bonabeau E (1999) *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY

Ferber J (1999) *Multi-agent systems*. Addison Wesley, Reading, MA

Rényi A (1970) *Probability Theory*. North-Holland, Amsterdam

2 Key Survey/Review Article

Solé RV, Ferrer-Cancho R, Montoya JM, Valverde S (2002) Selection, tinkering and emergence in complex networks – crossing the land of tinkering. *Complexity* 8(1): 20–33

3 Organisations, Societies, Special Interest Groups

Santa Fe Institute
<http://www.santafe.edu/index.php>

The New England Complex Systems Institute (NECSI)
<http://necsi.org/>

Institute for the Study of Complex Systems (ISCS)
<http://www.complexsystems.org/>

The Society for Modeling and Simulation International (SCS)

<http://www.scs.org/>

The ARC Complex Open Systems Research Network (COSNet)

<http://www.complexsystems.net.au/>

4 Research Groups

The ARC Centre of Excellence for Autonomous Systems, Sydney, Australia

<http://www.cas.edu.au/home.html>

The Adaptive Systems Research Group, University of Hertfordshire, UK

<http://homepages.feis.herts.ac.uk/~comqdp1/#asrg>

The Intelligent Interactive Distributed Systems group, Vrije Universiteit Amsterdam, The Netherlands

<http://www.iids.org/>

The Distributed Intelligence group, CSIRO ICT Centre, Sydney, Australia

<http://www.ict.csiro.au/page.php?cid=40>

Center for Complex Networks Research, The University of Notre Dame

<http://www.nd.edu/~networks/index.htm>

The Complex Systems Lab, The Universitat Pompeu Fabra

<http://complex.upf.es/~ricard/complexnets.html>

Networks and Chaos: Kaneko's Lab

<http://chaos.c.u-tokyo.ac.jp/>

5 Discussion Group, Forum

Complexity Digest

<http://www.comdig.org>

6 Key International Conferences/Workshops

International Conference on the Simulation of Adaptive Behavior (SAB)

European Conference on Artificial Life (ECAL)

Int. Conf. on the Simulation and Synthesis of Living Systems (ALife)

Intl. Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS)

Intl. Conf. Multisensor Fusion & Integration for Intelligent Systems (MFI)

Computational Intelligence in Agent-Based Computational Economics

Shu-Heng Chen

AI-ECON Research Center, Department of Economics, National Chengchi University, Taipei, Taiwan 116, chchen@nccu.edu.tw

1 Introduction

1.1 What is *Agent-Based Computational Economics* (ACE)?

Agent-based computational economics is the study of economics using *agent-based modeling and simulation*, which, according to [21], is the third way, in addition to deduction and induction, to undertake social sciences. An agent-based model is a model comprising *autonomous agents* placed in an interactive environment (society) or social network. Simulating this model via computers is probably the most practical way to visualize economic dynamics.

An autonomous agent is one which is able to behave (think, learn, adapt, make strategic plans) with a set of specifications and rules which are given initially; they are fixed and require no further intervention. The necessity for using autonomous agents in agent-based computational economics – or, more broadly, agent-based social sciences – is still an issue open for discussion. We make no attempt here to give a full account of the development of this issue – this would deserve a Chapter on its own. For a brief account, the use of autonomous agents is, in one way or the other, connected to the notion of *bounded rationality*, popularized by Herbert Simon [138].

In order to build autonomous agents, agent-based computational economists need to employ existing algorithms or develop new algorithms which can enable agents to behave with a degree of autonomy. Sections 1.2, 2 and 3 of the chapter will give a thorough review of the algorithmic foundations of ACE. We also introduce here the field known as *computational intelligence* (CI) and its relevance to economics. Section 4 then reviews the use of computational intelligence in agent-based economic and financial models. Section 5 gives some general remarks on these applications, as well as pointing to future directions. This is followed by concluding remarks in Sect. 6.

1.2 Algorithmic Foundations of ACE

The purpose of the next few Sections is to address a very important attribute of *autonomous agents*, this being their capability to adapt to a changing environment. The idea is to equip the artificial agents with some in-built algorithms so that they are able to develop some degree of sophisticated cognitive behavior; in particular, they are able to *learn from the past*, and hence are able to anticipate the future, and develop strategic behavior accordingly. The algorithms which can help our artificial agents achieve the above goal are initiated from many different fields and hence are interdisciplinary. Recently, they have been addressed together in the field known as *computational intelligence (CI)*. Therefore, the next few Sections may be read as an introduction to CI from the perspective of agent-based computational economics.

The aim of this Chapter is to review a number of important developments in computational intelligence, including artificial neural networks (Sect. 2) and evolutionary computation (Sect. 3). While these tools have been introduced to economists on numerous other occasions – for example, quantitative finance – we have a different motivation for studying them. Mainly, these two major CI tools allow us to discuss a number of crucial mental activities, such as attention control, memory, and pattern discovery. Therefore, even though our brief review will go through some important quantitative applications, we should remind readers at different places that our scope is broader.

Section 2 describes a number of different neural network models, which help us to understand how some algorithms, associated with the artificial brain, are able to conduct data compression, redundancy removal, classification, and forecasting. Let us be more specific with some of these. An important cognitive task for human agents is that, under some degree of survival pressure (or incentives), they are able to perform correct classification and react upon it properly. A simple example is the salesman who needs to identify those consumers who are willing to pay a high price for a specific new product, and to distinguish them from general buyers. A family of neural networks, also known as *supervised learning* (Sect. 2.1, 2.2, and 2.5), are able to equip agents with this capability.

Prior to classification, one more fundamental cognitive task is *concept formation* – in other words, to extract useful concepts from observations. Then, based on these concepts, new observations can be classified so as to facilitate decision-making. A typical example would be a stock trader who needs to recognize some special charts to make his market timing decisions. A family of neural networks, known as *unsupervised learning* (Sect. 2.6), can help agents to acquire this kind of cognitive capability.

Sometimes, it is hard to form concepts. In this case, one may directly deal with *cases*, and make decisions based on the similarity of cases. Sections 2.7 and 2.8 are devoted to the literature on *lazy learning* – that is, learning

by simply memorizing experiences, with little effort to develop generalized concepts on top of these experiences.

The third important cognitive task concerns the efficient use of limited brain space. This has something to do with data compression or redundancy removal. Section 2.4 introduces a network which can perform this task. In addition, Sect. 2.3 describes a device to reduce data storage space by building in loops in the ‘brain’.

The above three cognitive tasks do not involve social interaction. They mainly describe how an individual learns from his own experience without interacting with other individuals’ experiences. The latter case is referred to as *social learning* or *population learning* in the literature. Imitation (reproduction) is the clearest example of social learning: agents simply follow the behavior rules of whomever they consider the most suitable. Nonetheless, imitation is not enough to cover more complex patterns of social learning, such as innovations of using inspiration from others. Through evolutionary computation (Sect. 3), both forms (imitation and innovation) of learning with social interactions are operated with the familiar survival-of-the-fittest principle.¹ Genetic programming (GP) is a one kind of evolutionary computation. It differs from others in the sense that it gives us much more expressive power to observe changes.

2 Artificial Neural Networks

Among CI tools, the artificial neural network (ANN) is the most widely acceptable tool for economists and finance people, even though its history is much shorter than that of fuzzy logic so far as the application to economics and finance is concerned. The earliest application of ANNs was [156]. Since then we have witnessed an exponential growth in the number of applications. ANN is probably the only CI tool which drew serious econometricians’ attention and on which a lot of theoretical studies have been done. Both [131] and [157] gave a rigorous mathematical/statistical treatment of ANNs, and hence have established ANNs with a sound foundation in the econometrics field. Nowadays, ANNs have already become an integral part of textbooks in econometrics, and even moreso in financial econometrics and financial time-series. A great number of textbooks or volumes especially edited for economists and finance people are available, for example, [23, 24, 84, 130, 136, 147, 163], to name a few. Its significance to finance people can also be seen from the establishment of the *Neurove\$t* journal (now *Computational Intelligence in Finance*) in 1993.

It has been shown in a great number of studies that artificial neural nets, as representative of a more general class of nonlinear models, can outperform

¹ Evolutionary computation, in a sense, is a kind of ‘bio-sociology’.

many linear models, and can sometimes also outperform some other nonlinear models.²

Three classes of artificial neural nets have been most frequently used in economics and finance. These are *multilayer perceptrons*, *radial basis neural networks*, and *recurrent neural networks*. The first two classes will be introduced in Sects. 2.1 and 2.2, whereas the last one is introduced in 2.3.

2.1 Multilayer Perceptron Neural Networks

Let us consider the following general issue. We observe a time-series of an economic or financial variable, such as the foreign exchange rate, $\{x_t\}$. We are interested in knowing its future values, x_{t+1}, x_{t+2}, \dots . For that purpose, we need to search for a function relation $f(\cdot)$, such that when a vector \mathbf{x}_t is input into the function, a prediction on x_{t+1}, \dots can be made. The question then is how to construct such a function. Tools included in this Chapter provide two directions in which to work, distinguishable by different modeling philosophies. The first one is based on the *universal modeling approach*, and the second one is based on the *local modeling approach*. Alternatively, we can say that the first one aims to build the function in the *time domain*, whereas the second works in the *feature* or *trajectory domain*.³ We shall start with the first approach, and the canonical artificial neural network (ANN) can be considered to be a representative of this paradigm.

The reason why economists can embrace the ANN without any difficulty is due to the fact that ANN can be regarded as a generalization of their already familiar time-series model, *ARMA* (autoregressive moving-average) model. Formally, an ARMA(p, q) model is described as follows:

$$\Phi(L)x_t = \Theta(L)\epsilon_t \quad (1)$$

where $\Phi(L)$ and $\Theta(L)$ are polynomials of order p and q ,

$$\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \dots - \phi_p L^p \quad (2)$$

$$\Theta(L) = 1 - \theta_1 L - \theta_2 L^2 - \dots - \theta_q L^q \quad (3)$$

$\{\epsilon_t\}$ is white noise, and L is the lag operator.

ANNs can be regarded as a non-linear generalization of these ARMA processes. In fact, more concretely, *multilayer perceptron (MLP) neural networks* are nonlinear generalizations of the so-called autoregressive process,

$$x_t = f(x_{t-1}, \dots, x_{t-p}) + \epsilon_t \quad (4)$$

² This is not an appropriate place to provide a long list, but interested readers can find some examples from [7, 77, 90, 137, 153, 154] and [158].

³ There are alternate names for the local modeling approach, for example ‘*guarded experts*’ – see [18].

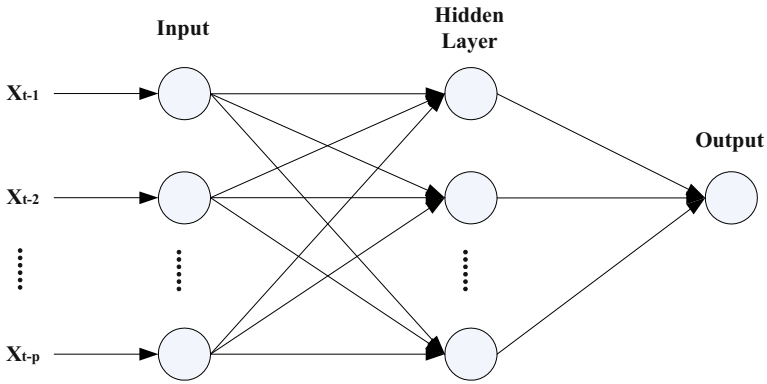


Fig. 1. The multilayer perceptron neural network of a non-linear AR process

whereas *recurrent neural networks* are non-linear generalizations of the ARMA processes,

$$x_t = f(x_{t-1}, \dots, x_{t-p}, \epsilon_{t-1}, \dots, \epsilon_{t-q}) + \epsilon_t \tag{5}$$

In terms of a multilayer perceptron neural network, Eqn. (4) can then be represented as:

$$x_t = h_2(w_0 + \sum_{j=1}^l w_j h_1(w_{0j} + \sum_{i=1}^p w_{ij} x_{t-i})) + \epsilon_t \tag{6}$$

Hence Eqn. (6) is a three-layer neural net (Fig. 1). The input layer has p inputs: x_{t-1}, \dots, x_{t-p} . The hidden layer has l hidden nodes, and there is a single output for the output layer \hat{x}_t . Layers are fully connected by *weights*; w_{ij} is the weight assigned to the i th input for the j th node in the hidden layer, whereas w_j is the weight assigned to the j th node (in the hidden layer) for the output; w_0 and w_{0j} are constants, also called *biases*; h_1 and h_2 are *transfer functions*.

There is a rich choice of transfer functions. According to [64], a multilayer perceptron network with any *Tauber-Wiener functions* as transfer function of the hidden units can be qualified as a *universal approximator*. Also, a necessary and sufficient condition for being a Tauber-Wiener function is that *it is non-polynomial*. In practice, a differentiable transfer function is desirable. Commonly used transfer functions for multilayer perceptron networks are the *sigmoid function*,

$$h_s(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

and *hyperbolic tangent function*,

$$h_t(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{8}$$

Clearly, $0 < h_s(x) < 1$, and $-1 < h_t(x) < 1$.

2.2 Radial Basis Network

Next to the multilayer perceptron neural network is the *radial basis network* (RBN), which is also popularly used in economics and finance. Radial basis function (RBF) networks are basically a feedforward neural networks with a *single hidden layer*,

$$f(x) = \sum_i^k w_i \varphi(\|x - c_i\|), \tag{9}$$

where $\varphi(\)$ is a *radial basis function*, c_i is the i th center, and k is the number of the center. Both w_i , c_i and k are determined by the data set of x . Typical choices of radial basis functions are:

- the *thin-plate-spline function*,

$$\varphi(x) = x^2 \log^x \tag{10}$$

- the *Gaussian function*,

$$\varphi(x) = \exp\left(-\frac{x^2}{\beta}\right) \tag{11}$$

- the *multi-quadratic function*,

$$\varphi(x) = (x^2 + \beta^2)^{\frac{1}{2}} \tag{12}$$

- the *inverse multi-quadratic function*,

$$\varphi(x) = \frac{1}{(x^2 + \beta^2)^{\frac{1}{2}}} \tag{13}$$

Theoretical investigation and practical results seem to show that the choice of radial basis function is not crucial to the performance of the RBF network.

It has been proved that the RBF network can indeed approximate arbitrarily well any continuous function if a sufficient number of radial-basis function units are given (the network structure is large enough), and the network parameters are carefully chosen. RBN also has the best approximation property in the sense of having the minimum distance from any given function under approximation.

2.3 Recurrent Neural Networks

In Sect. 2.1, we discussed the relation between time series models and artificial neural networks. Information transmission in the usual multilayer perceptron neural net is *feedforward* in the sense that information is transmitted *forward* from the input layer to the output layer, via all hidden layers in between, as shown in Fig. 1; transmission in the reverse direction between any two layers is not allowed.

This specific architecture makes the multilayer perceptron neural net unable to deal with the moving-average series, $MA(q)$, effectively. To see this, consider the following $MA(1)$ series:

$$x_t = \epsilon_t - \theta_1 \epsilon_{t-1} \tag{14}$$

It is well-known that if $|\theta_1| < 1$, then the above $MA(1)$ series can also be written as an $AR(\infty)$ series.

$$x_t = - \sum_{i=1}^{\infty} \theta_1^i x_{t-i} + \epsilon_t \tag{15}$$

In using the multilayer perceptron neural network to represent Eqn. (15), one needs to have an input layer with an infinite number of neurons (*infinite memory of the past*), namely, x_{t-1}, x_{t-2}, \dots , which is impossible in practice. Although from the viewpoint of approximation, an exact representation is not required and a compromise with a finite number of neurons (*finite memory*) is acceptable, in general quite a few inputs are still required. This inevitably increases the complexity of the network, leads to an unnecessary large number of parameters, and hence slows down the estimation and training process [116].

This explains why the multilayer perceptron neural net can only be regarded as a nonlinear extension of autoregressive (AR) time series models Eqn. (4), but not a nonlinear extension of autoregressive moving-average (ARMA) models Eqn. (16).

$$\begin{aligned} x_t &= f(x_{t-1}, \dots, x_{t-p}, \epsilon_{t-1}, \dots, \epsilon_{t-q}) + \epsilon_t \\ &= f(x_{t-1}, \dots, x_{t-p}, x_{t-p-1}, \dots) + \epsilon_t \end{aligned} \tag{16}$$

The finite memory problem of the multilayer perceptron neural net is well noticed by ANN researchers. In his celebrated article, Elman stated:

“...the question of how to represent time in connection models is very important. One approach is to represent time *implicitly* by its effects on processing rather than *explicitly* (as in a spatial representation)”. [76]: 179 (italics added)

The multilayer perceptron neural net tries to model time by giving it a spatial representation (that is, explicit) representation. What Elman suggests is to let time have an effect on the network response rather than represent it by an additional input dimension. Using an idea initiated by [95], Elman proposes an internal representation of memory by allowing the hidden unit patterns to be fed back to themselves. In this way, the network becomes *recurrent*.

The difference between the multilayer perceptron neural net (feed forward neural net) and the recurrent neural net can be shown as follows. For a multilayer perceptron neural network, Eqn. (4) can be re-formulated as Eqn. (6) (for a three-layer neural net – Fig. 1).

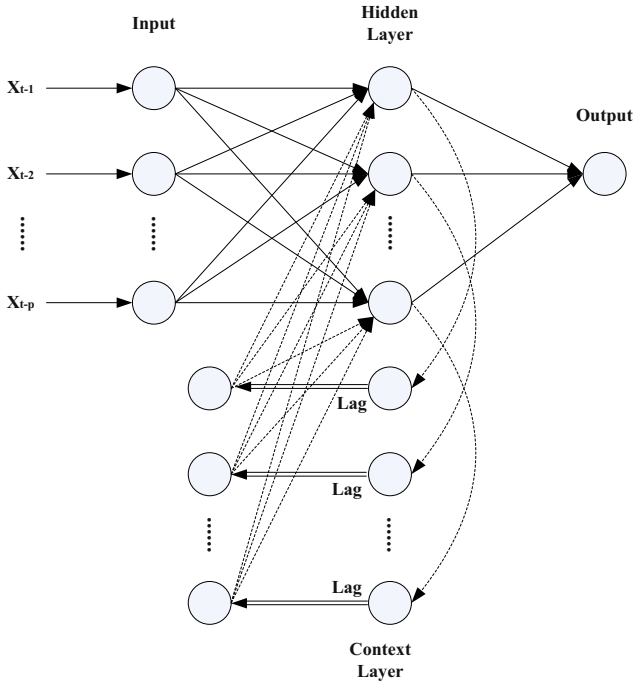


Fig. 2. The multilayer perceptron neural network model of a nonlinear AR process

A recurrent neural net – Eqn. (5) – can then be represented as:

$$x_t = h_2(w_0 + \sum_{j=1}^l w_j h_1(w_{0j} + \sum_{i=1}^p w_{ij} x_{t-i} + \sum_{m=1}^l \varpi_{mj} z_{m,t-1})) + \epsilon_t \quad (17)$$

where

$$z_{m,t} = w_{0m} + \sum_{i=1}^p w_{im} x_{t-i} + \sum_{k=1}^l \varpi_{kj} z_{k,t-1}, \quad m = 1, \dots, l \quad (18)$$

Compared to the multilayer perceptron and radial basis function neural nets, the recurrent neural net has been much less explored in the economic and financial domains.⁴ This is, indeed, a little surprising, considering the great exposure of its linear counterpart ARMA to economists.

2.4 Auto-Associative Neural Networks

While most economic and financial applications of neural networks consider the development of non-linear forecasting models, another important

⁴ Some early applications can be found in [36] and [108].

consideration is dimensionality reduction and/or feature extraction. In this application, ANN can provide a nonlinear generalization of the conventional *principal component analysis* (PCA). The specific kind of ANN for this application is referred to as the *auto-associative neural network* (AANN).

The fundamental idea of principal component analysis is dimensionality reduction, which is a quite general problem when we are presented with a large number of correlated attributes, and hence a large number of redundancies. It is, therefore, a natural attempt to compress or store this original large data set into a more economical space by getting rid of these redundancies. Thus, on the one hand, we want to have a reduced space that is as small as possible; on the other hand, we still want to keep the original information. These two objectives are, however, in conflict when attributes with complicated relations are presented. Therefore, techniques to make the least compromise between the two become important.

To introduce AANN and its relationship to principal component analysis, let us consider the following two mappings,

$$\mathcal{G} : \mathbf{R}^m \rightarrow \mathbf{R}^f \tag{19}$$

and

$$\mathcal{H} : \mathbf{R}^f \rightarrow \mathbf{R}^m \tag{20}$$

where \mathcal{G} and \mathcal{H} are, in general, nonlinear vector functions with their components indicated as $\mathcal{G} = \{G_1, G_2, \dots, G_f\}$ and $\mathcal{H} = \{H_1, H_2, \dots, H_m\}$. To represent these functions with multilayer perceptron neural nets, let us rewrite Eqn. (6) as follows,

$$\begin{aligned} y_k &= G_k(x_1, \dots, x_m) \\ &= h_2(w_{0k} + \sum_{j=1}^{l_1} w_{jk} h_1(w_{0j}^e + \sum_{i=1}^m w_{ij}^e x_i)), \quad k = 1, 2, \dots, f \end{aligned} \tag{21}$$

and

$$\begin{aligned} \hat{x}_i &= H_i(y_1, \dots, y_f) \\ &= h_4(w_{0i} + \sum_{j=1}^{l_2} w_{ji} h_3(w_{0j}^d + \sum_{k=1}^f w_{kj}^d y_k)), \quad i = 1, 2, \dots, m \end{aligned} \tag{22}$$

All the notations used in Eqns. (21) and (22) share the same interpretation as those in Eqn. (6), except that superscripts e and d stand for the encoding and decoding maps, respectively. By combining the two mappings, we have a mapping from $\mathbf{X} = \{x_1, \dots, x_m\}$ to its own reconstruction $\hat{\mathbf{X}} = \{\hat{x}_1, \dots, \hat{x}_m\}$. Let X_n be the n th observation of X , and

$$\mathbf{X}_n = \{x_{n,1}, \dots, x_{n,m}\} \tag{23}$$

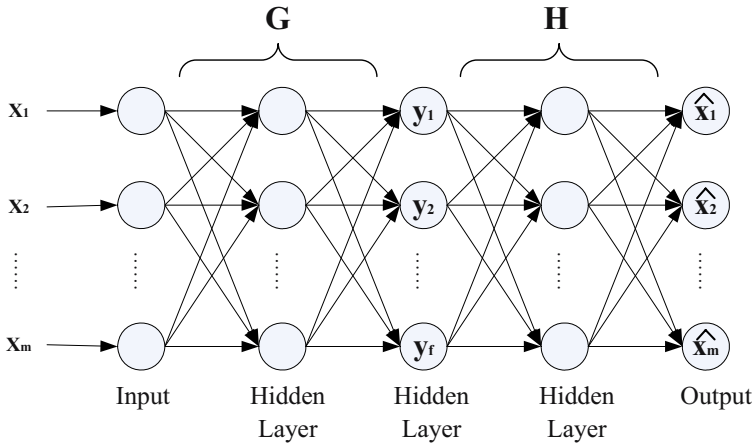


Fig. 3. The auto-associative neural network

Accordingly,

$$\hat{\mathbf{X}}_n = \{\hat{x}_{n,1}, \dots, \hat{x}_{n,m}\} \tag{24}$$

Then minimizing the difference between observation \mathbf{X}_n and its reconstruction $\hat{\mathbf{X}}_n$ over the entire set of N observations or

$$\min E = \sum_{n=1}^N \sum_{i=1}^m (x_{n,i} - \hat{x}_{n,i})^2 \tag{25}$$

by searching for the space of the connection weights and biases defines what is known as ‘auto-association neural networks’. Briefly, auto-associative neural networks are feedforward nets, with *three hidden layers*, as shown in Fig. 3, trained to produce an approximation of the *identity mapping* between network inputs and outputs using backpropagation or similar learning procedures.

The third hidden layer – namely the output layer of the MLPN, (Eqn. (21)) – is also called the *bottleneck layer*. If the transfer functions h_i ($i = 1, 2, 3, 4$) are all identical mappings, and we remove all the bias terms, then Eqn. (21) can be written as:

$$\begin{aligned} y_k &= G_k(x_1, \dots, x_m) \\ &= \sum_{j=1}^{l_1} w_{jk} \left(\sum_{i=1}^m w_{ij}^e x_i \right) = \sum_{j=1}^{l_1} \sum_{i=1}^m w_{jk} w_{ij}^e x_i, \\ &= \sum_{i=1}^m \sum_{j=1}^{l_1} w_{jk} w_{ij}^e x_i = \sum_{i=1}^m \beta_{i,k} x_i \quad k = 1, 2, \dots, f, \end{aligned} \tag{26}$$

where

$$\beta_{i,k} = \sum_{j=1}^{l_1} w_{jk} w_{ij}^e \tag{27}$$

In matrix notation, Eqn. (26) can be written as:

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix} \begin{bmatrix} \beta_{11} & \beta_{12} & \dots & \beta_{1f} \\ \beta_{21} & \beta_{22} & \dots & \beta_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{m1} & \beta_{m2} & \dots & \beta_{mf} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1f} \\ y_{21} & y_{22} & \dots & y_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \dots & y_{nf} \end{bmatrix}, \tag{28}$$

or simply

$$\mathbf{XB} = \mathbf{Y} \tag{29}$$

\mathbf{X} , \mathbf{B} and \mathbf{Y} correspond to the n -by- m , m -by- f , and n -by- f matrices in Eqn. (28), respectively. Likewise, Eqn. (22) can be simplified as:

$$\mathbf{YB}^* = \hat{\mathbf{X}} \tag{30}$$

\mathbf{B}^* is the reconstruction mapping and is an f -by- m matrix, and $\hat{\mathbf{X}}$ is the reconstruction of \mathbf{X} , and hence is an n -by- m matrix.

Equation (29) and (30), together with the objective function (Eqn. (25)), define the familiar *linear* principal component analysis. To see this, we can decompose \mathbf{X} as follows:

$$\mathbf{X} = \mathbf{YB}^* + \mathbf{E} = \mathbf{XBB}^* + \mathbf{E} = \mathbf{XP} + \mathbf{E} \tag{31}$$

where $\mathbf{P} = \mathbf{BB}^*$, and \mathbf{E} is the reconstruction error. Then the PCA frequently presented to us takes the form of the following minimization problem.

$$\min_{\mathbf{P}} \|\mathbf{E}\| \tag{32}$$

It is known that the optimal solution of this problem (Eqn. (32)) has the rows of \mathbf{P} being the eigenvectors corresponding to the f largest eigenvalues of the covariance matrix of \mathbf{X} . Therefore, we have shown how the self-associative neural network can be a nonlinear generalization of the familiar linear PCA, as well as how the linear PCA can be extended to the nonlinear PCA through a feedforward neural network with three hidden layers.

The concept of using a neural network with a bottleneck to concentrate information has been previously discussed in the context of *encoder/decoder* problems.⁵ [119] indicates some directions for financial applications using nonlinear PCA.

⁵ See [106] for a brief review.

2.5 Support Vector Machines

In the 1990s, based on results from *statistical learning theory* [149], an alternative to the artificial neural network was developed, in the form of the *support vector machine* (SVM). SVM was founded primarily by Vapnik, who contributed to the development of a general theory for minimizing the expected risk of losses using empirical data. Brief introductory material on the SVM can be found in [150], whereas [67] is a textbook devoted to the SVM.⁶

Support vector machines map non-linearly an n -dimensional input space into a high dimensional feature space.

$$\phi : V^n \rightarrow V^m \quad (33)$$

where V^n is an n -dimensional input vector space, and V^m is an m -dimensional feature vector space. Given a series of l historical observations:

$$(y_1, x_1), \dots, (y_l, x_l) \quad (34)$$

where $y_i \in V^1$ and $x_i \in V^n$.

We approximate and estimate the functional relation between y_i and x_i by

$$y = f(x) = \langle w, \phi(x) \rangle + b = \sum_{i=1}^m w_i \phi(x)_i + b \quad (35)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. The vector w and the constant b is to be determined by following the *structural risk minimization principle*, borrowed from statistical learning theory. It is interesting to note some similarities between the RBN and SVM, namely, Eqns. (9) and (35). However, there is also a noticeable difference. Consider an input x_i as a vector of three-dimensions: $(x_{i,1}, x_{i,2}, x_{i,3})$. Then for each neuron in the hidden layer of the RBN, they all share the same form as

$$(\varphi(x_{i,1}, x_{i,2}, x_{i,3}, c_1), \varphi(x_{i,1}, x_{i,2}, x_{i,3}, c_2), \dots) \quad (36)$$

while being associated with different centers. However, each neuron in the hidden layer of the SVM may actually take different inputs. For example, the first neuron takes the first two inputs, but the second takes the last two as

$$(\phi_1(x_{i,1}, x_{i,2}), \phi_2(x_{i,2}, x_{i,3}), \dots) \quad (37)$$

Also, notice that the transfer functions, $\varphi(\cdot)$ are the same for each neuron in the RBN, but in general are different for the SVM as ϕ_1, ϕ_2, \dots

⁶ Financial applications have kept on expanding; the interested reader can find some useful references directly from the SVM website: <http://www.svms.org/>

In the case where the y_i are categorical, such as $y_i \in \{-1, 1\}$, the minimization process also determines a subset of $\{x_i\}_{i=1}^l$, called *support vectors*, and the SVM when constructed has the following form.

$$f(x) = \sum_s y_i \alpha_i^* < \phi(x_s), \phi(x) > + b^* \tag{38}$$

where α_i^* and b^* are the coefficients satisfying the structural risk minimization principle, and s is the set of all support vectors.

The category assigned to the observation x , 1 or -1 , will then be determined by the sign of $f(x)$.

$$y = \begin{cases} 1, & \text{if } f(x) > 0 \\ -1, & \text{if } f(x) < 0 \end{cases} \tag{39}$$

Eqns. (38) and (39) are the SVM for the classification problem. A central concept of the SVM is that one does not need to consider the feature space in explicit form; instead, based on the Hilbert-Schmidt theory, one can use the *kernel function*, $K(x_s, x)$, where

$$K(x_s, x) = < \phi(x_s), \phi(x) > \tag{40}$$

Therefore, the SVM is also called the *kernel machine*. Eqn. (38) can then be rewritten as

$$f(x) = \sum_s y_i \alpha_i^* K(x_s, x) + b^* \tag{41}$$

Following a similar procedure, one can construct an SVM for regression problems as follow:

$$f(x) = \sum_{i=1}^l (\alpha_i^* - \beta_i^*) K(x, x_i) + b^* \tag{42}$$

where α_i^* , β_i^* and b^* are the coefficients minimizing the corresponding objective functions.

In addition to the functional form $f(\mathbf{x})$, the second important issue is the set of variables \mathbf{x} itself, and one has to deal naturally with the problem known as *variable selection* or *feature selection*. The involvement of irrelevant variables or features may lead to poor generalization capability.

2.6 Self-Organizing Maps and k-means

In the social and behavioral sciences, the ability to recognize patterns is an essential aspect of human heuristic intelligence. Herbert Simon, a Nobel Prize Laureate in Economics (1978), considered pattern recognition to be critical and advocated the need to pay much more explicit attention to the teaching

of pattern recognition principles. In the financial market, chartists appear to have been good at performing pattern recognition for many decades, yet little academic research has been devoted to a systematic study of these kinds of activities. On the contrary, sometimes it has been treated as nothing more than astrology, and hardly to be regarded as a rigorous science.

The Self-Organizing Map was invented by Kohonen [101]. It has been applied with great success to many different engineering problems and to many other technical fields. [71] was the first volume to demonstrate the use of the SOM in finance.

Self-organizing maps (SOMs) solve the pattern recognition problem which deals with a class of unsupervised neural networks. Basically, the SOM itself is a two-layer neural network. The input layer is composed of p cells, one for each system input variable. The output layer is composed of neurons which are placed on n -dimensional lattices (the value of n is usually 1 or 2). The SOM adopts so-called *competitive learning* among all neurons. Through competitive learning, the neurons are tuned to represent a group of input vectors in an organized manner.

k-means clustering, developed by [115], is a widely used *non-hierarchical clustering algorithm* that groups data with similar characteristics or features together. k-means and SOMs resemble each other. They both involve minimizing some measure of dissimilarity, called the cost function, in the samples within each cluster. The difference between the k-means and the SOM lies in their associated cost function to which we now turn. Consider a series of n observations, each of which has m numeric attributes:

$$\mathbf{X}_1^m, \mathbf{X}_2^m, \dots, \mathbf{X}_n^m, \quad \mathbf{X}_i^m \in \mathbf{R}^m \quad \forall i = 1, 2, \dots, n \tag{43}$$

where

$$\mathbf{X}_i^m \equiv \{x_{i,1}, x_{i,2}, \dots, x_{i,m}\}. \quad x_{i,l} \in \mathbf{R}, \forall l = 1, 2, \dots, m \tag{44}$$

The k-means clustering is to find a series of k clusters, the centroids of which are denoted, respectively, by

$$\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_k, \quad \mathbf{M}_j \in \mathbf{R}^m, \quad \forall j = 1, 2, \dots, k \tag{45}$$

such that each of the observations is assigned to one and only one of the clusters with minimal cost, and with cost function being defined as follows:

$$C_{k\text{-means}} = \sum_{i=1}^n \sum_{j=1}^k d(\mathbf{X}_i^m, \mathbf{M}_j) \cdot \delta_{i,j} \tag{46}$$

where $d(\mathbf{X}_i^m, \mathbf{M}_j)$ is the standard Euclidean distance between \mathbf{X}_i^m and \mathbf{M}_j ,⁷ and $\delta_{i,j}$ is the delta function:

⁷ Standard Euclidean distance assumes that the attributes are normalized and are of equal importance. However, this assumption may not hold in many application domains. In fact, one of the main problems in learning is to determine *which* are the important features.

$$\delta_{i,j} = \begin{cases} 1, & \text{if } \mathbf{X}_i^m \in \mathbf{Cluster}_j \\ 0, & \text{if } \mathbf{X}_i^m \notin \mathbf{Cluster}_j \end{cases} \quad (47)$$

To minimize the cost function (Eqn. (46)), one can begin by initializing a set of k cluster centroids. The positions of these centroids are then adjusted iteratively by first assigning the data samples to the nearest clusters and then re-computing the centroids.

Corresponding to Eqn. (46), the cost function associated with SOM can be roughly treated as follows⁸

$$C_{SOM} = \sum_{i=1}^n \sum_{j=1}^k d(\mathbf{X}_i^m, \mathbf{M}_j) \cdot h_w(\mathbf{x}_i^m),j \quad (48)$$

where $h_w(\mathbf{x}_i^m),j$ is the neighborhood function or neighborhood kernel, and $w_{\mathbf{X}_i^m}$ – the winner function – outputs the cluster whose centroid is nearest to the input \mathbf{X}_i^m .

In practice, the neighborhood kernel is chosen to be wide at the beginning of the learning process to guarantee global ordering of the map, and both its width and height decrease slowly during learning. For example, the Gaussian kernel whose variance monotonically decreases with iteration times t is frequently used.⁹ By comparing Eqn. (46) with Eqn. (48), one can see in SOM the distance of each input from all of the centroids weighted by the neighborhood kernel h , instead of just the closest one being taken into account.

Despite its greater simplicity, the economic and financial applications of k-means are surprisingly much less available than those of SOM and KNN. k-means have occasionally been applied to classify hedge funds [68], listed companies [128], and houses [93], but it can also be applied to the classification of trajectories of financial time series. To see this, we rewrite Eqns. (43) and (44) to fit the notations used in the context of time series:

$$\mathbf{X}_1^m, \mathbf{X}_2^m, \dots, \mathbf{X}_T^m, \quad \mathbf{X}_t^m \in \mathbf{R}^m, \quad \forall t = 1, 2, \dots, T \quad (49)$$

$$\mathbf{X}_t^m \equiv \{x_t, x_{t-1}, \dots, x_{t-m}\}, \quad x_{t-l} \in \mathbf{R}, \forall l = 0, 1, \dots, m - 1 \quad (50)$$

\mathbf{X}_t^m is a windowed series with an immediate past of m observations, also called the m -history. Eqn. (49), therefore, represents a sequence of T m -histories which are derived from the original time series, $\{x_t\}_{t=-m+1}^T$, by moving the m -long window consecutively, one step at a time. Accordingly, the end-product of applying k-means or SOMs to these windowed series is a number of centroids \mathbf{M}_j , which represents a specific shape of an m -long trajectory, also known as ‘charts’ by technical analysts.¹⁰

⁸ The rigorous mathematical treatment of the SOM algorithm is extremely difficult in general – see [102].

⁹ For details, see [51] Chap. 8: 205.

¹⁰ For example, see the charts presented in [44]: 206–207.

Then the essential question pursued here is whether we can meaningfully cluster the windowed financial time series \mathbf{X}_t^m by the k associated geometrical trajectories, $\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_k$. The clustering work can be meaningful if it can help us predict the future. In other words, conditional on a specific trajectory, we can predict the future better than without being provided this information, for example,

$$Prob(|\xi_{t+1}| > |\epsilon_{t+1}|) > 0.5 \tag{51}$$

where

$$\xi_{t+1} = x_{t+1} - E(x_{t+1}) \tag{52}$$

and

$$\epsilon_{t+1} = x_{t+1} - E(x_{t+1} | \mathbf{X}_t^m \in \text{Cluster}_j), \quad t > T \tag{53}$$

The conditional expectations above are made with the information of the trajectory (the cluster).

2.7 K Nearest Neighbors

In 1998, a time-series prediction competition was held during the *Intl. Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*. The data to be predicted were available from November 1997 through April 1998 at Leuven. The data was generated from a generalized Chua’s circuit, a well-known chaotic dynamic system. Seventeen entries had been submitted before the deadline. The winner of the competition turned out to be James McNames, and the strategy he used was the *nearest trajectory algorithm*. By using this algorithm to fast nearest neighbor algorithms, McNames was able to make an accurate prediction up to 300 points in the future of the chaotic time-series. At first sight, this result may be a surprise for some, because KNN is not technically demanding in contrast to many other well known tools as introduced in this Chapter, nevertheless it could outperform many other familiar advanced techniques, such as neural nets, wavelets, Kohonen maps (SOM), and Kalman filters in that competition.¹¹

KNN can be related to *decision trees*. What makes them different is that the latter have categories A_1, \dots, A_n to host input variables \mathbf{X}_t^m , while the former have \mathbf{X}_t^m itself as a center of a hosting category, which will invite its own *neighbors*, \mathbf{X}_s^m ($s < t$), by ranking the *distance* $\|\mathbf{X}_t^m - \mathbf{X}_s^m\|$ over all $s < t$ from the closest to the farthest. Then the k closest \mathbf{X}_s^m s will constitute the neighbors of \mathbf{X}_t^m , $\mathcal{N}(\mathbf{X}_t^m)$. Now, for the purpose of predicting x_{t+1} , one can first study the functional relation between x_{s+1} and \mathbf{X}_s^m , $\forall s \in \mathcal{N}(\mathbf{X}_t^m)$, in other words,

$$x_{s+1} = f_t(\mathbf{X}_s^m), s \in \mathcal{N}(\mathbf{X}_t^m) \tag{54}$$

¹¹ For details of the competition report, see [140].

One then forecasts x_{t+1} based on \hat{f}_t , an estimation of f_t ,

$$\hat{x}_{t+1} = \hat{f}_t(\mathbf{X}_t^m) \tag{55}$$

Let's make a brief remark on what makes KNN different from conventional time-series modeling techniques. Conventional time-series modeling, known as the Box-Jenkins approach, is a *global* model, which is concerned with the estimation of the function, be it linear or non-linear, in the following form:

$$x_{t+1} = f(x_t, x_{t-1}, \dots, x_{t-m}) + \epsilon_t = f(\mathbf{X}_t^m) + \epsilon_t \tag{56}$$

by using all of the information up to t – that is, $\mathbf{X}_s^m \forall s \leq t$ – and the estimated function \hat{f} is assumed to hold for every single point in time. As a result, what will affect x_{t+1} most is its immediate past x_t, x_{t-1}, \dots under the law of motion estimated by all available samples.

For KNN, while what affects x_{t+1} most is also its immediate past, the law of motion is estimated *only* with *similar* samples, *not all* samples. The estimated function \hat{f}_t is hence assumed to only hold for that specific point in time. Both KNN and SOM challenge the conventional Box-Jenkins methodology by characterizing the hidden patterns in a different form. In their formulation, hidden patterns are not characterized by time location, but by topological trajectories.

Technical issues involved here are the choice of distance function $d(\mathbf{X}_t^m, \mathbf{X}_s^m)$, choice of functional form f_t , choice of the number of neighbors k , and choice of the embedding dimension m .

2.8 Instance-Based Learning

KNN can be regarded as a special case of a broader class of algorithms, known as *instance-based learning* (IBL). To see this, let us use the notations introduced in Sect. 2.6, and use the time series prediction problem as an illustration.

Consider Eqn. (53). We have been given information regarding a time series up to time t , and we wish to forecast the next by using the current m -history, \mathbf{X}_t^m . In SOM or KNN, we will first decide which cluster \mathbf{X}_t^m belongs by checking $d(\mathbf{X}_t^m, \mathbf{M}_j)$ for all j ($j = 1, 2, \dots, k$), and use the forecast model associated with that cluster to forecast x_{t+1} . In other words, forecasting models are tailored to each cluster, say, \hat{f}_j ($j = 1, 2, \dots, k$).¹² Then

$$\hat{x}_{t+1} = \hat{f}_{j^*}(\mathbf{X}_t^m), \text{ if } j^* = \arg \min_j d(\mathbf{X}_t^m, \mathbf{M}_j) \quad j = 1, 2, \dots, k \tag{57}$$

¹² The notation \hat{f} is used, instead of f , to reserve f for the true relation, if it exists, and in that case, \hat{f} is the estimation of f . In addition, there are variations when constructing Eqn. (57) – see [44].

KNN, however, does not have such established clusters \mathbf{M}_j . Instead, it forms a cluster based on each $\mathbf{X}_t^m, \mathcal{N}(\mathbf{X}_t^m)$, as follows.

$$\mathcal{N}(\mathbf{X}_t^m) = \{s \mid \text{Rank}(d(\mathbf{X}_t^m, \mathbf{X}_s^m)) \leq k, \forall s < t\} \tag{58}$$

In other words, \mathbf{X}_t^m itself serves as the centroid of a cluster, called the *neighborhood* of $\mathbf{X}_t^m, \mathcal{N}(\mathbf{X}_t^m)$. It then invites its k nearest neighbors to be the members of $\mathcal{N}(\mathbf{X}_t^m)$ by ranking the distance $d(\mathbf{X}_t^m, \mathbf{X}_s^m)$ over the entire community

$$\{\mathbf{X}_s^m \mid s < t\} \tag{59}$$

from the closest to the farthest.

Then, by assuming a functional relation, f , between x_{s+1} and \mathbf{X}_s^m and using only the observations associated with $\mathcal{N}(\mathbf{X}_t^m)$ to estimate this function f_t ,¹³ one can construct the tailor-made forecast for each x_t ,

$$\hat{x}_{t+1} = \hat{f}_t(\mathbf{X}_t^m) \tag{60}$$

In practice, the function f used in Eqn. (60) can be very simple, either taking the *unconditional mean* or the *conditional mean*. In the case of the latter, the mean is usually assumed to be linear. In the case of the unconditional mean, one can simply use the simple average in the forecast,

$$\hat{x}_{t+1} = \frac{\sum_{s \in \mathcal{N}(\mathbf{X}_t^m)} x_{s+1}}{k} \tag{61}$$

but one can also take the weighted average based on the distance of each member.

The same idea can be applied to deal with the linear conditional mean (linear regression model): we can either take the ordinal least squares or the weighted least squares.¹⁴

From the above description, we find that KNN is different from k-means and SOM in the sense that, not just the forecasting function, but also the cluster for KNN is tailor-made. This style of tailor-made learning is known as *lazy learning* in the literature [2]. It is called ‘lazy’ because learning takes place when the time comes to classify a new instance, say \mathbf{X}_{T+t}^m , rather than when the *training set*, Eqn. (49), is processed, say T .¹⁵

¹³ Even though the functional form is the same, the coefficients can vary depending on \mathbf{X}_t^m and its resultant $\mathcal{N}(\mathbf{X}_t^m)$. Accordingly, we add a subscript t as f_t to make this time-variant property clear.

¹⁴ Details can be found in [78].

¹⁵ Note that a fixed T in Eqn. (49) implies a fixed training set without increments. A non-incremental training set can be typical for using k-means or SOM. However, KNN learning, also known as *rote learning*, memorizes everything that happens up to the present; therefore, the ‘training set’ (memory) for KNN grows with time.

To make this clear, consider two types of agents: the k-means agent and the KNN agent. The k-means agent learns from the history before new instances come, and the resultant knowledge from learning is represented by a set of clusters, which is *extracted* from a set of historical instances. Based on these clusters, some *generalization pictures* are already produced before the advent of new instances, say \mathbf{X}_{T+t}^m . The KNN agent, however, is not eager to learn. While he does store every instance observed, he never tries to extract knowledge (general rules) from them. In other words, he has the simplest form of ‘learning’, that is, rote learning (plain memorization). When the time $T + t$ comes and a new instance \mathbf{X}_{T+t}^m is encountered, his memory is then searched for the historical instances that most strongly resemble \mathbf{X}_{T+t}^m .

As stated previously, KNN, as a style of rote learning, stores all the historical instances, as shown in Eqn. (59). Therefore, amounts of storage increase with time. This may make the nearest-neighbor calculation unbearably slow. In addition, some instances may be regarded as redundant with regard to the information gained. This can be particularly the case when KNN is applied to *classification* rather than regression or time series forecasting. For example, if we are interested not in x_{t+1} itself, but in whether x_{t+1} will be greater than x_t – namely, whether x_t will go up or go down, then some regions of the instance space may be very stable with regard to class – for instance, up (1) or down (0) – and just a few exemplars are needed inside stable regions. In other words, we do not have to keep all historical instances or training instances. The *storage-reduction algorithm* is then used to decide which instances in Eqn. (59) to save and which to discard. This KNN with the storage-reduction algorithm is called *instance-based learning* (IBL) and was initiated by [3].¹⁶

The addition of a storage-reduction algorithm to KNN is also interesting from the perspectives of both neural sciences and economics. Considering the brain with its limited capacity for memory, then an essential question to ask is how the brain deals with increasing information by not memorizing all of it or by forgetting some of it. How does it perform pruning? This is still a non-trivial issue pursued by neural scientists today. The same issue can interest economists as well, because it concerns the efficient use of limited space. A recent study on reward-motivated memory formation by neural scientists may provide an economic foundation for the memory formation [1].¹⁷

In this vein, the *marginal productivity* of the new instance in IBL can be considered as the reward. The marginal productivity of an instance can

¹⁶ As a matter of fact, the storage-reduction algorithms are not just to deal with the *redundancy* issue, but also the *noise-tolerance* issue. [3] distinguish the two by calling the former *memory updating functions*, and the latter *noise-tolerant algorithms*.

¹⁷ [1] report brain-scanning studies in humans that reveal how specific *reward-related brain regions* trigger the brain’s learning and memory regions to promote memory formation.

be defined by its contribution to enhance the capability to perform a correct classification. For those instances which have low marginal productivity, it will be discarded (not remembered), and for those already stored instances, if their classification performances are poor, they will be discarded, too (forgotten). In this way, one can interpret the mechanism of the pruning algorithms or the storage-reduction algorithms used in computational intelligence in the fashion of neural economics.

3 Evolutionary Computation

The second important pillar of computational intelligence is so called *evolutionary computation* (EC). EC uses *Nature* as an inspiration. While it also has a long history of utilization in economics and finance, it is, relatively speaking, the ‘new kid on the block’, as compared with neural networks, and even more so with fuzzy logic. It has also drawn less attention from economists and financial analysts than the other two approaches. By comparison, there are already about a dozen books or volumes on the economic and financial applications using fuzzy logic and neural nets. In the area of EC, there are only three volumes edited for economists and financiers [25, 39, 40]. Evolutionary computation is generally considered to be a consortium of *genetic algorithms* (GA), *genetic programming* (GP), *evolutionary programming* (EP) and *evolutionary strategies* (ES).

The history of evolutionary computation can be traced back to the mid-1960s, where evolutionary strategies were originated by Rechenberg [129], Schwefel [134] and Bienert at the Technical University of Berlin. The development of genetic algorithms started with Holland at the University of Michigan, and evolutionary programming was originated by Fogel [80] at the University of California at Los Angeles.¹⁸ Despite their non-trivial differences, they share the common structure shown in Fig. 4.

Evolutionary computation starts with an initialization of a population of individuals (solution candidates), called $P(0)$, with a *population size* to be supplied by the users. These solutions will then be evaluated based on an *objective function* or a *fitness function* determined by the problem of interest. Continuation of the procedure will hinge on the *termination criteria* supplied by the users. If these criteria are not met, then we move to the next stage or *generation* by adding 1 to the time counter ($t \rightarrow t + 1$). Two major operators are conducted to form the new generation, which can be regarded as a *correspondence*, as follows,

$$F_{s_2} \circ F_a \circ F_{s_1}(P(t)) = P((t + 1)) \quad (62)$$

where F_{s_1} and F_{s_2} denotes *selection*, and F_a denotes *alteration*.

¹⁸ For a description of the birth of EC, see [75], [79], and [135].

```

begin
  t := 0;
  Initialize P(t);
  evaluate P(t);
  while not terminating do
    begin
      M(t) := select-mates(P(t));
      O(t) := alternation(M(t));
      evaluate(O(t));
      P(t+1) := select(O(t) ∪ P(t));
      t := t+1;
    end
  end

```

Fig. 4. Evolutionary computation (EC) pseudo-algorithm

The main purpose of the first-stage selection, F_{s_1} , is to form a mating pool (a collection of parents), $M(t)$, which can in turn be used to breed the new generation:

$$F_{s_1}(P(t)) = M(t). \quad (63)$$

Once the mating pool is formed, F_a is applied to generate *offspring*, $O(t)$, from these parents. Two major steps (*genetic operators*) are involved here, namely, *recombination* (*crossover*), denoted by F_r , and *mutation*, denoted by F_m , which shall be described in detail later.

$$F_a(M(t)) = F_m \circ F_r(M(t)) = O(t). \quad (64)$$

These offspring will be first evaluated, then enter the second-stage selection *with* or *without* their parents $P(t)$. Finally, the new generation $P(t + 1)$ is formed as a result of the second-stage selection.

$$F_{s_2}(O(t) \cup P(t)) = P((t + 1)). \quad (65)$$

After that, we go back to the beginning of the loop, and then check the termination criteria to see whether to stop or to start another generation of runs – see Fig. 5 for the evolution loop.

Based on the description above, it is perhaps beneficial to have the seven major components of evolutionary algorithms listed as follows for quick reference:

1. individuals and their representations,
2. initialization,
3. fitness evaluation,
4. selection,

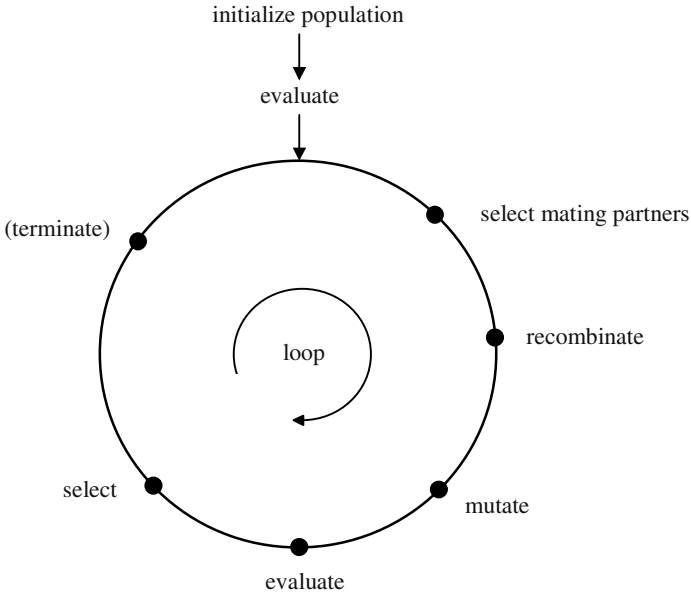


Fig. 5. The evolutionary loop

- 5. mutation,
- 6. recombination,
- 7. replacement.

3.1 Evolutionary Strategies

We shall illustrate each of these components mainly within the context of *evolutionary strategies*. Individuals are also called *chromosomes*. The individual in ES is represented as a pair of real-valued vectors $v = (x, \sigma)$, where the x represent a point in the solution space, and σ is a standard deviation vector that determines the mutation step size. Generally, σ is also called the *strategy parameter* in ES, and x is called the *object variable*.

The ES population size of is usually characterized by two parameters μ and λ . The former is the population size of $P(t)$, whereas the later is the population size of $O(t)$. Selection of F_{s_1} is much more straightforward in ES than in GA. Usually, it takes the whole $P(t)$ as the mating pool and parents are randomly selected therein. However, selection of F_{s_2} in ES can be more intriguing. There are two F_{s_2} schemes in ES, known as the $(\mu + \lambda)$ (Plus) scheme and the (μ, λ) (Comma) scheme. In the $(\mu + \lambda)$ scheme, μ individuals produce λ offspring, and a new population is formed by selecting μ individuals from $\mu + \lambda$. In the (μ, λ) scheme, μ individuals produce λ offspring, and a new population is formed by selecting μ individuals from the λ offspring. There is generally no constraint for μ and λ for the $(\mu + \lambda)$ scheme, but for the

(μ, λ) scheme, to make selection meaningful, μ has to be strictly less than λ ; moreover, $\lambda/\mu \approx 7$ is an ideal ratio.

Mutation is considered the major ES operator for altering chromosomes. Mutation is applied to this individual to perturb real-valued parameters. If we let v be the parent randomly selected from $P(t)$, then mutation on v can be described as follows:

$$v' = (x', \sigma') = (f_{m_x}(x), f_{m_\sigma}(\sigma)) \tag{66}$$

where

$$f_{m_x}(x) = x + N(0, (\sigma')^2) \tag{67}$$

and

$$f_{m_\sigma}(\sigma) = \sigma \exp(\tau N(0, 1)) \tag{68}$$

$N(0, \sigma^2)$ denotes the normal distribution with mean 0 and variance σ^2 .¹⁹ Notice that in implementation, Eqn. (68) has to be computed before Eqn. (67). This is because x' is obtained by mutating x with the new standard deviation σ' .²⁰

Recombination operators compose new chromosomes from corresponding parts of two or more chromosomes. For the binary case, two chromosomes $v_1 = (x_1, \sigma_1^2)$ and $v_2 = (x_2, \sigma_2^2)$ are to be recombined by an operator f_r . We can describe the composition of a new chromosome v' as follows:

$$v' = (x', \sigma') = (f_{r_x}(x_1, x_2), f_{r_\sigma}(\sigma_1^2, \sigma_2^2,)) \tag{69}$$

Each element of the object and strategy parameter is a recombination of the respective entries v_1 and v_2 . There is great variation of f_{r_x} and f_{r_σ} . In the ES literature, they are differentiated by the terms *discrete* or *intermediate*, *dual* (sexual) or *global* (panmictic). With a *discrete* recombination function, one of the corresponding components is chosen at random and declared the new entry. With an intermediate recombination, a linear combination of the corresponding components is declared the new entry. More formally, consider x' as an example:

$$x' = \begin{cases} x_1 \text{ or } x_2 & \textit{discrete} \\ \chi x_1 + (1 - \chi)x_2 & \textit{intermediate} \end{cases} \tag{70}$$

where $\chi \in [0, 1]$ denotes a uniform random variable.

¹⁹ Here, for simplicity, we assume that x is a real-valued number. In a more general setting, the variable x can be a vector; in that case, σ should be replaced by the variance-covariance matrix Σ .

²⁰ In Eqn. (67), $(\sigma')^2$ is determined randomly. There is, however, some way to make it adaptive. For example, in the (1 + 1)-ES case, one has the famous 1/5-success rule. $(\sigma')^2$ can also be determined in a self-adaptive way. In that case, the learning rate τ can be set as a function of time. For details, see [135].

So far we have only considered a one-dimensional x . An n -dimensional x can further complicate the recombination function, and that is where the terms *dual* and *global* come from. *Dual* means that two parents are chosen at random for the creation of the offspring. *Global* means that one parent is chosen anew for *each component* of the offspring.

$$x'_i = \begin{cases} x_{1,i} & \text{or } x_{2,i} & \text{discrete, dual} \\ x_{1,i} & \text{or } x_{(2),i} & \text{discrete, global} \\ \chi x_{1,i} + (1 - \chi)x_{2,i} & & \text{intermediate, dual} \\ \chi x_{1,i} + (1 - \chi)x_{(2),i} & & \text{intermediate, global} \end{cases} \quad (71)$$

where $x_{(2),i}$ indicates that parent 2 is chosen anew for each vector component i , ($i = 1, 2, \dots, n$).

3.2 Evolutionary Programming

While evolutionary programming (EP) was proposed about the same time as evolutionary algorithms, their initial motives were quite different. Evolutionary strategies were developed as a method to solve *parametric optimization problems*, whereas evolutionary programming was developed as a method to simulate *intelligent behavior*. Lacking a capability to predict, an agent cannot adapt its behavior to meet the desired goals, and success in predicting an environment is a prerequisite for intelligent behavior. As Fogel puts it:

“Intelligent behavior is a composite ability to predict one’s environment coupled with a translation of each prediction into a suitable response in the light of some objective”. ([82]: 11)

During the early stage, the prediction experiment can be illustrated with a sequence of symbols taken from a finite alphabet, say, a repeating sequence ‘(101110011101)*’ from the alphabet $\{0, 1\}$. The task then is to create an algorithm that would operate on the observed indexed set of symbols and produce an output symbol that agrees with the next symbol to emerge from the environment. Fogel took *finite state automata* (FSA) as the machine to predict the sequence. A FSA is a device which begins in one state and upon receiving an input symbol, changes to another state according to its current state and the input symbol. EP was first proposed to evolve a population of finite state machines that provides successively better predictions.

3.3 Genetic Programming and Genetic Algorithms

While genetic programming has been applied to economic modeling for more than half a decade, its relevance to the nature of economics has not been fully acknowledged. In the most sympathetic situations, it is regarded as nothing

but *alchemy*. In unsympathetic situations, it is notorious for its *black-box* operation. Sometimes, the process and results are so complicated that economists can hardly consider it relevant and interesting. This Section is intended to deliver a simple but strong message: *genetic programming is not just another fancy technique exploited by the unorthodox, but could be a faithful language to express the essence of economics*. In particular, it provides evolutionary economists with a way to *substantiate* some features which distinguish them from mainstream economists.

An Evolving Population of Decision Rules

Let's start from the most fundamental issue: *why is genetic programming relevant?* Lucas provided a notion of an *economic agent*.

“In general terms, we view or model an individual as *a collection of decision rules* (rules that dictate the action to be taken in given situations) and *a set of preferences* used to evaluate the outcomes arising from particular situation-action combinations”. [114]: 217 (italics added)

Immediately after this *static description* of an economic agent, Lucas described an *adaptive (evolutionary)* version:

“These decision rules are continuously under review and revision: new decision rules are tried and tested against experience, and rules that produce desirable outcomes supplant those that do not”. (*Ibid*: 217).

So, according to Lucas, the essence of an economic agent is *a collection of decision rules which are adapting (evolving) based on a set of preferences*. In short, it is the idea of an ‘evolving population’.

Suppose that an evolving population is the essence of the economic agent, then it seems important to know whether we economists know any operational procedure to substantiate this essence. Back in 1986, the answer was absolutely ‘no’. That certainly does not mean that we did not know anything about evolving *decision rules*. On the contrary, since the late 1970s, the literature known as ‘bounded rationality in macroeconomics’ has introduced a number of techniques to evolve a single decision rule (a single equation or a single system of equations): recursive regression, Kalman filtering, and Bayesian updating, to name a few; [132] made an extensive survey of this subject. However, these techniques shed little light on how to build a Lucasian agent, especially since what we wanted to evolve was not a single decision rule but a population of decision rules.

In fact, it may sound a little surprising that economists in those days rarely considered an individual as a population of decision rules, not to mention attending to the details of its evolution. Therefore, all the basic issues pertaining to models of the evolving population received little, if any, attention.

For example, how does the agent *initialize* a population of decision rules? Once the agent has a population of decision rules, which one should they follow? Furthermore, in what ways should this population of decision rules ‘be continuously under review and revision’? Should we review and revise them one by one because they are independent, or modify them together because they may correlate with each other? Moreover, if there are some ‘new decision rules to be tried’, how do we generate (or find) these new rules? What are the relations between these new rules and the old ones? Finally, it is also not clear how ‘rules that produce desirable outcomes should supplant those that do not.’

There is one way to explain why economists are not interested in, and hence not good at, dealing with a population of decision rules: economists used to derive the decision rule for the agent *deductively*, and the deductive approach usually leads to only one solution (decision rule), which is the *optimal* one. There was simply no need for a population of decision rules.

Genetic Algorithms and Classifier Systems

We do not know exactly when or how the idea of the *evolving population of decision rules* began to attract economists, but Holland’s contribution to *genetic algorithms* definitely exerted a great influence. Genetic algorithms simulate the biological evolution of a society of computer programs, each of which is represented by a chromosome or, normally, a string of binary ones and zeros. Each of these computer programs can be matched to a solution to a problem. This structure provides us with an operational procedure of the Lucasian agent. First, a collection of decision rules are now represented by a society of computer programs (a society of strings of binary ones and zeros). Second, the review and revision process is implemented as a process of natural selection.

While genetic algorithms have had a great impact on computer science, mathematics, and engineering since the early 1980s, their implications for social sciences were not acknowledged until the late 1980s. In 1987, Axelrod, a political scientist at the University of Michigan, published the first application of the GA to the social sciences [21]. A year later, the first PhD dissertation that applied GAs to the social sciences was completed by John Miller from, not surprisingly, the University of Michigan. The issue addressed by Axelrod and Miller is the well-known *repeated prisoner’s dilemma*. In addition to these two early publications, perhaps the most notable event that brought GAs into economics was the invited speech by John Holland at an economic conference at the Santa Fe Institute in the autumn of 1987. Among the audience were some of the most prestigious contemporary economists, including Kenneth Arrow, Thomas Sargent, Hollis Chenery, Jose Scheinkman, and Brian Arthur. In his lecture entitled ‘The global economy as an adaptive process’,

Holland introduced to the economics circle the essence of genetic algorithms as ‘building blocks’.

A building block refers to the specific pattern of a chromosome – that is, an essential characteristic of a decision rule. There is a formal word for this in the genetic algorithm; it is called a *schema*. In the genetic algorithm, a schema is regarded as the basic unit of learning, evolution, and adaptation. Each decision rule can be defined as a combination of some schemata. The review and revision process of decision rules is nothing more than a search for the right combination of those, possibly infinite, schemata. To rephrase Lucas’s description in Holland’s words, “economic agents are constantly revising and rearranging their building blocks as they gain experience”. Not only do genetic algorithms make the Lucasian economic agent implementable, but they also enrich its details.

After a gradual spread and accumulation of knowledge about GA among economists, modeling economic agents with an evolving population of decision rules finally began to increase in the 1990s. To the best of this author’s knowledge, the first refereed journal article was [117]. This paper is follow-up research to that of [100]. In a simple barter economy, Kiyotaki and Wright found that low storage costs are not the *only* reason why individuals use money. The other one is that money makes it easier to find a suitable partner. Replacing the rational agents in the Kiyotaki-Wright environment with *artificially intelligent* agents, [117], however, found that goods with low storage costs play the dominating role as a medium of exchange.

The population of decision rules used to model each agent is a *classifier system*, another contribution made by Holland in the late 1970s. A classifier system is similar to the Newell-Simon type expert system, which is a population of *if..then* or *condition-action* rules. However, the classical expert system is not adaptive. What Holland did with the classifier system was to apply the idea of competition in the market economy to a society of *if..then* rules. Market-like competition is implemented by way of a formal algorithm known as the *bucket-brigade algorithm*, credit rules generating good outcomes and debit rules generating bad outcomes. This accounting system is further used to resolve conflicts among rules. The shortcoming of the classifier system is that it cannot automatically generate or delete rules. Nonetheless, by adding a genetic algorithm on top of the bucket brigade and rule-based system, one can come up with something similar to a Lucasian agent, which not only learns from experience, but can be *spontaneous* and *creative*.

While Holland’s version of the adaptive agent is much richer and more implementable than the Lucasian economic agent, and the work was already completed before the publication of [91], its formal introduction to economists came five years after the publication of [114]. In 1991, Holland and Miller published a sketch of the *artificial adaptive agent* in the highly influential journal *American Economic Review*. The first technique to implement the

Lucasian economic agent was finally ‘registered’ in economics, and genetic algorithms and classifier systems were formally added to economic analysts’ toolkits. Is five years too long? *Maybe not*, given that ‘Economic analysis has largely avoided questions about the way in which economic agents make choices when confronted by a perpetually novel and evolving world’ ([92]: 365).

What’s next? If the Lucasian economic agent is a desirable incarnation of the economic agent in economic theory, and if Holland’s artificial adaptive agent is indeed an effective implementation of it, then follow-up research can proceed in three directions: first, novel applications of this new technology, second, theoretical justifications, and finally, technical improvements to it. That is exactly what we experienced during the 1990s.

For the first line of research, Jasmina Arifovic, a student of Sargent’s, finished the first PhD dissertation that applied GAs to macroeconomics in 1991. It was not until 1994, however, that she published her work as a journal article. [10] replaced the rational representative firm in the cobweb model with Holland’s adaptive firms, and demonstrated how the adaptation of firms, driven by market forces (natural selection), collectively make the market price converge to the rational-expectations equilibrium price. Since then, a series of her papers has been published in various journals with a range of new application areas, including inflation [11], exchange rates [12] and coordination games [15].

Santa Fe Institute (SFI) Economics

Although Holland introduced this powerful toolkit to economists, he did not conduct any economic research with this toolkit himself, except for some joint work with Brian Arthur. Holland and Arthur met in September 1987 at a physics and economics Workshop hosted by the Santa Fe Institute. They had a great conversation on the nature of economics. The *chess* analogy proposed by Arthur led Holland to believe that the real problem with economics is “how do we make a science out of imperfectly smart agents exploring their way into an essentially infinite space of possibilities?” [152]: 151. On the other hand, Arthur was impressed by Holland’s approach to complex adaptive systems. Holland’s ideas of adaptation, emergence, and perpetual novelty, along with other notions, offered illuminating revelations to Arthur – insights he could never have had gained if he had confined himself to theorizing on equilibria.

This new vision of economics turned out to be the approach of the Santa Fe Institute when it established its economics program in 1988. The essence of the SFI economics was well documented by [19]. Instead of explaining genetic algorithms and classifier systems, which [92] had already done, this paper put a great emphasis on motivation. Arthur eloquently argued why the deductive approach should give way to the inductive approach when we are dealing with a model of heterogeneous agents. His paper thus built the microfoundation

of economics upon agents' cognitive processes, such as pattern recognition, concept formation, and hypothesis formulation and refutation. Arthur then showed how the dynamics of these cognitive processes can be amenable to analysis with Holland's toolkit.

Maybe the best project to exemplify the SFI approach to economics is the *artificial stock market*. This research project started in 1988. Despite progress made in 1989, journal articles documenting this research were not available until 1994. [127] first built their stock market from a standard asset pricing model [89]. They then replaced the rational representative agent in the model with Holland's artificial adaptive agents, and then simulated the market.

For Arthur, the relevance of genetic algorithms to economics is much more than just strengthening the rational expectations equilibrium. He would like to see how one can use this tool to simulate the evolution of a real economy, such as the emergence of barter trading, money, a central bank, labor unions, and even Communism. However, he understood that one should start with a more modest problem than building a whole artificial economy, and this led to *the artificial stock market*.

Given this different motive, it is also interesting to see how SFI economists programmed agents in their models, and, given their coding or programming, how complex their agents can evolve to be. [127] also used the standard ternary string to code different types of trading rules frequently used by financial market traders. Each bit of a string was randomly drawn from the ternary alphabet $\{0, 1, *\}$. Each bit corresponds to the *condition part* of a single trading rule. For example, the condition part of a *double moving average rule* could be 'The 20-period moving average of price is above the 100-period moving average.' The appropriate bit is 1 if the condition is true, and 0 if it is false. They typically used strings of 70-80 symbols – that is, the same as the number of trading rules. This defines a search space of between 3^{70} and 3^{80} possible non-redundant classifiers. However, each artificial trader has *only* 60 classifiers in their own classifier system. Consider a case with 100 computerized traders: there are at most 6000 different rules being evaluated in one single trading run. Compared with the size of the search space, the number of rules is infinitesimal.

This rather large search space is certainly beyond what [19] called the *problem complex boundary*, a boundary beyond which arriving at the deductive solution and calculating it are unlikely or impossible for human agents, and this is where the SFI stock market comes into play. It provides the right place to use genetic algorithms and a great opportunity to watch evolution. As depicted by [19]: 24

“We find no evidence that market behavior ever settles down; the population of predictors continually co-evolves. One way to test this is to take agents out of the system and inject them in again later on.

If market behavior is stationary they should be able to do as well in the future as they are doing today. But we find that when we ‘freeze’ a successful agent’s predictors early on and inject the agent into the system much later, the formerly successful agent is now a dinosaur. His predictions are unadapted and perform poorly. The system has changed. From our vantage point looking in, the market – the ‘only game in town’ on our computer – looks much the same. But internally it co-evolves and changes and transforms. It never settles.”

Maybe the real issue is not whether GAs are used to strengthen the idea of REE, or to simulate artificial life, but *how we program adaptive agents*. This is crucial because different programming schemes may lead to different results. As Hahn pointed out, while there is only one way to be perfectly rational, there are an infinite number of ways to be partially rational ([152]: 250–251). This unlimited ‘degree of freedom’ of programming adaptive agents was also noticed by [132]: 2

“This area is wilderness because the researcher faces so many choices after he decides to forgo the discipline provided by equilibrium theorizing.”

Arthur would consider letting the agents start off ‘perfectly stupid’, and become smarter and smarter as they learn from experience. Now comes the core of the issue: how to program agents so that they can be initialized as perfectly stupid individuals, but can potentially get very smart. To answer this question, let us go back to the origin of genetic algorithms.

List Programming (LISP)

It is interesting to note that the binary strings initiated by Holland were originally motivated by an analogy to machine codes. After decoding, they can be computer programs written in a specific language, say, LISP or FORTRAN. Therefore, when a GA is used to evolve a population of binary strings, it behaves as if it is used to evolve a population of computer programs. If a decision rule is explicit enough not to cause any confusion in implementation, then one should be able to write it in a computer program. It is the *population of computer programs* (or their machine codes) which provides the most general representation of the *population of decision rules*. However, the equivalence between computer programs and machine codes *breaks down* when what is coded is the parameters of decision rules rather than decision rules (programs) themselves, as we often see in economic applications with GAs. The original meaning of evolving binary strings as evolving computer programs is lost.

The gradual loss of the original function of GAs has finally been noticed by Koza [104]. He chose the language LISP as the medium for the programs

created by genetic programming (GP) because the syntax of LISP allows computer programs to be manipulated easily like the bit strings in GAs, so that the same genetic operations used on bit strings in GAs can also be applied to GP.

LISP S-expressions consist of either *atoms* or *lists*. Atoms are either members of a *terminal set*, that comprise the data (for example, constants and variables) to be used in the computer programs, or members of a *function set* that consists of a number of pre-specified functions or operators that are capable of processing any data value from the terminal set *and* any data value that results from the application of any function or operator in the function set. Lists are collections of atoms or lists, grouped within parentheses. In the LISP language, everything is expressed in terms of operators operating on some operands. The operator appears as the leftmost element in the parentheses and is followed by its operands and a closing (right) parenthesis. For example, the S-expression $(+X\ 3)$ consists of three atoms: from the left-most to right-most they are the function '+', the variable X and the constant 3. As another example, $(\times X\ (-Y\ 3))$ consists of two atoms and a list. The two atoms are the function '×' and the variable 'X,' which is then followed by the list $(-Y\ 3)$.

LISP was invented in the late 1950s by John McCarthy at MIT as a formalism for reasoning about the use of certain kinds of logical expressions, called recursion equations. LISP possesses unique features that make it an excellent medium for complex compositions of functions of various types, handling hierarchies, recursion, logical functions, self-modifying computer programs, self-executing computer programs, iterations, and structures whose size and shapes are dynamically determined. The most significant of these features is the fact that LISP descriptions of processes (routines) can themselves be represented and manipulated as LISP data (subroutines). As Koza demonstrated, LISP's flexibility in handling procedures as data makes it one of the most convenient languages in existence for exploring the idea of evolving computer programs genetically [104]. However, Koza and others have noted that the use of LISP is not necessary for genetic programming; what *is* important for genetic programming is the implementation of a LISP-like environment, where individual expressions can be manipulated like data, and are immediately executable.

Symbolic Regression

The distinguishing feature of GP is manifested by its first type of application in economics, known as *symbolic regression*. In symbolic regression, GP is used to discover the underlying data-generation process of a series of observations. While this type of application is well known to econometricians, the perspective from GP is novel. As Koza stated,

“An important problem in economics is finding the mathematical relationship between the empirically observed variables measuring a system. In many conventional modeling techniques, one necessarily begins by selecting the size and shape of the model. After making this choice, one usually then tries to find the values of certain coefficients required by the particular model so as to achieve the best fit between the observed data and the model. But, in many cases, *the most important issue is the size and shape of the model itself.*” [105]: 57 (italics added)

Econometricians offer no general solution to the determination of size and shape (the functional form), but for Koza, finding the functional form of the model can be viewed as *searching a space of possible computer programs* for the particular computer program which produces the desired output for given inputs.

Koza employed GP to rediscover some basic physical laws from experimental data, for example, Kepler’s third law and Ohm’s law [104]. He then also applied it to eliciting a very fundamental economic law, namely, the *quantity theory of money* or the *exchange equation* [105]. Genetic programming was thus formally demonstrated as a *knowledge discovery* tool. This was probably the closest step ever made toward the original motivation of Holland’s invention: ‘Instead of trying to write your programs to perform a task you don’t quite know how to do, *evolve them.*’ Indeed, Koza did not evolve the parameters of an arbitrary chosen equation; instead, he evolved the whole equation from scratch. This style of application provides an evolutionary determination of bounded rationality.

Koza motivated a series of economic applications of genetic programming in the mid-1990s [105]. Chen and Yeh applied genetic programming to rediscovering the *efficient market hypothesis* in a financial time series [52]. They then moved one step forward to propose an alternative formulation of the efficient market hypothesis in the spirit of the *Kolmogorov complexity* of algorithms for pattern extraction from asset price data [54]. [54] and [141] employed GP to discover the underlying chaotic laws of motion of time series data. [6] and [123] also adopted a GP approach to discover profitable technical trading rules for the foreign exchange market and the stock market, respectively. Another area in which GP was actively applied is *option pricing*. [61] used GP for hedging derivative securities. [98] showed that genetically determined formulas outperformed most frequently quoted analytical approximations in calculating the implied volatility based on the Black-Scholes model. [65] and [99] derived approximations for calculating option prices and showed that GP-models outperformed various other models presented in the literature.

Needless to say, one can expect many more applications of GP to the automatic discovery of economic and financial knowledge (automatic generation

of economic and financial knowledge in terms of their computer-programmed representations). However, its significant contribution to economics should not be mistaken for a perfect solution to knowledge discovery, data mining, or, more generally, *function optimization*. In a nutshell, genetic programming should be used to grow *evolving hierarchies* of building blocks (subroutines) – the basic units of learning and information, from an immense space of subroutines. All evolution can do is look for improvements, not perfection. Holland believed that these evolving hierarchies are generic in adaptation, and can play a key role in understanding human learning and adaptive processes.

4 Agent-Based Economic Simulations with CI

In this Section, we shall review the applications of CI to ACE. Given the size limitations, it is impossible to give an exhaustive survey here. We can therefore only review a few selected areas which we consider most representative and characterize the early development of the literature. We shall give a macroscopic view of the literature in Sects. 5.1 and 5.2, and introduce some most recent developments, which point to the future research, in Sect. 5.3.

4.1 The Cobweb Model

The cobweb model is a familiar playground in which to investigate the effects of production decisions on price dynamics. In this model consumers base their decisions on the current market price, but producers decide how much to produce based on the past prices. Agricultural commodities serve as a good example of the cobweb model. This model plays an important role in macroeconomics, because it is the place in which the concept ‘rational expectations’ originated [121]. Moreover, it is also the first neo-classical macroeconomic prototype to which an agent-based computational approach was applied [10]. This Section will first briefly formulate the cobweb model and then review the work on agent-based modeling of the cobweb model.

Consider a competitive market composed of n firms which produce the same goods by employing the same technology and which face the same cost function described in Eqn. (72):

$$c_{i,t} = xq_{i,t} + \frac{1}{2}ynq_{i,t}^2 \quad (72)$$

where $q_{i,t}$ is the quantity supplied by firm i at time t , and x and y are the parameters of the cost function. Since at time $t - 1$, the price of the goods at time t , P_t , is not available, the decision about optimal $q_{i,t}$ must be based on the expectation (forecast) of P_t – that is, $P_{i,t}^e$. Given $P_{i,t}^e$ and the cost function $c_{i,t}$, the expected profit of firm i at time t can be expressed as follows:

$$\pi_{i,t}^e = P_{i,t}^e q_{i,t} - c_{i,t} \quad (73)$$

Given $P_{i,t}^e$, $q_{i,t}$ is chosen at a level such that $\pi_{i,t}^e$ can be maximized and, according to the first-order condition, is given by

$$q_{i,t} = \frac{1}{yn} (P_{i,t}^e - x) \quad (74)$$

Once $q_{i,t}$ is decided, the aggregate supply of the goods at time t is fixed and P_t , which sets demand equal to supply, is determined by the demand function:

$$P_t = A - B \sum_{i=1}^n q_{i,t} \quad (75)$$

where A and B are parameters of the demand function.

Given P_t , the actual profit of firm i at time t is:

$$\pi_{i,t} = P_t q_{i,t} - c_{i,t} \quad (76)$$

The neo-classical analysis simplifies the cobweb model by assuming the homogeneity of market participants – in other words, a representative agent. In such a setting, it can be shown that the homogeneous rational expectations equilibrium price (P^*) and quantity (Q^*) are ([53]: 449):

$$P_t^* = \frac{Ay + Bx}{B + y}; \quad Q_t^* = \frac{A - x}{B + y} \quad (77)$$

CI in the Agent-Based Cobweb Model

The neo-classical analysis based on homogeneous agents provides us with a limited understanding of the price dynamics or price instability in a real market, since firms' expectations of the prices and the resultant production decisions in general must be heterogeneous. Using genetic algorithms to model the adaptive behavior of firms' production, Arifovic gave the first agent-based model of the cobweb model [10]. She applied two versions of GAs to this model. The basic GA involves three genetic operators: reproduction, crossover, and mutation. Arifovic found that in each simulation of the basic GA, individual quantities and prices exhibited fluctuations for its entire duration and did not result in convergence to the rational expectations equilibrium values, which is quite inconsistent with experimental results with human subjects.

Arifovic's second GA version – the *augmented GA* – includes the election operator in addition to reproduction, crossover, and mutation. The election operator involves two steps. First, crossover is performed. Second, the potential fitness of the newly-generated offspring is compared with the actual fitness values of its parents. Among the two offspring and two parents, the two highest fitness individuals are then chosen. The purpose of this operator is to overcome difficulties related to the way mutation influences the convergence

process, because the election operator can bring the variance of the population rules to zero as the algorithm converges to the equilibrium values.

The results of the simulations show that the augmented GA converges to the rational expectations equilibrium values for all sets of cobweb model parameter values, including both stable and unstable cases, and can capture several features of the experimental behavior of human subjects better than other simple learning algorithms. To avoid the arbitrariness of choice of an adaptive scheme, [114] suggested that comparison of the behavior of adaptive schemes with behavior observed in laboratory experiments with human subjects can facilitate the choice of a particular adaptive scheme. From this suggestion, the GA could be considered an appropriate choice to model learning agents in a complex system.

The application of genetic programming to the cobweb model started from [53], who compared the learning performance of GP-based learning agents with that of GA-based learning agents. They found that, like GA-based learning agents, GP-based learning agents also can learn the homogeneous rational expectations equilibrium price under both the stable and unstable cobweb case. However, the phenomenon of ‘price euphoria’, which did not happen in [10], does show up quite often at the early stages of the GP experiments. This is mainly because agents in their setup were initially endowed with very limited information as compared to [10]. Nevertheless, GP-based learning can quickly coordinate agents’ beliefs so that the emergence of price euphoria is only temporary. Furthermore, unlike [10], Chen and Yeh did not use the election operator. Without the election operator, the rational expectations equilibrium is exposed to potentially persistent perturbations due to agents’ adoption of the new, but untested, rules. However, what shows up in [53] is that the market can still bring any price deviation back to equilibrium. Therefore, the self-stabilizing feature of the market, known as the ‘invisible hand’, is more powerfully replicated in their GP-based artificial market.

The self-stabilizing feature of the market demonstrated in [53] was further tested with two complications. In the first case, [55] introduced a population of speculators to the market and examined the effect of speculations on market stability. In the second case, the market was perturbed with a structural change characterized by a shift in the demand curve; [57] then tested whether the market could restore the rational expectations equilibrium. The answer to the first experiment is generally negative, namely that speculators do not enhance the stability of the market; on the contrary, they destabilize the market. Only in special cases when trading regulations – such as the transaction cost and position limit – were tightly imposed could speculators enhance the market stability. The answer for the second experiment is, however, positive. Chen and Yeh showed that GP-based adaptive agents could detect the shift in the demand curve and adapt to it [57]. Nonetheless, the transition phase was non-linear and non-smooth; one can observe slumps,

crashes, and bursts in the transition phase. In addition, the transition speed is uncertain. It could be fast, but could be slow as well.

This series of studies on the cobweb model enriches our understanding of the self-stabilizing feature of the market. The market has its limit, beyond which it can become unstable with crazy fluctuations. However, imposing trading regulations may relax the limit and enhance market stability. One is still curious to know where the self-stabilizing capability comes from in the first place. Economists have known for a long time that it comes from the free competition principle, or the survival-of-the-fittest principle. In GA or GP, this principle is implemented through *selection pressure*. Chen studied the role of selection pressure by replacing the usual proportionate selection scheme with the one based on the approximate uniform distribution, showing that if selection pressure is removed or alleviated, then the self-stabilizing feature is lost [37]. In a word, selection pressure plays the role of the invisible hand in economics.

It is interesting to know whether the time series data generated by the artificial market can replicate some dynamic properties observed in the real market. [46] and [57] started the analysis of the time series data generated from the artificial market. The time series data employed was generated by simulating the agent-based cobweb model with the presence of speculators. It was found that many stylized features well documented in financial econometrics can in principle be replicated from GP-based artificial markets, which include leptokurtosis, non-IIDness, and volatility clustering. Furthermore, [57] performed a CUSUMSQ test, a statistical test for structural change, on the data. The test indicated the presence of structural changes in the data, which suggested that the complex interaction process of these GP-based producers and speculators can even generate endogenous structural changes.

4.2 Overlapping Generations Models

While there are several approaches to introducing dynamic general equilibrium structures to economics, the overlapping generations model (hereafter, OLG) may be regarded as the most popular one in current macroeconomics. Over the last two decades, the OLG model has been extensively applied to studies of savings, bequests, demand for assets, prices of assets, inflation, business cycles, economic growth, and the effects of taxes, social security, and budget deficits. In the following, we shall first give a brief illustration of a simple OLG model of inflation, a *two-period* OLG model.

Two-Period OLG Model

A simple OLG model can be described as follows. It consists of overlapping generations of two-period-lived agents. At time t , N young agents are born. Each of them lives for two periods $(t, t+1)$. At time t , each of them is endowed

with e^1 units of a perishable consumption good, and with e^2 units at time $t + 1$ ($e^1 > e^2 > 0$). Presumably e^1 is assumed to be greater than e^2 in order to increase the likelihood (but not ensure) that agents will choose to hold money from period 1 to 2 so as to push value forward. An agent born at time t consumes in both periods. Term c_t^1 is the consumption in the first period (t), and c_t^2 the second period ($t + 1$). All agents have identical preference given by

$$U(c_t^1, c_t^2) = \ln(c_t^1) + \ln(c_t^2) \tag{78}$$

In addition to perishable consumption goods, there is an asset called *money* circulating in the society. The nominal money supply at time t , denoted by H_t , is exogenously determined by the government and is held distributively by the old generation at time t . For convenience, we shall define h_t to be $\frac{H_t}{N}$ – in other words, the nominal per capita money supply.

This simple OLG gives rise to the following agent’s maximization problem at time t :

$$\begin{aligned} & \max_{(c_{i,t}^1, c_{i,t}^2)} \ln(c_{i,t}^1) + \ln(c_{i,t}^2) \\ \text{such that } & c_{i,t}^1 + \frac{m_{i,t}}{P_t} = e^1, \quad c_{i,t}^2 = e^2 + \frac{m_{i,t}}{P_{t+1}} \end{aligned} \tag{79}$$

where $m_{i,t}$ represents the nominal money balances that agent i acquires at time period t and spends in time period $t + 1$, and P_t denotes the nominal price level at time period t . Since P_{t+1} is not available at period t , what agents actually can do is to maximize their expected utility $E(U(c_t^1, c_t^2))$ by regarding P_{t+1} as a random variable, where $E(\cdot)$ is the expectation operator. Because of the special nature of the utility function and budget constraints, the first-order conditions for this expected utility maximization problem reduce to the certainty equivalence form:

$$c_{i,t}^1 = \frac{1}{2}(e^1 + e^2 \pi_{i,t+1}^e) \tag{80}$$

where $\pi_{i,t+1}^e$ is agent i ’s expectation of the inflation rate $\pi_{t+1}(\equiv \frac{P_{t+1}}{P_t})$. This solution tells us the optimal decision of savings for agent i given her expectation of the inflation rate, $\pi_{i,t+1}^e$.

Suppose the government deficit G_t is all financed through seignorage and is constant over time ($G_t = G$). We can then derive the dynamics (time series) of nominal price $\{P_t\}$ and inflation rate $\{\pi_t\}$ from Eqn. (80). To see this, let us denote the savings of agent i at time t by $s_{i,t}$. Clearly,

$$s_{i,t} = e^1 - c_{i,t}^1 \tag{81}$$

From Eqn.(79), we know that

$$m_{i,t} = s_{i,t} P_t, \quad \forall i, t \tag{82}$$

In equilibrium, the nominal aggregate money demand must equal nominal money supply, namely,

$$\sum_{i=1}^N m_{i,t} = H_t = H_{t-1} + GP_t, \quad \forall t \tag{83}$$

The second equality says that the money supply at period t is the sum of the money supply at period $t - 1$ and the nominal deficit at period t , GP_t . This equality holds, because we assume the government deficits are all financed by seignorage.

Summarizing Eqns. (82) and (83), we get

$$\sum_{i=1}^N s_{i,t}P_t = \sum_{i=1}^N s_{i,t-1}P_{t-1} + GP_t \tag{84}$$

The price dynamics are hence governed by the following equation:

$$\pi_t = \frac{P_t}{P_{t-1}} = \frac{\sum_{i=1}^N s_{i,t-1}}{\sum_{i=1}^N s_{i,t} - G} \tag{85}$$

Now suppose that each agent has perfect foresight, that is,

$$\pi_{i,t}^e = \pi_t, \quad \forall i, t \tag{86}$$

By substituting the first-order condition Eqn. (80) into Eqn. (84), the paths of equilibrium inflation rates under perfect foresight dynamics are then

$$\pi_{t+1} = \frac{e^1}{e^2} + 1 - \frac{2g}{e^2} - \left(\frac{e^1}{e^2}\right)\left(\frac{1}{\pi_t}\right) \tag{87}$$

where $g = \frac{G}{N}$ is the real per capita deficit.

At steady state ($\pi_{t+1} = \pi_t$), Eqn. (87) has two real stationary solutions (fixed points), a low-inflation stationary equilibrium, π_L^* , and a high-inflation one, π_H^* , given by

$$\pi_L^* = \frac{1 + \frac{e^1}{e^2} - \frac{2g}{e^2} - \sqrt{\left(1 + \frac{e^1}{e^2} - \frac{2g}{e^2}\right) - 4\frac{e^1}{e^2}}}{2} \tag{88}$$

$$\pi_H^* = \frac{1 + \frac{e^1}{e^2} - \frac{2g}{e^2} + \sqrt{\left(1 + \frac{e^1}{e^2} - \frac{2g}{e^2}\right) - 4\frac{e^1}{e^2}}}{2} \tag{89}$$

Despite its popularity, the OLG models are well known for their multiplicity of equilibria, in our case, the coexistence of two inflation equilibria: Eqns. (88) and (89). Things can be even more intriguing if these equilibria have different welfare implications. In our case, the one with a higher inflation rate is the Pareto-inferior equilibrium, whereas the one with a lower inflation rate is the Pareto-superior equilibrium.

CI in Agent-Based OLG Models of Inflation

To see whether decentralized agents are able to coordinate intelligently to single out a Pareto-superior equilibrium rather than be trapped in a Pareto-inferior equilibrium, [11] proposed the first agent-based modification of an OLG model of inflation. She applied genetic algorithms (GAs) towards modeling the learning and adaptive behavior of households. In her study, GA-based agents were shown to be able to select the Pareto-superior equilibrium. She further compared the simulation results based on GAs with those from laboratories with human subjects, concluding that GAs were superior to other learning schemes, such as the recursive least squares.

This line of research was further carried out in [27,31–33] and [69]. Bullard and Duffy made the distinction between two implementations of GA learning: depending on what to encode, GA learning can be implemented in two different ways, namely, learning how to optimize [11] and learning how to forecast [33]. It was found that these two implementations lead to the same result: agents can indeed learn the Pareto-superior equilibrium. The only difference is the speed of convergence. The ‘learning how to forecast’ version of genetic algorithm learning converges faster than the ‘learning how to optimize’ implementation studied by [11]. Nevertheless, a robust analysis showed that coordination was more difficult when the number of inflation values considered (search space) by agents was higher, when government deficits increased, and when agents entertained inflation rate forecasts outside the bounds of possible stationary equilibria.

Chen and Yeh generalized Bullard and Duffy’s ‘learning how to forecast’ version of GA learning with GP [56]. In [33], what agents learn is just a the inflation rate *per se*, rather than regularity about its *motion*, which is a function. Chen and Yeh considered it too restrictive to learn just a number. From [86], if the equilibrium of an OLG is characterized by limit cycles or strange attractors rather than by fixed points, then what agents need to learn is not just a number, but a functional relationship, such as $x_t = f(x_{t-1}, x_{t-2}, \dots)$. Chen and Yeh therefore generalized Bullard and Duffy’s evolution of *beliefs* from a sequence of populations of numbers to a sequence of populations of functions. Genetic programming serves as a convenient tool to make this extension.

The basic result observed in [56] is largely consistent with [10] and [33], namely, agents being able to coordinate their actions to achieve the Pareto-superior equilibrium. Furthermore, their experiments showed that the convergence is not sensitive to the initial rates of inflation. Hence, the Pareto-superior equilibrium has a large domain of attraction. A test on a structural change (a change in deficit regime) was also conducted. It was found that GP-based agents were capable of converging very fast to the new low-inflationary stationary equilibrium after the new deficit regime was imposed. However,

the basic result was not insensitive to the dropping of the survival-of-the-fittest principle. When that golden principle was not enforced, we experienced dramatic fluctuations of inflation and occasionally the appearance of super inflation. The agents were generally worse off.

Birchenhall and Lin provided perhaps the most extensive coverage of robustness checks ever seen in agent-based macroeconomic models [27]. Their work covers two different levels of GA designs: one is genetic operators, and the other is architecture. For the former, they consider different implementations of the four main GA operators – namely, selection, crossover, mutation, and election. For the latter, they consider a single-population GA (social learning) versus a multi-population GA (individual learning). They found that Bullard and Duffy’s results are sensitive to two main factors: the election operator and architecture. Their experimental results in fact lend support to some early findings – for example, the significance of the election operator [10] and the different consequences of social learning and individual learning [151]. What is particularly interesting is that individual learning reduces the rate of convergence to the same belief. This is certainly an important finding, because most studies on the convergence of GAs to Pareto optimality are based on the social learning version.

Ballard and Duffy studied a more complicated version of the two-period OLG model, based on [86]. They consider the following utility function for the households [32],

$$U(c_t^1, c_t^2) = \frac{\ln(c_t^1)^{1-\rho_1}}{1-\rho_1} + \frac{\ln(c_t^2)^{1-\rho_2}}{1-\rho_2} \tag{90}$$

Under time-separable preferences and provided that the value of the coefficient of relative risk aversion for the old agent (ρ_2) is high enough and that of the young agents is low enough (ρ_1), [86] showed that stationary perfect-foresight equilibria also may exist in which the equilibrium dynamics are characterized either as *periodic* or *chaotic trajectories* for the inflation rate, and these complicated stationary equilibria are also Pareto optimal. To have these possibilities, they set ρ_2 equal to 2 and then increased the value of this preference parameter up to 16 by increments of 0.1, while fixed ρ_1 at 0.5 in all cases.

The forecast rule considered by Bullard and Duffy is to use the price level that was realized $k + 1$ periods in the past as the forecast of next period’s price level, namely,

$$P_{i,t}^e = P_{t-k-1}, \quad k \in [0, \bar{k}] \tag{91}$$

In their case, \bar{k} was set to 256, which allows the agents to take actions consistent with a periodic equilibrium of an order as high as 256. Alternatively,

agent i 's forecast of the gross inflation factor between dates t and $t + 1$ is given by

$$\pi_{i,t}^e = \frac{P_{t-k-1}}{P_{t-1}} \quad (92)$$

As usual, the lifetime utility function was chosen as the fitness function to evaluate the performance of a particular forecast rule. Instead of roulette wheel selection, tournament selection was applied to create the next generation.

It was found that the stationary equilibria on which agents coordinate were always relatively simple – either a steady state or a low-order cycle. For low values of ρ_2 (in particular, those below 4.2), they observed convergence to the monetary steady state in every experiment, which is the same prediction made by the limited backward perfect-foresight dynamics. As ρ_2 was increased further, the limiting backward perfect foresight dynamics displayed a bifurcation, with the monetary steady state losing stability and never regaining it for values of $\rho_2 \geq 4.2$. However, in their system with learning, the monetary steady state was always a limit point in at least 1 of the 10 experiments conducted for each different value of ρ_2 . Also, for $\rho_2 \geq 4.2$, their system often converged, in at least one experiment, to a period-2 stationary equilibrium, even in cases in which that equilibrium, too, had lost its stability in the backward perfect-foresight dynamics.

It is difficult, however, for an economy comprised of optimizing agents with initial heterogeneous beliefs to coordinate on especially complicated stationary equilibria, such as the period- k cycles where $k \geq 3$. In particular, the period-3 cycle that is stable in the backward perfect-foresight dynamics for values $\rho_2 \geq 13$ was never observed in their computational experiments. Interesting enough, three is the last entry of *Sarkovskii's ordering*, whereas one, two and four are first few entries.

They also found that the time it took agents to achieve coordination tended to increase with the relative risk aversion of the old agents over a large portion of the parameter space. Usually, it was the case when the system converged to the period-2 cycle. Moreover, when cycles exist, the transient dynamics of their systems could display qualitatively complication dynamics for long periods of time before eventually to relatively simple, low-periodicity equilibria.

A related phenomenon to cyclical equilibria is *sunspot equilibria*. The sunspot variable is the variable which has no intrinsic influence on an economy – in other words, it has nothing to do with an economy's fundamentals. Sunspot equilibria exist if the sunspot variable can impact the economy simply because a proportion of agents believe so and act accordingly to their belief. [22] showed that the connection between cyclical and sunspot equilibria is very close. They proved that a two-state stationary sunspot equilibrium exists if and only if a period-2 equilibrium exists. [69] started with an OLG model of

inflation comparable to [32]. He studied an economy whose households have the following utility function,

$$U(c_t^1, c_t^2) = 0.1[c_t^1]^{0.9} + 10 - \left[\frac{10}{1 + c_t^2}\right]^2 \tag{93}$$

This utility function has the property that the concavity with respect to c_t^1 is much smaller than the concavity with respect to c_t^2 , which is necessary for the existence of a periodic equilibrium [86].

He first found that in cases where periodic equilibria exist, households' beliefs were successfully coordinated to the period-2 cycle rather than the steady state. He then assumed all households to be sunspot believers and showed that households' beliefs converged to the sunspot equilibrium. In that case, the observed values of the price levels are completely governed by something which has nothing to do with the economy's fundamentals. Finally, he relaxed the assumption by simulating an explicit contest between 'sunspot believers' and 'sunspot agnostics'. The simulation showed that in most cases, the population consisted, after a rather short period, only of households whose actions depended on the value of the sunspot variable.

4.3 Foreign Exchange Rate Fluctuations

Another popular class of OLG models to which an agent-based approach is applied is the the OLG model of foreign exchange rates, which is a version of the two-country OLG model with fiat money [96].

The OLG Model of Exchange Rate

There are two countries in the model. The residents of both countries are identical in terms of their preferences and lifetime endowments. The basic description of each country is the same as the single-country OLG model. Each household of generation t is endowed with e^1 units of a perishable consumption good at time t , and e^2 of the good at time $t + 1$, and consumes c_t^1 of the consumption good when young and c_t^2 when old. Households in both countries have common preferences given by

$$U(c_t^1, c_t^2) = \ln(c_t^1) + \ln(c_t^2). \tag{94}$$

The government of each country issues its own unbacked currency, $H_{1,t}$ and $H_{2,t}$. Households can save only through acquiring these two currencies. There are no legal restrictions on holdings of foreign currency. Thus, the residents of both countries can freely hold both currencies in their portfolios. A household at generation t solves the following optimization problem at time t :

$$\begin{aligned} & \max_{(c_{i,t}^1, m_{i,1,t})} \ln(c_{i,t}^1) + \ln(c_{i,t}^2) \\ \text{such that } & c_{i,t}^1 + \frac{m_{i,1,t}}{P_{1,t}} + \frac{m_{i,2,t}}{P_{2,t}} = e^1, \quad c_{i,t}^2 = e^2 + \frac{m_{i,1,t}}{P_{1,t+1}} + \frac{m_{i,2,t}}{P_{2,t+1}} \end{aligned} \tag{95}$$

where $m_{i,1,t}$ is household i ' nominal holdings of currency 1 acquired at time t , $m_{i,2,t}$ is household i ' nominal holdings of currency 2 acquired at time t , $P_{1,t}$ is the nominal price of the good in terms of currency 1 at time t , and $P_{2,t}$ is the nominal price of the good in terms of currency 2 at time t . The savings of household i at time t by $s_{i,t}$ is

$$s_{i,t} = e^1 - c_{i,t}^1 = \frac{m_{i,1,t}}{P_{1,t}} + \frac{m_{i,2,t}}{P_{2,t}} \tag{96}$$

The exchange rate e_t between the two currencies is defined as $e_t = P_{1,t}/P_{2,t}$. When there is no uncertainty, the return on the two currencies must be equal,

$$R_t = R_{1,t} = R_{2,t} = \frac{P_{1,t}}{P_{1,t+1}} = \frac{P_{2,t}}{P_{2,t+1}}, \quad t \geq 1 \tag{97}$$

where $R_{1,t}$ and $R_{2,t}$ are the gross real rate of return between t and $t + 1$, respectively. Rearranging Eqn. (97), we obtain

$$\frac{P_{1,t+1}}{P_{2,t+1}} = \frac{P_{1,t}}{P_{2,t}} \quad t \geq 1 \tag{98}$$

From Eqn. (98) it follows that the exchange rate is constant over time:

$$e_{t+1} = e_t = e, \quad t \geq 1 \tag{99}$$

Savings demand derived from household's maximization problem is given by

$$s_{i,t} = \frac{m_{i,1,t}}{p_{1,t}} + \frac{m_{i,2,t}}{p_{2,t}} = \frac{1}{2}[e^1 - e^2 \frac{1}{R_t}] \tag{100}$$

Aggregate savings of the world at time period t , S_t , are equal to the sum of their savings in terms of currency 1, $S_{1,t}$, and in terms of currency 2, $S_{2,t}$. With the homogeneity assumption, we have

$$S_{1,t} = \sum_{i=1}^{2N} \frac{m_{i,1,t}}{P_{1,t}} = \frac{2Nm_{1,t}}{P_{1,t}} \tag{101}$$

and

$$S_{2,t} = \sum_{i=1}^{2N} \frac{m_{i,2,t}}{P_{2,t}} = \frac{2Nm_{2,t}}{P_{2,t}} \tag{102}$$

The equilibrium condition in the loan market requires

$$S_t = S_{1,t} + S_{2,t} = N[e^1 - e^2 \frac{P_{1,t+1}}{P_{1,t}}] = \frac{H_{1,t} + H_{2,t}e}{P_{1,t}} \tag{103}$$

Eqn. (103) only informs us of the real saving in terms of the real world money demand. This equation alone cannot determine the household real

demands for each currency. Hence, this equation cannot uniquely determine a set of price $(P_{1,t}, P_{2,t})$, and leave the exchange rate indeterminate as well. This is known as the famous *indeterminacy of exchange rate proposition*. The proposition says that if there exists a monetary equilibrium in which both currencies are valued at some exchange rate e , then there exists a monetary equilibrium at any exchange rate $\hat{e} \in (0, \infty)$ associated with a different price sequence $\{\hat{P}_{1,t}, \hat{P}_{2,t}\}$ such that

$$R_t = \frac{P_{1,t}}{P_{1,t+1}} = \frac{P_{2,t}}{P_{2,t+1}} = \frac{\hat{P}_{1,t}}{\hat{P}_{1,t+1}} = \frac{\hat{P}_{2,t}}{\hat{P}_{2,t+1}} \quad (104)$$

and

$$S_t = \frac{H_{1,t} + H_{2,t}e}{P_{1,t}} = \frac{H_{1,t} + H_{2,t}\hat{e}}{\hat{P}_{1,t}} \quad (105)$$

where

$$\hat{P}_{1,t} = \frac{H_{1,t} + \hat{e}H_{2,t}P_{1,t}}{H_{1,t} + eH_{2,t}}, \quad \hat{P}_{2,t} = \frac{\hat{P}_{1,t}}{\hat{e}}. \quad (106)$$

Rearranging Eqn. (103), one can derive the law of motion of $P_{1,t}$

$$P_{1,t+1} = \frac{e^1}{e^2}P_{1,t} - \frac{H_{1,t} + eH_{2,t}}{Ne^2} \quad (107)$$

For any given exchange rate e , this economy with constant supplies of both currencies, H_1 and H_2 , has a steady-state equilibrium, namely,

$$P_{1,t+1} = P_{1,t} = P_1^* = \frac{H_1 + eH_2}{N(e^1 - e^2)} \quad (108)$$

Like e , the level of P_1^* is also indeterminate. In addition, since households are indifferent between the currencies that have the same rates of return in the homogeneous-expectations equilibrium, the OLG model in which agents are rational does not provide a way to determine the portfolio $\lambda_{i,t}$, which is the fraction of the savings placed into currency 1.

CI in Agent-Based OLG Models of the Exchange Rate

In order to examine the behavior of the exchange rate and the associated price dynamics, Arifovic initiated the agent-based modeling of the exchange rate in the context of the OLG model [12]. In the OLG model of the exchange rate, households have two decisions to make when they are young, namely, saving ($s_{i,t}$) and portfolio ($\lambda_{i,t}$). These two decisions were encoded by concatenation of two binary strings, the first of which encoded $s_{i,t}$, whereas the second of which encoded $\lambda_{i,t}$. The single-population augmented genetic algorithm was then applied to evolve these decision rules. The length of a binary string, l , is 30: The first 20 elements of a string encode the first-period consumption

of agent i of generation t ; the remaining 10 elements encode the portfolio fraction of agent i :

$$\underbrace{010100\dots110}_{20 \text{ bits: } s_{i,t}} \underbrace{101..001}_{10 \text{ bits: } \lambda_{i,t}}$$

While Eqn. (99) predicts the constancy of the exchange rate, genetic algorithm simulations conducted by [12] indicated no sign of the setting of the exchange rate to a constant value. Instead, they showed persistent fluctuations of the exchange rate. Adaptive economic agents in this model can, in effect, endogenously generate *self-fulfilling arbitrage opportunities*, which in turn make exchange rates continuously fluctuate.

The fluctuating exchange rate was further examined using formal statistical tests in both [12] and [16]. First, in [12], the stationarity (Dickey-Fuller) test was applied to examine whether the exchange rate series is non-stationary. The result of the test did not indicate non-stationarity. Second, [16] analyzed the statistical properties of the exchange rate returns, namely, the logarithm of e_t/e_{t-1} . The independence tests (Ljung-Box-Pierce and BDS) clearly rule out the lack of persistence (dependence) in the return series. Third, they plotted the phase diagrams of the return series and found that there is a well-defined attractor for all series. The shapes of the attractor are robust to the changes in the OLG model parameters as well as to the changes in the GA parameters. Fourth, to verify that this attractor is chaotic, the largest two Lyapunov exponents were calculated. The largest Lyapunov exponent is positive in all series, which supports that attractors under investigation are chaotic. Finally, volatility clustering was also found to be significant in the return series. This series of econometric examinations confirms that agent-based modeling is able to replicate some stylized facts known in financial markets.

Arifovic considered a different application of GAs to modeling the adaptive behavior of households [14]. Instead of savings and portfolio decision rules, she turned to the forecasting behavior of households. The forecasting models of exchange rates employed by agents are simple moving-average models. They differ in the rolling window size, which are endogenously determined and can be time-variant. What is encoded by GAs is the size of the rolling window rather than the usual savings and portfolio decision. Simulations with this new coding scheme resulted in the convergence of the economies to a single-currency equilibrium – that is, the collapse of one of the two currencies. This result was not found in [12]. This study therefore shows that different implementations of GA learning may have non-trivial effects on the simulation results. In one implementation, one can have persistent fluctuation of the exchange rate [12]; in another case, one can have a single-currency equilibrium.

Following the design of [81], Arifovic combined two different applications of GA learning. In addition to the original population of agents, who are learning how to forecast, she added another population of agents, who are learning how to optimize [14]. Nevertheless, unlike [81], these two populations

of agents did not compete with each other. Instead, they underwent separate genetic algorithm updating. Simulations with these two separate evolving populations did not have the convergence to single currency equilibrium, but were characterized instead by persistent fluctuation.

A different scenario of the currency collapse is also shown in [13], which is an integration of the OLG model of exchange rate with the OLG model of inflation. In this model, the governments of both countries have constant deficits ($G_i, i = 1, 2$) which were financed via seignorage,

$$G_i = \frac{H_{i,t} - H_{i,t-1}}{P_{i,t}}, \quad i = 1, 2 \quad (109)$$

Combining Eqns. (103) and (109) gives the condition for the monetary equilibrium in which both governments finance their deficits via seignorage:

$$G_1 + G_2 = S_t - S_{t-1}R_{t-1} \quad (110)$$

This integrated model inherits the indeterminacy of the exchange rate from the OLG model of the exchange rate and the indeterminacy of the inflation rate from the OLG model of inflation. Any constant exchange rate e ($e \in (0, \infty)$) is an equilibrium that supports the same stream of government deficits (G_1, G_2), and the same equilibrium gross rate of return (and thus the same equilibrium savings). The existence of these equilibrium exchange rates indicates that the currencies of both countries are valued despite the difference of the two countries' deficits. In fact, in equilibrium the high-deficit country and the low-deficit county experience the same inflation rate, and hence so do their currencies' rates of return. Nonetheless, since the high-deficit country has a higher money supply, if both currencies are valued, then the currency of the high-deficit country will eventually drive the currency of the low-deficit country out of households' portfolios. Given this result, it might be in the interest of a country with lower deficits to impose a degree of capital control.

Arifovic showed that agent-based dynamics behave quite different from the above homogeneous rational expectations equilibrium analysis [13]. In her agent-based environment, the evolution of households' decision rules of savings and portfolio results in a flight away from the currency used to finance the larger of the two deficits. In the end, households hold all of their savings in the currency used to finance the lower of the deficits. Thus, the economy converges to the equilibrium in which only the low-deficit currency is valued. The currency of the country that finances the larger of the two deficits become valueless, and we have a single-currency equilibrium again.

4.4 Artificial Stock Markets

Among all applications of the agent-based approach to macroeconomic modeling, the most exciting one is the *artificial stock market*. By all standards, the stock market is qualified to be a complex adaptive system. However,

conventional financial models are not capable of demonstrating this feature. On the contrary, the famous *no-trade theorem* shows how inactive this market can be in equilibrium [146]. It was therefore invigorating when Holland and Arthur established an economics program at the Santa Fe Institute in 1988 and chose artificial stock markets as their initial research project. The SFI artificial stock market is built upon the standard asset pricing model [88, 89]. What one can possibly learn from this novel approach was well summarized in [127], which is in fact the first journal publication on an agent-based artificial stock market. A series of follow-up studies materialized the content of this new fascinating frontier in finance.

Agent Engineering and Trading Mechanisms

Agent-based artificial stock markets have two mainstays: agent engineering and institution (trading mechanism) designs. Agent engineering mainly concerns the construction of financial agents. [144] showed how to use genetic algorithms to encode trading strategies of traders. A genetic fuzzy approach to modeling trader's behavior was shown in [143], whereas the genetic neural approach was taken by [110]. To simulate the agent-based artificial stock market based on the standard asset pricing model, the AI-ECON Research Center at the National Chengchi University, Taiwan developed software known as the *AI-ECON Artificial Stock Market (AIE-ASM)*. The *AIE Artificial Stock Market* differs from the *SFI Artificial Stock Market* in the computational tool that is employed. The former applies genetic programming, while the latter has genetic algorithms. In AIE-ASM, genetic programming is used to model agents' expectations of the price and dividends. A menu-like introduction to AIE-ASM Ver. 2 can be found in [63].

In [35] and [160] we see a perfect example of bringing different learning schemes into the model. The learning schemes incorporated into [35] include empirical Bayesian traders, momentum traders, and nearest-neighbor traders, whereas those included in [160] are neural network and momentum traders. [109] gave a more thorough and general discussion of the construction of artificial financial agents. In addition to models, data is another dimension of agent engineering. What can be addressed here is the issue of stationarity that the series traders are looking at. Is the entire time series representative of the same dynamic process, or have things changed in the recent past? LeBaron studied traders who are initially heterogeneous in perception with different time horizons, which characterize their interpretation of how much of the past is relevant to the current decision making [110].

Chen and Yeh contributed to agent engineering by proposing a modified version of social learning [58]. The idea is to include a mechanism, called the *business school*. Knowledge in the business school is open for everyone. Traders can visit the business school when they are under great survival pressure. The social learning version of genetic programming is applied to model the

evolution of the business school rather than directly on traders. Doing it this way, one can avoid making an implausible assumption that trading strategies, as business secrets, are directly imitable. [161] further combined this modified social learning scheme with the conventional individual learning scheme in an integrated model. In this integrated model a more realistic description of traders' learning behavior is accomplished: the traders can choose to visit the business school (learning socially), to learn exclusively from their experience (learning individually), or both. In their experiments, based on the effectiveness of different learning schemes, traders will switch between social learning and individual learning. Allowing such a competition between these two learning styles, their experiment showed that it is the individual learning style which won the trust of the majority. To the best of our knowledge, this is the only study which leaves the choice of the two learning styles to be endogenously determined.

The second component of agent-based stock markets is the institutional design. An institutional design should answer the following five questions: (i) who can trade, (ii) when and how can orders be submitted, (iii) who may see or handle the orders, (iv) how are orders processed, and (v) how are prices eventually set. Trading institutional designs in the conventional SFI artificial stock market either follow the Walrasian 'tatonnement' scheme or the rationing scheme. This scheme describes a market operation procedure. Basically, there is an auctioneer who serves as a market coordinator. In each market period, the auctioneer announces a price to all market participants. Based on this market price, participants submit their transaction plans, for instance how much to buy or how much to sell. The auctioneer will then collect all submissions. If there is an imbalance between demand and supply, the auctioneer will then announce a new price, and the market participants will submit new plans accordingly. This process continues until the auctioneer finds a price which can equate demand to supply, and all transaction plans will be carried out with this price, also called the 'equilibrium price'. The essence of tatonnement is that no single transaction can be allowed unless the equilibrium price is found. This highly centralized trading system needs to be distinguished from other less centralized or distributed trading systems.

[35] and [160], however, considered a double auction mechanism. This design narrows the gap between artificial markets and the real market, and hence makes it possible to compare the simulation results with the behavior of real data, such as tick-by-tick data. Since stock market experiments with human subjects were also conducted within the double auction framework [139], this also facilitates conversation between the experimental stock market and the agent-based artificial stock market.

Based on agent engineering and trading mechanism designs, agent-based artificial stock markets can generate various market dynamics, including price, trading volumes, the heterogeneity and complexity of traders' behavior, and wealth distribution. Among them, price dynamics is the one under the most

intensive study. This is not surprising, because ever since the 1960s price dynamics has been the focus of studies on random walks, the efficient market hypothesis, and market rationality (the rational expectations hypothesis). With the advancement of econometrics, it further became the focus of the study of non-linear dynamics in the 1980s.

Mis-Pricing

Agent-based artificial stock markets make two important contributions to our understanding of the behavior of stock prices. First, they enable us to understand what may cause the price to deviate from rational equilibrium price or the so-called ‘fundamental value’.

Both [35] and [160] discussed the effect of momentum traders on price deviation. Yang found that the presence of momentum traders can drive the market price away from the homogeneous rational equilibrium price [160]. Chan reported a similar finding: adding momentum traders to a population of empirical Bayesian traders has an adverse impact on market performance, although price deviation decreased as time went on [35]. Empirical Bayesian basically behaves like a Bayesian, except that the posterior distribution is built upon the empirical rather upon a subjective distribution. For example, in this context, the empirical Bayesian trader forms its posterior distribution of the dividends by using the empirical distributions of both dividends and prices.

LeBaron inquired whether agents with a long-horizon perception can learn to effectively use their information to generate a relatively stable trading environment [110]. The experimental results indicated that while the simple model structure with fixed long horizon agents replicates the usual efficient market results, the route to evolving a population of short horizon agents to long horizons may be difficult. [20] and [111] found that when the speed of learning (the length of a genetic updating cycle) decreased (which forces agents to look at longer horizon features), the market approached the REE.

[47] is another study devoted to price deviation. They examined how well a population of financial agents can track the equilibrium price. By simulating the artificial stock market with different dividend processes, interest rates, risk attitudes, and market sizes, they found that the market price is not an unbiased estimator of the equilibrium price. Except in a few extremely bad cases, the market price deviates from the equilibrium price moderately from -4% to $+16\%$. The pricing errors are in fact not patternless. They are actually negatively related to market sizes: a thinner market size tends to have a larger pricing error, and a thicker market tends to have a smaller one. For the thickest market which they have simulated, the mean pricing error is only 2.17%. This figure suggests that the new classical simplification of a complex world may still provide a useful approximation if some conditions are met, such as in this case, the market size.

Complex Dynamics

As to the second contribution, agent-based artificial stock markets also enhance our understanding of several stylized features well documented in financial econometrics, such as fat tails, volatility clusters, and non-linear dependence. [111] showed that the appearance of the ARCH effect and the non-linear dependence can be related to the speed of learning. [160] found that the inclusion of momentum traders generates a lot of stylized features, such as excess volatility, excess kurtosis (leptokurtotic), lack of serial independence of return, and high trading volume.

Another interesting line is the study of emergent properties within the context of artificial stock markets. Emergence is about “how large interacting ensembles exhibit a collective behavior that is very different from anything one may have expected from simply scaling up the behavior of the individual units” ([107]: 3). Consider the efficient market hypothesis (EMH) as an example. If none of the traders believe in the EMH, then this property will not be expected to be a feature of their collective behavior. Thus, if the collective behavior of these traders indeed satisfies the EMH as tested by standard econometric procedures, then we would consider the EMH as an emergent property. As another example, consider the rational expectations hypothesis (REH). It would be an emergent property if all our traders are boundedly rational, with their collective behavior satisfying the REH as tested by econometrics.

Chen and Yeh applied a series of econometric tests to show that the EMH and the REH can be satisfied with some portions of the artificial time series [59]. However, by analyzing traders’ behavior, they showed that these aggregate results cannot be interpreted as a simple scaling-up of individual behavior. The main feature that produces the emergent results may be attributed to the use of genetic programming, which allows us to generate a very large search space. This large space can potentially support many forecasting models in capturing short-term predictability, which makes simple beliefs (such as that where the dividend is an iid (independent and identically distributed) series, or that when the price follows a random walk) difficult to be accepted by traders. In addition to preventing traders from easily accepting simple beliefs, another consequence of a huge search space is the generation of sunspot-like signals through mutually-reinforcing expectations. Traders provided with a huge search space may look for something which is originally irrelevant to price forecasts. However, there is a chance that such kinds of attempts may mutually become reinforced and validated. The generation of sunspot-like signals will then drive traders further away from accepting simple beliefs.

Using Granger causality tests, [59] found that dividends indeed can help forecast returns. By their experimental design, the dividend does not contain the information of future returns. What happens is a typical case of

mutually-supportive expectations that make the dividend eventually contain the information of future returns.

As demonstrated in [58] and [59], one of the advantages of agent-based computational economics (the bottom-up approach) is that it allows us to observe what traders are actually thinking and doing. Are they martingale believers? Are they sunspot believers? Do they believe that trading volume can help predict returns? By counting the number of traders who actually use sunspots or trading volumes to forecast returns, one can examine whether sunspot effects and the causal relation between stock returns and trading volume can be two other emergent properties [49, 62].

Market Diversity and Market Efficiency

Yeh and Chen examined another important aspect of agent engineering, this being *market size* (number of market participants) [162]. Few studies have addressed the significance of market size on the performance of agent-based artificial markets. One good exception is [26], whose simulation results showed that the simple tradable emission permit scheme (an auction scheme) can be the most effective means for pollution control when the number of participants is small. However, as the number of participants increases, its performance declines dramatically and becomes inferior to that of the uniform tax scheme. Another exception is [33]. In most studies, the number of market participants is usually determined in an arbitrary way, mainly constrained by the computational load. [10], however, justified the number of participants from the viewpoint of search efficiency. She mentioned that the minimal number of strings (agents) for an effective search is usually taken to be 30 according to the artificial intelligence literature. Nonetheless, agent-based artificial markets have different purposes and concerns.

Related to market size is *population size*. In the case of social learning (single-population GA or GP), market size is the same as population size. However, in the case of individual learning (multi-population GA or GP), population size refers to something different, namely, the number of solution candidates each trader has. Like market size, population size is also arbitrarily determined in practice.

Yeh and Chen studied the effect of market size and population size upon market efficiency and market diversity under social and individual learning styles [162]. Their experimental results can be summarized as two effects on market efficiency (price predictability), namely, the *size effect* and the *learning effect*. The size effect says that the market will become efficient when the number of traders (market size) and/or the number of models (GP trees) processed by each trader (population size) increases. The learning effect says that the price will become more efficient if traders' adaptive behavior becomes more independent and private. Taking a look at market diversity, we observe

very similar effects except for population size: market diversity does not go up with population size. These findings motivate us to search for a linkage between market diversity and market efficiency. A ‘theorem’ may go as follows: a larger market size and a more independent learning style will increase the diversity of traders’ expectations, which in turn make the market become more active (high trading volume), and hence more efficient (less predictable). Their simulation results on trading volumes also supported this ‘theorem’. They further applied this ‘theorem’ to explain why the US stock market behaves more efficiently than Taiwan’s stock market.

4.5 Market/Policy Design

One of the research areas in which agent-based computational economics and experimental economics are closely intertwined is the *double-auction market* (DA market), or the agent-based DA market. The agent-based market serves as a good starting point for applying agent-based simulation to market/policy design. One important application of agent-based computational models to market/policy design is the electricity supply market [29, 125, 126]. In this application area, we are convinced that agent engineering (learning schemes) plays a crucial role in simulating the consequences of various market designs.

By agent engineering, [73] categorized agent-based models which have been developed to characterize or understand data from human subject experiments into three classes, namely, zero intelligent (ZI) agents, reinforcement and belief learning, and evolutionary algorithms. Among the three, ZI agents were considered to be a useful benchmark or a good building block for developing more advanced agent-based models. Zero-intelligent agents are introduced by [85], which is the earliest ACE work motivated by the double-auction market experiment.²¹ However, as far as market efficiency is concerned, ZI traders are not sufficient for the market to converge to the social-welfare maximization price, or the equilibrium price. The necessary condition, therefore, requires agents to learn. Among all learning agents studied in the agent-based DA models, the simplest one is the ZI Plus (ZIP) agents, introduced by [66].

Wu and Bhattacharyya continued this line of research and studied the boundary beyond which ZIP traders may fail the market mechanism [159]. They introduced speculators into standard DA markets. They found that ZIP traders can no longer guarantee market efficiency when there is a large number of speculators, as compared to the number of normal traders. In some scenarios, the efficiency losses about 25% of the social welfare.

The purpose in studying the agent-based double auction (DA) market is to adequately equip ourselves to tackle the much more complex agent-based electricity market. [124] gave a splendid review of the well-known Electricity Market Complex Adaptive System (EMCAS) developed by the Argonne

²¹ For a survey of later developments, see [38].

National Laboratory. EMCAS is an agent-based electricity supply market model written using the Recursive Agent Simulation Toolkit (**Repast**), a special-purpose agent-based simulation tool. The research on the agent-based electricity market is motivated by the undergoing transition from centrally regulated electricity markets to decentralized markets. These transitions introduce a highly intricate web of interactions of a large number of heterogeneous companies and players, which causes the consequences of new regulatory structures largely unknown and leaves policy design in a state of high stakes.²² Given this uncertainty, agent-based models can help construct suitable laboratories that can provide ranges of possibilities and test regulatory structures before they are actually implemented. EMCAS now serves as the basis for evaluating Illinois' deregulation of the market.

Boyle presented an ambitious project on the agent-based model of the whole criminal justice system in the UK, which was funded by the Home Office in the UK [30]. The criminal justice system in England is delivered by three diverse government bodies, the Home Office, the Department of Constitutional Affairs, and the Crown Prosecution Service. Within the criminal justice system as a whole, there must be some dependencies among the functions of the three agencies. Nonetheless, the three constituents might not have been 'joined up' sufficiently well to encourage the best use of resources, and this caught the attention of the Treasury in their biennial spending review. Therefore, the purpose of this project is to build an agent-based model to help diverse operating groups engage in strategic policy making and take into account the complex interactions within the criminal justice system so as to better observe the impacts of policy.

To make the model fulfill this objective, [30] introduced a new thinking regarding agent-based models, called *the mirror function*, which is equivalent to producing a model of the whole criminal justice system in which all actors in the system acknowledge that the model was really 'them'. The work entailed gathering evidence of links between the behavior and actions of one person or group of people, and those of another, and through this making arguments for the best use of resources, while also reaching agreement between each group of people regarding all of this. This is essentially to do with encouraging a change in the style of working of these core government agencies. Boyle therefore demonstrates a very distinctive class of agent-based models, which integrates a vein of social work into model-building [30].²³

²² This is exemplified by the extremely unsatisfactory experience of California. While, according to economic theory, deregulation and free competition will lead to increased economic efficiency expressed in higher quality services and products at lower prices, the reality of today's emerging electricity markets does not fit this straightforward economic model.

²³ At present, there are very few agent-based models of this sort; [120] is the only case known to this author.

5 Pushing the Research Frontier with CI

5.1 Developments in Agent Engineering

An essential element of the agent-based modeling is *agent engineering*. Over the last decade, the progress made in modeling adaptive behavior has been particularly noticeable. There seems to have been a general tendency to *enrich* agents' adaptive behavior from several different perspectives. This enrichment has been made possible mainly due to extensive applications of *computational intelligence* to economics.

First, simple adaptive behavior has been extended to complex adaptive behavior. Initially, agents' decisions were simply characterized by *parametric* models; usually, there were just numbers over a bounded real space. [10]–[12] are typical examples (see Sect. 4 for details). All important decisions such as quantity supply, labor supply, savings, financial portfolios, and investment in human capital were characterized by *numbers* rather than *rules*. As a result, the things revealed by the adaptive processes were best viewed as a series of *number crunching* exercises. Sophisticated learning or adaptive behavior were not able to appear in these simple adaptive models.

Later on, the notion of using *rules* instead of *numbers* to characterize agents' decisions was brought in by [31]– [33], [81], and many others. These series of efforts brought about discernible progress: they formally introduced agents *which are able to forecast with rules (models)*. Nonetheless, their forecasting behavior was largely confined to *linear regression models*. This restriction was unavoidable because at that stage economists did not know much about dealing with non-parametric adaptive behavior, and linear regression models seemed to be the natural starting point. However, there is neither sound theoretic nor empirical support for the assumption that agents' adaptive behavior may be parameterized.

A breakthrough was made by [9], [53] and [112] via genetic programming (GP). The use of genetic programming not only makes agents able to engage in non-linear and non-parametric forecasting, but it also makes them able to *think* and *reason*. This last virtue is crucial because it helps us to represent a larger class of cognitive capabilities, such as making plans and strategies. This development contributes to the advancement of the agent-based models which are full of the non-trivial strategic behavior of agents, for instance, games, auctions, and financial markets. The AI-ECON Research Center in Taipei has now launched a research project – referred to as *the Functional-Modularity Foundation of Economics* – that has further enlarged the adaptive behavior to encompass preferences, commodities, technology, human capital, and organizations [41, 42].

By manipulating a set of primitives with genetic operators, one can *grow* a great variety of human cognitive processes. In principle, there is no limit to

those growing processes. It is the *survival pressure* endogenously generated via agent interactions that determines their size. In this case, neither do we need to assume that the agents follow simple rules, as the KISS (keep it simple, stupid) principle suggests, nor do we assume that they are sophisticated. Simplicity or complexity is not a matter of an *assumption* but a matter of *emergence*. For example, in a simple deterministic agent-based cobweb model, the literature shows that all surviving firms have indeed followed simple and myopic rules to forecast price. However, their behavior became more complicated when speculators were introduced into the markets. In addition, when turning to the stock market, agents' behavior could switch between simple rules and sophisticated rules.²⁴ In a nutshell, in ACE, what determines the survivability of a type of agent is not the model designers, but the *natural law* of the models; we shall see more on this in Sect. 5.2.

The second development is concerned with the *behavioral foundations* of agent engineering. While CI tools have been extensively applied to agent engineering, their ability to represent *sensible* adaptive behavior has been questioned since agent-based economic models became popular. Since 1999, a series of efforts have been made in an attempt to justify the use of genetic algorithms in agent-based modeling. However, most of these studies are mainly built upon theoretical arguments. [94] were the first to use evidence from *interviews* and *questionnaires* to justify the use of genetic algorithms in their agent-based foreign exchange markets. Their study highlights the significance of the *field study* – an approach frequently used by sociologists – to agent engineering. Duffy's agent-based model of a medium of exchange applied the data from laboratory experiments with human subjects to justify the use of *reinforcement learning* [72]. His study showed how agent-based economic models can benefit from *experimental economics*.

Another related development has occurred in the use of *natural language*. People frequently and routinely use natural language or linguistic values, such as 'high', 'low', and so on, to describe their perception, demands, expectations, and decisions. Some psychologists have argued that our ability to process information efficiently is the outcome of applying *fuzzy logic* as part of our thought process. Evidence on human reasoning and human thought processes supports the hypothesis that at least some categories of human thought are definitely fuzzy. Yet, early agent-based economic models have assumed that an agent's adaptive behavior is *crisp*. Tay and Linn made progress in this direction by using a *genetic-fuzzy classifier system* (GFCS) to model traders' adaptive behavior in an artificial stock market [143].

[143] provided a good illustration of the *non-equivalence* between the acknowledgement of the *cognitive constraint* and the assumption of *simple agents*. It is well-known that the human mind is notoriously bad at intuitively

²⁴ In plain English parlance, they sometimes regarded George Soros as their hero, while at other times they developed a great admiration for Warren Buffett.

comprehending exponential growth. However, there is no evidence that traders on Wall Street are simple-minded. Tay and Linn's work recognized the difference, and appropriately applied the GFCS to *lessen* agents' reasoning load via the use of natural language.

[72], [94], and [143] can all be regarded as a starting point for a more remarkable development in agent engineering: the CI tools employed to model agents' adaptive behavior are grounded in strong evidence within the cognitive sciences. It is at this point that agent-based modeling should have closer interactions with the *field and panel study*, *experimental economics* and *behavioral economics* (See more below in Sect. 5.3).

5.2 Distinguishing Features

While the progress made in agent engineering is evident, a more subtle issue of ACE is: "does agent-based computational economics have anything worthwhile to offer economists in general, or is it only of interest to practitioners of its own paradigm?" In this Section, we shall argue that the development of ACE has already demonstrated some distinguishing features with insightful lessons which are generally not available from neoclassical macroeconomics. The distinguishing features, which may interest economists in general, are two-fold. First, ACE helps build a *true* micro-foundation of macroeconomics by enabling us to study the *micro-macro relation*. This relation is not just a linear scaling-up, but can have a complex 'chemical' effect, known as the *emergent property*. Consequently, economics becomes a part of the *Sciences of Emergence*. Second, ACE is able to demonstrate a lively *co-evolution* process, which provides a new platform for testing economic theories. Moreover, what comes with the co-evolution process is a *novelty-generation* process. The latter is, in particular, the weakest area of neoclassical economics.

Micro-Macro Relation and Emergent Properties

Agent-based modeling provides us with a rich opportunity to study the so-called '*micro-macro relation*', which is beyond the feasibility of the neoclassical economics that consists of only a few representative agents. The first type of micro-macro study involves laying the *foundation* for the aggregate behavior upon the *agents' interacting adaptive schemes*. A series of efforts were made by [17] and [70] to attribute, in an analytical way, the appearance of some interesting macroeconomic phenomena, such as fluctuations in foreign exchange rates, the bid-ask spread, hyperinflation and economic take-off, to the adaptive behavior driven by GA. Elements, such as *self-reinforcement* and *critical mass*, upon which the conventional arguments are built, are actually encapsulated into GAs. [53], [56] and [60], on the other hand, showed the significance of the *survival-of-the-fittest principle* to the convergence to Pareto

optimality. In their agent-based cobweb model, OLG model of saving and inflation, and coordination games, it was shown that the property of converging to Pareto optimality will break down if *survival pressure* is removed.

The second type of micro-macro study is concerned with the *consistency* between the micro behavior and the macro behavior. A particularly interesting thing is that the micro behavior can sometimes be quite different from the macro behavior. Both the work done by [81] on the cobweb model and [58] and [59] on the asset pricing model showed that the time series of the market price (an aggregate variable) followed a simple stochastic process. However, there is no simple description of the population dynamics of individual behavior. The simple stochastic price behavior was, in effect, generated by a great diversity of agents whose behavior was constantly changing. [58] proposed a measure for the *complexity* of an agent's behavior and a measure of the *diversity* of an agent's complexity, and it was found that both measures can vary quite widely, regardless of the simple aggregate price behavior.

In addition, using the micro-structure data, [49], [58], [59], and [62] initiated an approach to study what is called the *emergent property*. By that definition, they found that a series of aggregate properties, such as the efficient market hypothesis, the rational expectations hypothesis, the price-volume relation and the sunspot effect, which were proved by rigorous econometric tests, were generated by a majority of agents who did not believe in these properties. Once again, our understanding of the micro behavior does not lead to a consistent prediction of the macro behavior. The latter is simply not just the linear scaling-up of the former. Conventional economics tends to defend the policy issues concerned with the individual's welfare (for instance the national annuity program), based on macroeconomic tests such as the permanent income hypothesis. Agent-based macroeconomics may invalidate this approach due to emergent properties.

Co-Evolution

Briefly, co-evolution means that everything depends on everything else. The performance of one strategy depends on the composition of the strategies with which it interacts, and the fundamental push for agents to adapt arises because other agents are adapting as well. This idea is by no means new to economists. Actually, it is the main subject of evolutionary game theory. However, what has not been shown explicitly in the evolutionary game theory or mainstream economics is that *it is the force of co-evolution which generates novelties*. We shall say a few words concerning their relation here, but more on novelty in the next Section.

Novelties-generation, from its general characteristics to its formation process, is little known in mainstream economics. For example, there is no formal (mathematical) description of how the MS-DOS system eventually led

to the MS-Windows system. Neither is there an abstract description showing how commodities A_1, A_2, \dots, A_n in the early days lead to commodities B_1, B_2, \dots, B_m at a later stage, or how a population of behavior years ago leads to a different population of behavior at present. Quite ironically, the vision of the ‘Father of Neoclassical Economics’, Alfred Marshall, namely, “Economics, like biology, deals with a matter, of which the inner nature and constitution, as well as outer form, are constantly changing,” was virtually not carried out at all by his offspring (neoclassical economists) [118].

ACE attempts to recast economics along biological and evolutionary lines. Within the co-evolutionary framework, the system which an agent faces is essentially *open* and *incomplete*. The optimal kinds of behavior or strategies which interest most economists may not necessarily exist in this system. In his agent-based cobweb model, [81] used the *survival distribution function* of firms to show *waves of evolutionary activity*. In each wave, one witnesses the sudden collapse of a strongly dominating strategy, the ‘optimal’ strategy. Very typically, the co-evolution demonstrated in the agent-based model is not a peaceful state of co-existence, but is an incessant struggle for survival where no strategy can be safe from being replaced in the near future. Novel strategies are spontaneously developed and old ‘optimal’ strategies are continually replaced.

This feature casts doubt on the ‘optimal’ economic behavior which is not derived from the agent-based co-evolutionary context. In this way, Chen and Huang’s agent-based model of investment lent support to the non-optimality of the capital asset pricing model (CAPM) [45]. The optimality of the CAPM was originally derived from a general equilibrium setting. However, they simulated an agent-based multi-asset market, and showed that, in most of their simulations, the fund managers who followed the CAPM did not survive when investors with the constant relative risk aversion presented.

In [45], the CAPM traders and many different types of traders were all introduced to the market right at the beginning (at the initialization stage). They were competing with other agents whose portfolio strategies were evolving over time and which were characterized by GA. The annihilation of the CAPM traders was the result of this setting. This kind of test is referred to as the *formula-agent* approach. Formula agents are agents whose behavior or decision rules are inspired by economic theory. Based on this approach, the economic behavior predicted by economic theory is tested by directly adding formula agents to the initial population. Doing so may be biased because the resultant co-evolution process may be determined by these initial *hints*, a common phenomenon known as *path dependence*.²⁵ Therefore, the formula-agent approach is relatively *weak* as opposed to an alternative approach to

²⁵ Path dependence is ubiquitous in ABM. For example, in Dawid’s agent-based model of double auctions, the distribution of competitive prices is sensitively dependent on the distribution of initial bids and asks [69], [70].

the co-evolution test, and Lensberg's agent-based model of investment is an illustration of this alternative [112].

Lensberg's model tested *Bayesian rational investment behavior*. However, unlike [45], [112] did not initialize the market with any *Bayesian rational investor*. In other words, all agents' investment rules were generated from scratch (by GP). It was then shown that, in later periods of evolution, what dominated the populations (the surviving firms) were the behavioral rules as if they were expected utility maximizers with Bayesian learning rules. Therefore, the Bayesian rational investment rule was validated as a behavior emerging from the bottom.

However, not all cases have lent support to what economic theory predicts. [110]'s version of the SFI (Santa Fe Institute) artificial stock market is a case in point. *Stationarity* associated with the *asymptotic theory* plays an important role in current developments in econometrics. In the mainstream rational-expectations econometrics, agents are assumed to be able to learn from this stationary environment by using the so-called *Kolmogorov-Wiener filter*. The use of this filter can make sense only if agents believe that the entire time series is stationary, and never doubt that things may have changed in the recent past. Agents with this belief are called '*long-horizon agents*' in LeBaron's ABM. In a similar way to [112], LeBaron questioned whether these long-horizon agents can eventually emerge from the evolution of a population of short-horizon agents, given that the true dividends-generation process is indeed stationary [110]. Interestingly, he found that while long-horizon agents are able to replicate usual efficient market results, evolving a population of short-horizon agents into long-horizon agents is *difficult*. This study, therefore, presents a typical coordination failure problem frequently addressed in macroeconomics.

Within this co-evolution test framework, the maximizing-expected-utility (MEU) behavior of investors, known as the *principle of maximizing certainty equivalence*, was also rejected by [34], [113], and [142]. In their agent-based models of *investment under uncertainty*, they all came up with the same conclusion: *those who survive were not the most efficient in a normative sense* – in other words, the MEU agents were not able to survive. Hence, in a sense, the equivalence between *efficiency* and *survival* broke down. What happened instead was that the surviving investors either took too much risk [113] or were too cautious [34, 142].

Novelties

As mentioned earlier, in ACE, what may come with a co-evolutionary process is a novelties-generation process. This feature is similar to Hayek's evolutionary concept of '*competition as a discovery procedure*.' The neoclassical economic models are completely silent on the novelties-generation process,

from their general characteristics to their formation process. One basically cannot anticipate anything unanticipated from the neoclassical model. All types of economic behavior are determined exogenously and can only be renewed manually by the model designers in a top-down manner. This makes it very hard for neoclassical economics to give a constructive notion of preferences, commodities, technology, human capital, and organization, concepts that are fundamentally related to the *theory of economic change*.

Back in the late 1980s, Holland and Arthur had already sketched a research idea, known as ‘*growing artificial economy*’, which was basically to simulate the evolution of an economy from its primitive state to the advanced state. This big plan, however, was never carried out. Instead, what was actually implemented was found in Epstein and Axtell’s famous book, ‘*growing artificial societies*.’ In a model of cellular automata, they evolved many interesting kinds of economic and social behavior, including trade, migration, disease, distribution of wealth, social networks, sexual reproduction, cultural processes, and combat. In addition to this major piece of work, [132] studied how money as a medium of exchange can emerge from a bartering economy, and [17] also simulated the appearance of an economic take-off (the industrial revolution).

Despite these studies, one has to say that the novelties-generation process has not been well exploited given the current state of ABM. There should be more left for the researchers to do. In their research project, the *functional-modularity foundation of economics*, [41, 42] proposed an agent-based model of preference changes and technology formation to *grow* both technology and preferences. In their model, consumers’ current preferences will determine the direction of technology advancement. However, the technology developed will in turn evolve the preferences as well. GP is applied here to give size-free and shape-free representation of technology and preferences.

The use of genetic programming in economics provides economists a great opportunity to rethink some hundred-year-old ideas. In particular, it enables economists to implement the ideas of economic evolution or progress by incorporating and hence acknowledging the importance of modularity. Simulating economic evolution with functional modularity is not restricted to technology or product innovation. More challenging tasks are its application to labor markets and organizations. While the idea that labor as capital (known as ‘human capital’), has been studied for almost 40 years, the process of accumulating human capital – and hence the role of education, as well as on-job training – is yet to be established.

5.3 Future Directions

Before ending this Section, we would like point out some directions for further research so as to see more opportunities and challenges opening for applications of computational intelligence tools.

Experimental Economics and Behavioral Economics

It becomes gradually clear that agent-based computational economics should be able to interact with experimental and behavioral economics in a more integrated framework. A series of papers published recently motivated the need for an integrated framework and sketched how this work can be done. First, the behavioral approach and the agent-based approach can collaboratively work together in a bi-directional manner. On the one hand, experimental and behavioral approaches can help answer some modeling issues related to agent engineering, while, on the other hand, agent-based computational finance can help test the robustness or the generality of some behavioral rules observed from psychological laboratory experiments.

[43] serves as an example of the first direction. While the essence of agent-based computing is agents, not so much has been said as to how to model or program these agents. Disputes still prevail on the issue like the simple/naive agents versus the sophisticated/smart agents.²⁶ A proposed solution to this problem is to work with real human behavior, in particular, when the respective fields or an experimental study are available. For example, there are already some empirical observations regarding gamblers' behavior; hence, one may get some ideas on how a gambling agent should be programmed in light of the empirical evidence. Chen and Chie's work on the agent-based modeling of lottery markets serves as a demonstration of this idea [43].

[48] serves as an example of the other direction. Psychologists have been long distinct from economists in the rational assumption of human behavior. The gap between the two has, however, been narrowed in recent years, thanks to a series of celebrated works by Tversky, Kahneman, and their followers. Findings based on a series of psychological experiments concerning decision-making under risk and uncertainty are now applied to address a number of financial anomalies, which fosters the growing field currently known as *behavioral finance*.

Chen and Liao, however, questioned the legitimacy of financial models directly built upon psychological experiments [48]. Their main concern is that psychological experiments which lend support to various cognitive biases seem to focus only on independent individual behavior in a rather static environment. This setting is, therefore, distant from financial markets, where agents are able to learn and adapt in an interactively dynamic environment. As a result, various cognitive biases observed from the psychological experiments may be corrected via learning and may not be exogenously fixed as in most behavioral financial models. [48] proposed an alternative: instead of exogenously imposing a specific kind of behavioral bias (for example overconfidence or conservatism) on the agents, we can canvass the emergence and/or

²⁶ See [50] for an in-depth discussion of this issue.

the survivorship of this behavioral bias in the highly dynamic and complex environment through computer simulations.

Second, when software agents are commonly used to replace human agents in making decisions and taking action in an era of electronic commerce, human agents and software agents can quite often be placed in a common arena and their interaction becomes more intense than ever. Questions pertaining to the consequences of this interaction, therefore, become crucial. [87] pioneered such a research direction, and raised two fundamental issues which define this line of research. First, will artificial agents in markets influence human behavior? Second, will the interaction between human and artificial agents have a positive or negative effect on the market's efficiency? They designed a continuous double auction market in the style of the Iowa electronic market, and introduced software agents with a passive arbitrage seeking strategy to the market experiment with human agents. Whether or not the human agents are well informed of the presence of the software agents can have significant impacts upon market efficiency (in the form of price deviations from the fundamental price). They found that if human agents are well informed, then the presence of software agents triggers more efficient market prices when compared to the baseline treatment without software agents. Otherwise, the introduction of software agents results in lower market efficiency.²⁷

Agent-Based Econometric Modeling

We now have seen some progress regarding how agent-based models can be built upon laboratory experiments with human subjects, field studies, and social work, but not directly with the data themselves. This is concerned with agent-based econometric models. The complexity of the agent-based models makes their empirical estimation a daunting task, if not an impossible one. Therefore, few attempts have been made to conduct an econometric analysis of an agent-based model. However, recently, we have started to see some progress in the estimation of some relatively simple agent-based models; [122] was one of the pioneering efforts.

[122] can be regarded as an outcome of the new research trend that embeds conventional discrete choice models, also known as the qualitative response models, in a social network, and examines the impact of the social interaction upon individuals' discrete choices. Other similar works can be found in [28] and [74]. With moderate degrees of simplifying assumptions on individuals' decision models as well as interaction mechanisms, this network-based agent-based model can be parameterized and estimated as an econometric model. This is basically what was done in [122], which estimated the interaction mechanism among young people in relation to smoking behavior by using the

²⁷ These two issues have been further pursued in the recent development of the U-Mart platform [103, 133, 145].

result of [8].²⁸ Its empirical results strongly support the presence of positive peer effects in smoking behavior among young people.

Certainly, not all agent-based econometric models are network-based. There are a series of agent-based financial econometric models which do not explicitly refer to a network or a graph [4, 5, 155].

Agent-Based Social Networks

Having noticed that agent-based econometric models were first successfully developed in the area of the network-based discrete choice models, we noticed that the social network plays an increasingly important role in ACE models. In fact, the network should be an essential ingredient of agent-based models, while most agent-based simulation models do not explicitly include this element.

Network externalities may be viewed as one of the best places to see the use of agent-based social networks. The celebrated work [97] was demonstrated in an agent-based manner by [148], who built an agent-based model and evaluated it by verifying the simulation results with conventional Beta and VHS systems. [83] enhances our understanding of the significance of network effects by creating agent-based computational simulations of such markets. Insights into the dominance of the inferior technologies are further explored within a model called ‘Standard-Scape’.

6 Concluding Remarks

Unlike most tutorials on the economic and financial applications of computational intelligence, this Chapter is not about how CI tools are applied to economics and finance as merely an optimization numerical tool. Instead, we have a broader scope, namely to use CI tools to build economic agents with some reasonable and realistic degree of autonomy, and then study the emergent phenomena resulting from a society of these autonomous agents. In this sense, CI is introduced to economics as an algorithmic foundation of autonomous agents. We review two major algorithmic foundations, namely, neural networks and evolutionary computation. While the review is not exhaustive, the essential idea of using other tools to build autonomous agents and hence evolve the economy is largely the same. A well-motivated reader should be able to see room for other alternative algorithmic foundations, such

²⁸ [8] can be read as one of the earliest available econometric results of agent-based models. Given the profile of individual attributes and the social interaction mechanism, [8] provides an analytical solution for the equilibrium distribution of the collection of individuals’ behavior. Hence, it is possible to describe the macro equilibrium from the micro level.

as fuzzy logic, decision trees, Bayesian networks, and reinforcement learning. The general question left for further study is how these different CI tools, under what specific environment, can successfully characterize human decision-making process.

In the second part of this Chapter, we reviewed the early applications of CI to agent-based computational economics. We saw how CI can help relax the stringent assumptions frequently used in old-fashion economics, and bring back some missing processes due to the learning or bounded rational behavior of agents. While making economic predictions using ACE models is still difficult, the use of CI certainly enhance some of our flexibility to simulate possible futures. In this review, we have witnessed how CI can help build more realistic ACE models such that they can be useful in policy design.

The nature of economics is change and evolution, and what makes it change and evolve is humans. CI provides alternative hypotheses or modeling of the microscopic details of human behavior. So long as these details are not trivial, CI can help economists to establish quality models as illustrated in the many works reviewed in this Chapter.

Acknowledgements

The author is grateful to one of the anonymous referees for their helpful suggestions. The author is also grateful to Professor John Fulcher for his painstaking efforts made in editing the Chapter. Research support in the form of NSC grant No. NSC. 95-2415-H-004-002-MY3 is gratefully acknowledged.

References

1. Adcock A, Thangavel1 A, Whitfield-Gabrieli S, Knutson B, Gabrieli J (2006) Reward-motivated learning: Mesolimbic activation precedes memory formation. *Neuron*, 50(3): 507–517.
2. Aha D (1997) *Lazy Learning*. Kluwer, Boston, MA.
3. Aha D, Kibler D, Marc K (1991) Instance-based learning algorithms. *Machine Learning*, 6(1): 37–66.
4. Alfarano S, Lux T, Wagner F (2005) Estimation of agent-based models: the case of an asymmetric herding model. *Computational Economics*, 26(1): 19–49.
5. Alfarano S, Lux T, Wagner F (2007) Empirical validation of stochastic models of interacting agents: a ‘maximally skewed’ noise trader model. *European Physics J. B*, 55: 183–187.
6. Allen F, Karjalainen R (1999) Using genetic algorithms to find technical trading rules. *J. Financial Economics*, 51(2): 245–71.
7. Alvarez-Diaz M, Alvarez A (2005) Genetic multi-model composite forecast for nonlinear prediction of exchange rates. *Empirical Economics*, 30: 643–663.
8. Amemiya T (1975), Qualitative response models. *Annals Economics and Social Management*, 4: 363–372.

9. Andrew M, Prager R (1994) Genetic programming for the acquisition of double auction market strategies. In: Kinnear K Jr. (ed.), *Advances in Genetic Programming*. MIT Press, Cambridge, MA: 355–368.
10. Arifovic J (1994) Genetic algorithms learning and the cobweb model. *J. Economic Dynamics and Control*, 18(1): 3–28.
11. Arifovic J (1995) Genetic algorithms and inflationary economies. *J. Monetary Economics*, 36(1): 219–43.
12. Arifovic J (1996) The behavior of the exchange rate in the genetic algorithm and experimental economies. *J. Political Economy*, 104(3): 510–541.
13. Arifovic J (2001) Evolutionary dynamics of currency substitution. *J. Economic Dynamics and Control*, 25: 395–417.
14. Arifovic J (2002) Exchange rate volatility in the artificial foreign exchange market. In: Chen S-H (ed.) *Evolutionary Computation in Economics and Finance*. Physica-Verlag, Berlin: 125–136.
15. Arifovic J, Eaton B (1995) Coordination via genetic learning. *Computational Economics*, 8(3): 181–203.
16. Arifovic J, Gencay R (2000) Statistical properties of genetic learning in a model of exchange rate. *J. Economic Dynamics and Control*, 24: 981–1005.
17. Arifovic J, Bullard J, Duffy J (1997) The transition from stagnation to growth: an adaptive learning approach. *J. Economic Growth*, 2(2): 185–209.
18. Armano G, Murru A, Marchesi M (2002) NXCS – A hybrid approach to stock indexes forecasting. In: Chen, S-H (ed.) *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer, Boston, MA.
19. Arthur B (1992) On learning and adaptation in the economy. *SFI Economics Research Program*, 92-07-038.
20. Arthur W, Holland J, LeBaron B, Palmer R, Tayler P (1997) Asset pricing under endogenous expectations in an artificial stock market. In: Arthur W, Durlauf S, Lane D (eds.) *The Economy as an Evolving Complex System II*. Addison-Wesley, Reading, MA: 15–44.
21. Axelrod R (1997) Advancing the art of simulation in the social sciences. In: Conte R, Hegselmann R, Terna P (eds.) *Simulating Social Phenomena*. Springer-Verlag, Berlin: 21–40.
22. Azariadis C, Guesnerie R (1986) Sunspots and cycle. *Review Economic Studies* LIII: 725–737.
23. Azoff M (1994) *Neural Network Time Series: Forecasting of Financial Markets*. Wiley, New York, NY.
24. Baestaens D, Van Den Bergh W, Wood D (1994) *Neural Network Solutions for Trading in Financial Markets*. Pitman, London, UK.
25. Bauer R. Jr (1994) *Genetic Algorithms and Investment Strategies*. Wiley, New York, NY.
26. Bell R, Beare S (2002) Emulating trade in emissions permits: An application of genetic algorithms. In: Chen S-H (ed.) *Evolutionary Computation in Economics and Finance*. Physica-Verlag, Heidelberg, Germany: 161–175.
27. Birchenhall C, Lin J-S (2002) Learning and convergence to Pareto optimality. In: Chen S-H. (ed.) *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer, Boston, MA: 419–440.
28. Bonabeau E (2003) Econometrics of agent-based models. *Proc. 2nd Lake Arrowhead Conf. Human Complex Systems (Keynote Speech)*, 19–22 March, Lake Arrowhead, CA.

29. Bower J, Bunn D (2001) Experimental analysis of the efficiency of uniform-price versus discriminatory auctions in the England and Wales electricity market. *J. Economic Dynamics and Control*, 25: 561–592.
30. Boyle S, Guerin S, Kunkle D (2005) An application of multi-agent simulation to policy appraisal in the criminal justice system. In: Chen S-H, Jain LC, Tai C-C (eds.) *Computational Economics: A Perspective from Computational Intelligence*. Idea Group, Hershey, PA: 228–234.
31. Bullard J, Duffy J (1998) A model of learning and emulation with artificial adaptive agents. *J. Economic Dynamics and Control*, 22: 179–207.
32. Bullard J, Duffy J (1998) Learning and the stability of cycles. *Macroeconomic Dynamics*, 2(1): 22–48.
33. Bullard J, Duffy J (1999) Using genetic algorithms to model the evolution of heterogeneous beliefs. *Computational Economics*, 13(1): 41–60.
34. Cacho O, Simmons P (1999) A genetic algorithm approach to farm investment. *Australian J. Agricultural and Resource Economics*, 43(3): 305–322.
35. Chan N, LeBaron B, Lo A, Poggio T (1999). Agent-based models of financial markets: a comparison with experimental markets. *Unpublished Working Paper*, MIT Artificial Markets Project, MIT, MA.
36. Chen J, Xu D (1998) An economic forecasting system based on recurrent neural networks. In: *Proc. IEEE Intl. Conf. Systems, Man, and Cybernetics*, 14–18 October, San Diego, CA. IEEE Press, New York, NY, 2: 1762–1767.
37. Chen S-H (1997) On the artificial life of the general economic system (I): the role of selection pressure. In: Hara F, Yoshida K (eds.) *Proc. Intl. Symp. System Life*, 21–22 July, Tokyo, Japan: 233–240.
38. Chen S-H (2000) Toward an agent-based computational modeling of bargaining strategies in double auction markets with genetic programming. In: Leung K-S, Chan L-W, Meng H (eds.) *Intelligent Data Engineering and Automated Learning – IDEAL 2000: Data Mining, Financial Engineering, and Intelligent Agents*, Lecture Notes in Computer Science 1983. Springer-Verlag, Berlin: 517–531.
39. Chen S-H (ed.) (2002) *Evolutionary Computation in Economics and Finance*. Physica-Verlag, Berlin.
40. Chen S-H (ed.) (2002) *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer, Boston, MA.
41. Chen S-H, Chie B-T (2004) Agent-based economic modeling of the evolution of technology: the relevance of functional modularity and genetic programming. *Intl. J. Modern Physics B*, 18(17–19): 2376–2386.
42. Chen S-H, Chie B-T (2005) A functional modularity approach to agent-based modeling of the evolution of technology. In: Namatame A, Kaizouji T, Aruka Y (eds.) *The Complex Networks of Economic Interactions: Essays in Agent-Based Economics and Econophysics*, Lecture Notes in Economics and Mathematical Systems 567, Springer-Verlag, 165–178.
43. Chen S-H, Chie B-T (2007) Lottery markets design, micro-structure, and macro-behavior: An ACE approach. *J. Economic Behavior and Organization*, (in press).
44. Chen S-H, He H (2003) Searching financial patterns with self-organizing maps. In: Chen S-H, Wang P (eds.) *Computational Intelligence in Economics and Finance*. Springer-Verlag, Berlin: 203–216.

45. Chen S-H, Huang Y-C (2007) Risk preference, forecasting accuracy and survival dynamics: simulations based on a multi-asset agent-based artificial stock market. *J. Economic Behavior and Organization*. (in press).
46. Chen S-H, Kuo T-W (1999) Towards an agent-based foundation of financial econometrics: an approach based on genetic-programming artificial markets. In: Banzhaf W, Daida J, Eiben A, Garzon M, Honavar V, Jakiela M, Smith R (eds.) *Proc. Genetic and Evolutionary Computation Conf.*, Morgan Kaufmann, San Mateo, CA, 2: 966–973.
47. Chen S-H, Liao C-C (2002) Price discovery in agent-based computational modeling of artificial stock markets. In: Chen S-H (ed.) *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer, Boston, MA: 333–354.
48. Chen, S-H, Liao C-C (2004) Behavior finance and agent-based computational finance: toward an integrating framework. *J. Management and Economics*, 8.
49. Chen S-H, Liao C-C (2005) Agent-based computational modeling of the stock price-volume relation. *Information Sciences*, 170: 75–100.
50. Chen S-H, Tai C-C (2006) On the selection of adaptive algorithms in ABM: a computational-equivalence approach. *Computational Economics*, 28(1): 51–69.
51. Chen S-H, Wang P (eds.) (2003) *Computational Intelligence in Economics and Finance*. Springer-Verlag, Berlin.
52. Chen S-H, Yeh C-H (1996) Genetic programming and the efficient market hypothesis. In: Koza J, Goldberg D, Fogel D, Riolo R (eds.) *Genetic programming 1996: Proc. 1st Annual Conf.* MIT Press, Cambridge, MA: 45–53.
53. Chen S-H, Yeh C-H (1996) Genetic programming learning and the cobweb model. In: Angeline P (ed.) *Advances in Genetic Programming 2*. MIT Press, Cambridge, MA: 443–466.
54. Chen S-H, Yeh C-H (1997) Toward a computable approach to the efficient market hypothesis: an application of genetic programming. *J. Economic Dynamics and Control*, 21: 1043–1063.
55. Chen S-H, Yeh C-H (1997) Modeling speculators with genetic programming. In: Angeline P, Reynolds R, McDonnell J, Eberhart R (eds.) *Evolutionary Programming VI*, Lecture Notes in Computer Science 1213. Springer-Verlag, Berlin: 137–147.
56. Chen S-H, Yeh C-H (1999) Modeling the expectations of inflation in the OLG model with genetic programming. *Soft Computing*, 3(2): 53–62.
57. Chen S-H, Yeh C-H (2000) Simulating economic transition processes by genetic programming. *Annals Operation Research*, 97: 265–286.
58. Chen S-H, Yeh C-H (2001) Evolving traders and the business school with genetic programming: a new architecture of the agent-based artificial stock market. *J. Economic Dynamics and Control*, 25: 363–393.
59. Chen S-H, Yeh C-H (2002) On the emergent properties of artificial stock markets: the efficient market hypothesis and the rational expectations hypothesis. *J. Economic Behavior and Organization*, 49(2): 217–239.
60. Chen S-H, Duffy J, Yeh C-H (2001) Equilibrium selection via adaptation: using genetic programming to model learning in a co-ordination game. *Electronic J. Evolutionary Modeling and Economic Dynamics*. 1: 1002.
61. Chen, S-H, Lee W-C, Yeh C-H (1999) Hedging derivative securities with genetic programming. *Intl. J. Intelligent Systems in Accounting, Finance and Management*, 8(4): 237–251.

62. Chen S-H, Liao C-C, Chou P-J (2007) On the plausibility of sunspot equilibria: simulations based on agent-based artificial stock markets. *J. Economic Interaction and Coordination*, (in press).
63. Chen S-H, Yeh C-H, Liao C-C (2002) On AIE-ASM: Software to simulate artificial stock markets with genetic programming. In: Chen S-H (ed.) *Evolutionary Computation in Economics and Finance*. Physica-Verlag, Heidelberg, Germany: 107–122.
64. Chen T, Chen H (1995) Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans. Neural Networks*, 6: 911–917.
65. Chidambaran N, Lee C, Trigueros J (2000) Option pricing via genetic programming. In: Abu-Mostafa Y, LeBaron B, Lo A, Weigend A (eds.) *Computational Finance – Proc. 6th Intl. Conf.* MIT Press, Cambridge, MA: 583–598.
66. Cliff D, Bruten J (1997) Minimal-intelligence agents for bargaining behaviors in market-based environments. *Technical Report 97-91*, Hewlett-Packard Lab.
67. Cristianini N, Shawe-Taylor J (2000) *An Introduction to Support Vector Machines – and Other Kernel-Based Learning Methods*. Cambridge University Press, UK.
68. Das N (2003) Hedge fund classification using k-means method. *EconPapers*, 204: 284.
69. Dawid H (1996) Learning of cycles and sunspot equilibria by genetic algorithms. *J. Evolutionary Economics*, 6(4): 361–373.
70. Dawid H (1999) On the convergence of genetic learning in a double auction market. *J. Economic Dynamics and Control*, 23: 1544–1567.
71. Deboeck G, Kohonen T (1998) *Visual Explorations in Finance with Self-Organizing Maps*. Springer-Verlag, Berlin.
72. Duffy J (2001) Learning to speculate: experiments with artificial and real agents. *J. Economic Dynamics and Control*, 25: 295–319.
73. Duffy J (2006) Agent-based models and human-subject experiments. In: Tesfatsion L, Judd K (eds.) *Handbook of Computational Economics 2*. North Holland, Amsterdam, The Netherlands.
74. Dugundji E, Gulyas L (2003) Empirical estimation and multi-agent-based simulation of a discrete choice model with network interaction effects. In: Macal C, North M, Sallach D (eds.) *Proc. Agent 2003 Conf. on Challenges in Social Simulation*, 2–4 October, University of Chicago. Argonne National Laboratory, Chichago, IL: 437–453.
75. Eberhart R, Simpson P, Dobbins R (1996) *Computational Intelligence PC Tools*. Academic Press, Boston, MA.
76. Elman J (1990) Finding structure in time. *Cognitive Science*, 14: 179–211.
77. Episcopos A, Davis J (1996). Predicting returns on Canadian exchange rates with artificial neural networks and EGARCHM-M model. *Neural Computing and Applications*, 4:168–174.
78. Fernández-Rodríguez F, Sosvilla-Rivero S, Andrada-Félix J (2003) Nearest-neighbour predictions in foreign exchange markets. In: Chen S-H, Wang P (eds.) *Computational Intelligence in Economics and Finance*. Springer-Verlag, Berlin: 297–325.
79. Fogel D (1995) *Evolutionary Computation – Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway NJ.
80. Fogel L (1964) On the Organization of Intellect. *PhD Thesis*, University of California at Los Angeles, CA.

81. Franke R (1998) Coevolution and stable adjustments in the cobweb model. *J. Evolutionary Economics*, 8(4): 383–406.
82. Fogel L, Owens A, Walsh M (1966) *Artificial Intelligence through Simulated Evolution*. Wiley, New York, NY.
83. Frels J, Heisler D, Reggia J (2003) Standard-scape: An agent-based model of adoption with incomplete information and network externalities. In: Chen K, Chen S-H, Cheng H-D, Chiu D, Das S, Duro R, Jiang Z, Kasabov N, Kerre E, Leong H-V, Li Q, Lu M, Romay MG, Ventura D, Wang P-P, Wu J (eds.) *Proc. 7th Joint Conf. Information Sciences*, 26–30 September, Cary, NC: 1219–1222.
84. Gately E (1996) *Neural Networks for Financial Forecasting*. Wiley, New York, NY.
85. Gode D, Sunder S (1993) Allocative efficiency of markets with zero-intelligence traders: market as a partial substitute for individual rationality. *J. Political Economy*, 101(1): 119–137.
86. Grandmont J-M (1985) On endogeneous competitive business cycles. *Econometrica*, 53: 995–1045.
87. Grossklags J, Schmidt C (2006) Software agents and market (in)efficiency: a human trader experiment. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 36(1): 56–67.
88. Grossman S (1976) On the efficiency of competitive stock markets where traders have diverse information. *J. Finance*, 31: 573–585.
89. Grossman S, Stiglitz J (1980) On the impossibility of informationally efficient markets. *American Economic Review*, 70: 393–408.
90. Hann T, Steurer E (1996). Much ado about nothing? Exchange rate forecasting: Neural networks vs. linear models using monthly and weekly data. *Neurocomputing*, 10: 323–339.
91. Holland J, Holyoak K, Nisbett R (1986) *Induction: Processes of inference, learning and discovery (computational models of cognition and perception)*. MIT Press, Cambridge, MA.
92. Holland J, Miller J (1991) Artificial adaptive agents in economic theory. *American Economic Review*, 81(2): 365–370.
93. Hollans H, Munneke H. (2003) Housing markets and house price appreciation: An Interacity Analysis. *Working paper*. University of Georgia.
94. Izumi K, Ueda K (1999) Analysis of dealers' processing financial news based on an artificial market approach. *J. Computational Intelligence in Finance*, 7: 23–33.
95. Jordan M (1986) Serial order: A parallel distributed processing approach. *Technical Report 8604*. Institute for Cognitive Science, University of California.
96. Kareken J, Wallace N (1981) On the indeterminacy of equilibrium exchange rate. *Quarterly J. Economics*, 96: 207–222.
97. Katz M, Shapiro C (1985) Network externalities, competition, and compatibility. *American Economic Review*, 75: 424–440.
98. Keber C (1999) Genetically derived approximations for determining the implied volatility. *OR Spektrum*, 21: 205–238.
99. Keber C (2000) Option valuation with the genetic programming approach. In: Abu-Mostafa Y, LeBaron B, Lo A, Weigend A (eds.) *Computational Finance – Proc. 6th Intl. Conf.* MIT Press, Cambridge MA.
100. Kiyotaki N, Wright R (1989) On money as a medium of exchange. *J. Political Economy*, 97: 927–954.

101. Kohonen T (1982) Self-organized foundation of topologically correct feature maps. *Biological Cybernetics*, 43: 59–69.
102. Kohonen T (1995) *Self-Organizing Maps*. Springer-Verlag, Berlin.
103. Koyama Y, Sato H, Matusi H, Nakajima Y (2005) Report on UMIE 2004 and summary of U-Mart experiments based on the classification of submitted machine agents. In: Terano T, Kita H, Kaneda T, Arai K, Deghchi H (eds.) *Agent-Based Simulation: From Modeling Methodologies to Real-World Applications*, (Springer Series on Agent-Based Social Systems 1), Springer-Verlag, Tokyo: 158–166.
104. Koza J (1992) *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA.
105. Koza J (1992) A Genetic approach to econometric modeling. In: Bourgin P, Walliser B (eds.) *Economics and Cognitive Science*. Pergamon Press, Oxford, UK: 57–75.
106. Kramer M (1990) Nonlinear principal analysis using autoassociative neural networks. *AIChE J.*, 37(2): 233–243.
107. Krugman P (1996) *The Self-Organizing Economy*. Blackwell, Cambridge, MA.
108. Kuan C-M, Liu T (1995) Forecasting exchange rates using feedforward and recurrent neural networks. *J. Applied Econometrics*, 10: 347–364.
109. LeBaron B (1999) Building financial markets with artificial agents: Desired goals and present techniques. In: Karakoulas G (ed.) *Computational Markets*. MIT Press, Cambridge, MA.
110. LeBaron B (2001) Evolution and time horizons in an agent based stock market. *Macroeconomic Dynamics*, 5: 225–254.
111. LeBaron B, Arthur W, Palmer R (1999) Time series properties of an artificial stock market. *J. Economic Dynamics and Control*, 23: 1487–1516.
112. Lensberg T (1999) Investment behavior under Knightian uncertainty – an evolutionary approach. *J. Economic Dynamics and Control*, 23: 1587–1604.
113. Lettau M (1997) Explaining the facts with adaptive agents: the case of mutual fund flows. *J. Economic Dynamics and Control*, 21(7): 1117–1147.
114. Lucas R (1986) Adaptive behavior and economic theory. In: Hogarth R, Reder M (eds.) *Rational choice: The contrast between economics and psychology*. University of Chicago Press, IL: 217–242.
115. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: LeCam LM, Neyman N (eds.) *Proc. 5th Berkeley Symp. Mathematical Statistics and Probability*. University of California Press, Berkeley, CA, 1: 281–297.
116. Mandic D, Chambers J (2001). *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures, and Stability*. Wiley, New York, NY.
117. Marimon R, McGrattan E, Sargent T (1990) Money as Medium of Exchange in an Economy with Artificially Intelligent Agents. *J. Economic Dynamics and Control*, 14: 329–373.
118. Marshall A (1961) *Principles of Economics (9th (variorum) ed., with annotations by CW Guillebaud)*. Macmillan, London, UK.
119. McNelis P (2005). *Neural Networks in Finance: Gaining Predictive Edge in the Market*. Elsevier, Burlington, MA.
120. Midgley G (2005) Systemic intervention for community involvement in complex policy. In: *Proc. 1st Intl. Workshop Complexity and Policy Analysis*, 22–24 June, Cork, Ireland.

121. Muth J (1961) Rational expectations and the theory of price movements. *Econometrics*, 29: 315–335.
122. Nakajima R (2003) Measuring peer effects in youth smoking behavior. In: Chen K, Chen S-H, Cheng H-D, Chiu D, Das S, Duro R, Jiang Z, Kasabov N, Kerre E, Leong H-V, Li Q, Lu M, Romay MG, Ventura D, Wang P-P, Wu J (eds.) *Proc. 7th Joint Conf. Information Sciences*, 26–30 September, Cary, NC: 1206–1210.
123. Neely C, Weller P, Ditmar R (1997) Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *J. Financial and Quantitative Analysis*, 32(4): 405–427.
124. North M (2003) Applying computational Intelligence to economic policy. In: Chen K, Chen S-H, Cheng H-D, Chiu D, Das S, Duro R, Jiang Z, Kasabov N, Kerre E, Leong H-V, Li Q, Lu M, Romay MG, Ventura D, Wang P-P, Wu J (eds.) *Proc. 7th Joint Conf. Information Sciences*, 26–30 September, Cary, NC: 1231–1234.
125. Nicolaisen J, Smith M, Petrov V, Tesfatsion L (2000) Concentration and capacity effects on electricity market power. In: Alzala A (ed.) *Proc. 2000 Congress Evolutionary Computation*, IEEE Society Press, Piscataway, NJ: 1041–1047.
126. Nicolaisen J, Petrov V, Tesfatsion L (2001) Market power and efficiency in a computational electricity market with discriminatory double-auction pricing. *IEEE Trans. Evolutionary Computation*, 5(5): 504–523.
127. Palmer R, Arthur W, Holland J, LeBaron B, and Taylor P (1994). Artificial economic life: a simple model of a stock market. *Physica D*, 75: 264–274.
128. Qian Y (2006) k-means algorithm and its application for clustering companies listed in Zhejiang province. *WIT Trans Information and Communication Technologies*, 37: 35–44.
129. Rechenberg I (1965) Cybernetic Solution Path of an Experimental Problem. *Royal Aircraft Establishment*, Library Translation 1122, August.
130. Refenes A (1995) *Neural Networks in the Capital Markets*. Wiley, New York, NY.
131. Refenes A, Zaprana A (1999) *Principles of Neural Model Identification, Selection and Adequacy: With Applications in Financial Econometrics*. Springer-Verlag, Berlin.
132. Sargent T (1993) *Bounded Rationality in Macroeconomics*. Oxford University Press, UK.
133. Sato H, Kawachi S, Namatame A (2003) The statistical properties of price fluctuations by computer agents in a U-Mart virtual future market simulator. In: Terano T, Dehuchi H, Takadama K (eds.) *Meeting the Challenge of Social Problems via Agent-Based Simulation*. Springer-Verlag, Tokyo: 67–76.
134. Schwefel H (1965) Kybernetische Evolution als Strategien der Experimentellen Forschung in der Strömungstechnik. *Diploma Thesis*, Technical University of Berlin.
135. Schwefel H (1995) *Evolution and Optimum Seeking*. Wiley, New York, NY.
136. Shadbolt J, Taylor J (2002) *Neural Networks and the Financial Markets—Predicting, Combining, and Portfolio Optimisation*. Springer-Verlag, Berlin.
137. Shi S, Xu L, Liu B (1999) Improving the accuracy of nonlinear combined forecasting using neural networks. *Expert Systems with Applications*, 16: 49–54.
138. Simon HA (1997) *Models of Bounded Rationality, Vol. 3*. MIT Press, Cambridge, MA.

139. Smith V, Suchanek G, Williams A (1988) Bubbles, crashes, and endogenous expectations in experimental spot asset markets. *Econometrica*, 56(6): 1119–1152.
140. Suykens J, Vandewalle J (1998) The K.U. Leuven time series prediction competition. In: Suykens J, Vandewalle J (eds.) *Nonlinear Modeling: Advanced Black-Box Techniques*. Kluwer, Boston, MA: 241–253.
141. Szpiro G (1997) Forecasting chaotic time series with genetic algorithms. *Physical Review E*, 55: 2557–2568.
142. Szpiro G (1997) The emergence of risk aversion. *Complexity*, 2: 31–39.
143. Tay N, Linn S (2001) Fuzzy inductive reasoning, expectation formation and the behavior of security prices. *J. Economic Dynamics and Control*, 25: 321–361.
144. Tayler P (1995) Modeling artificial stock markets using genetic algorithms. In: Goonatilake S, Treleaven P (eds.) *Intelligent Systems for Finance and Business*. Wiley, New York, NY: 271–287.
145. Terano T, Shiozawa Y, Deguchi H, Kita H, Matsui H, Sato H, Ono I, Kakajima Y (2003), U-Mart: An artificial market testbed for economics and multiagent systems. In: Terano T, Dehuchi H, Takadama K (eds.) *Meeting the Challenge of Social Problems via Agent-Based Simulation*. Springer-Verlag, Tokyo: 53–66.
146. Tirole J. (1982) On the possibility of speculation under rational expectations. *Econometrica*, 50: 1163–1182.
147. Trippi R, Turban E. (1993) *Neural Networks in Finance and Investing*. Irwin
148. Tsuji M, Kawamura H, Ohuchi A (2003) Measuring the effect of indirect network externality in VCR standardization process. In: Chen K, Chen S-H, Cheng H-D, Chiu D, Das S, Duro R, Jiang Z, Kasabov N, Kerre E, Leong H-V, Li Q, Lu M, Romay MG, Ventura D, Wang P-P, Wu J (eds.) *Proc. 7th Joint Conf. Information Sciences*, 26–30 September, Cary, NC: 1215–1218.
149. Vapnik V (1998) *Statistical Learning Theory*. Wiley, New York, NY.
150. Vapnik V (1998) The support vector method of function estimation. In: Suykens J, Vandewalle J (eds.) *Nonlinear Modeling: Advanced Black-Box Techniques*. Kluwer, Boston, MA: 55–85.
151. Vriend N (2001) On two types of GA-Learning. In: Chen S-H (ed.) *Evolutionary Computation in Economics and Finance*. Physica-Verlag, Heidelberg, Germany: 233–243.
152. Waldrop M (1992) *Complexity: The Emerging Science at the Edge of Order and Chaos*. Simon and Schuster, New York, NY.
153. Wei W-X, Jiang Z-H (1995) Artificial neural network forecasting model for exchange rate and empirical analysis. *Forecasting*, 2: 67–69.
154. Weigend A, Huberman B, Rumelhart D (1992) Predicting sunspots and exchange rates with connectionist networks. In: Casdagli M, Eubank S (eds.) *Nonlinear Modeling and Forecasting*. Addison-Wesley, Reading, MA: 395–432.
155. Westerhoff F, Reitz S (2003) Nonlinearities and cyclical behavior: The role of chartists and fundamentalists. *Studies in Nonlinear Dynamics and Econometrics*, 7(4): Article 3.
156. White H (1988) Economic prediction using neural networks: The case of IBM daily stock returns. In: *Proc. IEEE Intl. Conf. Neural Networks 2*, 24–27 July, San Diego, CA. IEEE Press, New York, NY: 451–458.
157. White H (1992) *Artificial Neural Networks—Approximation Theory and Learning Theory*. Blackwell, Cambridge, MA.

158. Wu B (1995) Model-free forecasting for nonlinear time series (with application to exchange rates). *Computational Statistics and Data Analysis*, 19: 433–459.
159. Wu S, Bhattacharyya S (2005) Minimal intelligence Agents in double auction markets with speculators. In: Chen S-H, Jain LC, Tai C-C (eds.) *Computational Economics: A Perspective from Computational Intelligence*. Idea Group, Hershey, PA: Chapter IV.
160. Yang J. (2002) The efficiency of an artificial double auction stock market with neural learning agents. In: Chen S-H. (ed.) *Evolutionary Computation in Economics and Finance*. Physica-Verlag, Heidelberg, Germany: 87–107.
161. Yeh C-H, Chen S-H (2001) Toward an integration of social learning and individual learning in agent-based computational stock markets: The approach based on population genetic programming. *J. Management and Economics*, 5.
162. Yeh C-H, Chen S-H (2001) Market diversity and market efficiency: The approach based on genetic programming. *J. Artificial Simulation of Adaptive Behavior*, 1(1): 147–167.
163. Zirilli J (1996) *Financial Prediction Using Neural Networks*. Thomson, London, UK.

Resources

1 Key Books

Arthur W, Holland J, LeBaron B, Palmer R, Tayler P (1997) Asset pricing under endogenous expectations in an artificial stock market. In: Arthur W, Durlauf S, Lane D (eds.) *The Economy as an Evolving Complex System II*. Addison-Wesley, Reading, MA: 15–44.

Axelrod R (1997) Advancing the art of simulation in the social sciences. In: Conte R, Hegselmann R, Terna P (eds.) *Simulating Social Phenomena*. Springer-Verlag, Berlin: 21–40.

Chen S-H (ed.) (2002) *Evolutionary Computation in Economics and Finance*. Physica-Verlag, Berlin.

Chen S-H (ed.) (2002) *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer, Boston, MA.

Fogel L, Owens A, Walsh M (1966) *Artificial Intelligence through Simulated Evolution*. Wiley, New York, NY.

Koza J (1992) A Genetic approach to econometric modeling. In: Bourguine P, Walliser B (eds.) *Economics and Cognitive Science*. Pergamon Press, Oxford, UK: 57–75.

LeBaron B (1999) Building financial markets with artificial agents: Desired goals and present techniques. In: Karakoulas G (ed.) *Computational Markets*. MIT Press, Cambridge, MA.

Lucas R (1986) Adaptive behavior and economic theory. In: Hogarth R, Reder M (eds.) *Rational choice: The contrast between economics and psychology*. University of Chicago Press, IL: 217–242.

2 Key Survey/Review Articles

Arifovic J (1994) Genetic algorithms learning and the cobweb model. *J. Economic Dynamics and Control*, 18(1): 3–28.

Bullard J, Duffy J (1998) A model of learning and emulation with artificial adaptive agents. *J. Economic Dynamics and Control*, 22: 179–207.

Holland J, Miller J (1991) Artificial adaptive agents in economic theory. *American Economic Review*, 81(2): 365–370.

LeBaron B (2001) Evolution and time horizons in an agent based stock market. *Macroeconomic Dynamics*, 5: 225–254.

LeBaron B, Arthur W, Palmer R (1999) Time series properties of an artificial stock market. *J. Economic Dynamics and Control*, 23: 1487–1516.

3 Journals

Computational Economics

Intl. J. Intelligent Systems in Accounting, Finance and Management

J. Computational Intelligence in Finance

J. Economic Dynamics and Control

J. Evolutionary Economics

J. Financial Economics

J. Monetary Economics

4 Key International Conferences/Workshops

4.1 Economics

Computational Intelligence in Economics and Finance (CIEF)

Intl. Conf. Computing in Economics and Finance (CEF)

Intl. Conf. Economic Science with Heterogeneous Interacting Agents (ESHIA)

World Conference on Social Simulation (WCSS)

4.2 Agents

Annual Conference on Neuroeconomics

Intl. ESA Conf. Experimental Economics (Economic Science Association)

International Workshop on Agent-Based Approaches in Economic and Social Complex Systems (AESCS)

North American Association for Computational Social and Organizational Sciences (NAACSOS)

5 (Open Source) Software

MASON: Multi-Agent Simulator

<http://cs.gmu.edu/~eclab/projects/mason/>

MATLAB

<http://www.mathworks.com/>

NetLogo

<http://ccl.northwestern.edu/netlogo/>

Repast

<http://repast.sourceforge.net/>

Sociodynamica

<http://atta.labb.usb.ve/Klaus/Programas.htm>

StarLogo

<http://education.mit.edu/starlogo/>

Swarm

http://www.swarm.org/wiki/Main_Page

6 Data Bases

COMPUSTAT

<https://www.compustatresources.com/support/index.html>

CRSP

<http://www.crsp.com/products/stocks.htm>

DatAnalysis

<http://www.deakin.edu.au/library/search/title/datanalysis.php>

Datastream

<http://www.datastream.com/>

Global Financial Data

<http://www.globalfinancialdata.com/>

Yahoo! Finance

<http://finance.yahoo.com/>

Fuzzy Systems

Semantics and Perception of Fuzzy Sets and Fuzzy Mappings

Witold Pedrycz

Department of Electrical and Computer Engineering, University of Alberta,
Edmonton, Canada, pedrycz@ece.ualberta.ca
Systems Research Institute, Polish Academy of Science, ul. Newelska 6,
01-447 Warszawa, Poland

1 Semantics of Fuzzy Sets: Some General Observations

Fuzzy sets are constructs that come with a well defined meaning [36–38]. They capture the semantics of the framework they intend to operate within. Fuzzy sets are the building conceptual blocks (generic constructs) that are used in problem description, modeling, control, pattern classification tasks and the like. Their estimation and interpretation are of paramount relevance. Before discussing specific techniques of membership function estimation, it is worth casting the overall presentation in a certain context by emphasizing the aspect of the use of a finite number of fuzzy sets leading to some essential vocabulary reflective of the underlying domain knowledge. In particular, we are concerned with the related semantics, calibration capabilities of membership functions, and the locality of fuzzy sets.

The limited capacity of a short-term memory, as lucidly identified by [16], implies that we could easily and comfortably handle and process 7 ± 2 items. This implies that the number of fuzzy sets to be considered as meaningful conceptual entities should be kept more or less at the same level. The observation sounds very convincing, and we can witness a lot of commonly encountered situations in which this condition holds. For instance, when describing linguistically quantified variables – say error or change of error – we may use seven generic concepts (descriptors) labeling them as positive(negative) *large*, positive(negative) *medium*, and positive(negative) *small*, and *around zero*. When characterizing speed, we may talk about its quite intuitive descriptors, such as *low*, *medium* and *high*. In the description of an approximation error, we may typically use the concept of a *small* error around a point of linearization (in all these examples, the terms are indicated in italics to emphasize the granular character of the constructs and the role being played there by fuzzy sets). While embracing very different tasks, all these descriptors exhibit a striking

similarity. They are information granules, not numbers (whose descriptive power is very much limited). In modular software development when dealing with a collection of modules (procedures, functions and alike), the list of their parameters is always limited to a few items, which is again a reflection of the limited capacity of the short-term memory. An excessively long parameter list is strongly discouraged due to possible programming errors and rapidly increasing difficulties of an effective comprehension of the software structure and ensuing flow of control.

In general, the use of an excessive number of terms does not offer any advantage. On the contrary, it remarkably clutters our description of the phenomenon and hampers further effective usage of such concepts we intend to establish to capture the essence of the domain knowledge. With the increase in the number of fuzzy sets, their semantics also become negatively impacted. Obviously, fuzzy sets may be built into a hierarchy of terms (descriptors) but at each level of this hierarchy (when moving down towards higher specificity in increasing level of detail), the number of fuzzy sets is kept at a certain limited level.

While fuzzy sets capture concept semantics, they may require some calibration depending upon the specification of the problem at hand [5–9, 12–14, 25, 28, 33, 35]. This flexibility of fuzzy sets should not be treated as a shortcoming but rather we should regard it as a definite and fully exploited advantage. For instance, a term *low* temperature comes with a clear meaning yet it requires a certain calibration depending upon the environment and the context it was put into. The concept of *low* temperature is used in different climate zones and is of relevance in any communication between people yet for each community the meaning of the term is different thereby requiring some calibration. This could be realized, for example, by shifting the membership function along the universe of discourse of temperature, affecting the universe of discourse by some translation, dilation and the like. As a means of communication, linguistic terms are fully legitimate and as such they appear in different settings. They require some refinement so that their meaning is fully understood and shared by the community of users.

When discussing methods aimed at the determination of membership functions or membership grades, it is worthwhile to underline the existence of the three main categories of approaches being reflective of the origin of the numeric values of membership. The first one is reflective of the domain knowledge and opinions of experts. In the second one, we consider experimental data whose global characteristics become reflected in the form and parameters of the membership functions. In the first group we can refer to the pairwise comparison [29, 30] as one representative example, while fuzzy clustering is usually presented as a typical example of the data-driven method of membership function estimation. There are also some hybrid approaches that tend to build upon the experimental data while taking advantage of the domain knowledge.

The key objectives of this study revolve around the fundamental concept of membership function estimation, calibration, perception, and reconciliation of views at information granules. Being fully cognizant of the semantics of fuzzy sets, it becomes essential to come up with a suite of effective mechanisms for the development of fuzzy sets and offer a comprehensive view of their interpretation.

Given the main objectives of the study, its organization is reflective of them. We start with a discussion on the use of domain knowledge and the problem-oriented formation of fuzzy sets (Sect. 2). In Sect. 3, we focus on user-centric estimation of membership functions (including horizontal, vertical and pairwise comparison). In the following Section, we focus on the construction of fuzzy sets regarded as granular representatives of numeric data. Fuzzy clustering is covered in Sect. 5. Several design guidelines are presented in Sect. 6. Fuzzy sets are typically cast in some context, and in this regard we discuss the issue of nonlinear mappings (transformations) that provide a constructive view of the calibration of fuzzy sets. The crux of the calibration deals with a series of contractions and expansions of selected regions of the universe of discourse. Further on we discuss several ways of reconciliation of fuzzy sets and fuzzy mappings being a direct result of various views (perspectives) of the same fuzzy set/fuzzy mapping.

The main points are augmented by some illustrative examples. Throughout the study we adhere to the standard notation of fuzzy sets [1, 21, 36].

2 Domain Knowledge and Problem-Oriented Formation of Fuzzy Sets

Fuzzy sets are often reflective of the nature of the problem and serve as its concise descriptors. In this sense the domain knowledge is encapsulated in some form in the corresponding membership functions. We discuss several major categories of the estimation methods.

2.1 Fuzzy Set as a Descriptor of Feasible Solutions

The underlying idea is to relate membership function with the level of feasibility of individual elements of a family of solutions to the problem at hand. Let us consider a certain function $F(x)$ defined in some domain Ω , that is $F: \Omega \rightarrow \mathbf{R}$, where $\Omega \subset \mathbf{R}$. Our intent is to determine its maximum – namely $x_0 = \arg \max_x F(x)$. On the basis of the values of $F(x)$, we construct a fuzzy set A which describes a collection of feasible solutions that could be labeled as ‘optimal’ or ‘near-optimal’. More specifically, we use the fuzzy set to represent the extent (degree) to which some specific values of ‘ x ’ could be regarded as potential (optimal) solutions to the problem. Taking this into consideration, we relate the membership function of A with the corresponding value of $F(x)$

cast in the context of the boundary values assumed by F . For instance, the membership function of A could be formed as follows:

$$A(x) = \frac{F(x) - F_{min}}{F_{max} - F_{min}} \tag{1}$$

The higher the membership grade $A(x)$, the higher the feasibility of ‘ x ’ as a possible solution to the maximization problem. The interpretation of the boundary conditions is also straightforward: $F_{min} = \min F(x)$ and $F_{max} = \max F(x)$, where the minimum and the maximum are computed over Ω . For other values of ‘ x ’ where F attains its maximal value, $A(x)$ is equal to 1 and around this point the membership values are reduced when ‘ x ’ is likely to be a solution to the problem $F(x) < F_{max}$. The form of the membership function depends upon the character of the problem under consideration. The following example illustrates the essence of the construction of membership functions carried out in this manner.

Example

Let us consider a problem of determining a maximum of the function $2\sin(0.5x)$, defined in $[0, 2\pi]$. The minimum and maximum of F in the range of the arguments between 0 and 2π is equal to 0 and 2, respectively. The maximal value of F is reached at $x = \pi$. The membership function of the solution to the optimization problem is computed, using Eqn. (1) to be $A(x) = \sin(0.5x)$ (Fig. 1).

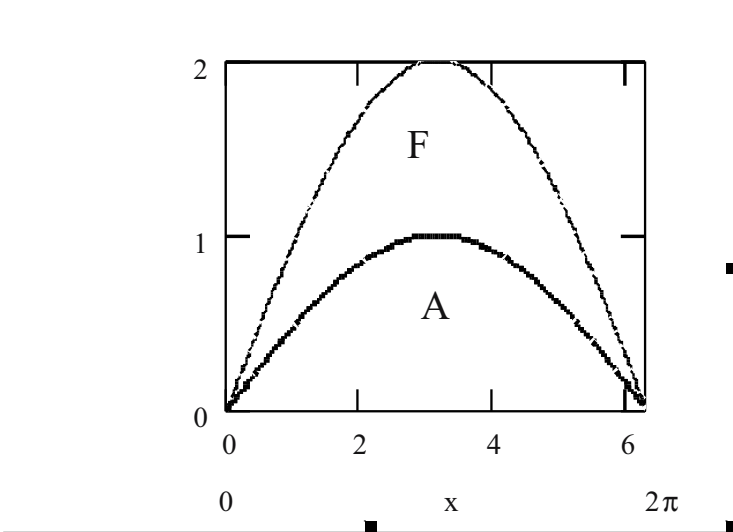


Fig. 1. Function- F and the induced membership function- A

Linearization, its quality and description of quality falls under the same banner as the optimization problem. The quality of linearization is not a binary concept. When linearizing a function around some given point, the quality of such linearization can be represented in the form of some fuzzy set. Its membership function attains the value 1 for all these points where the linearization error is equal to 0 (in particular, this holds at the point around which the linearization is carried out). The following example illustrates this idea.

Example

We are interested in the linearization of the function $y = f(x) = e^{-x}$ around $x_o = 1$, and assessing the quality of this linearization in the range $[-1, 7]$. The linearization formula reads as $y - y_0 = f'(x_0)(x - x_0)$. Given the form of the function $f(x) = \exp(-x)$, the linearized version of the function around x_0 reads as $\exp(-1)(2 - x)$. Next let us define the quality of this linearization by taking the absolute value of the difference between the original function and its linearization, $F(x) = |f(x) - \exp(-1)(2 - x)|$. As the fuzzy set A describes the quality of linearization, its membership function has to take into consideration the expression

$$A(x) = 1 - \frac{F(x) - F_{min}}{F_{max} - F_{min}} \tag{2}$$

where $F_{max} = F(7) = 1.84$, and $F_{min} = 0.0$. Obviously, when at some z we have $F(z) = F_{min}$, this means that $A(z) = 1$, which in the sequel indicates that the linearization at this point is perfect; no linearization error has been generated.

The plot of the fuzzy set A is shown in Fig. 2. We note that a higher quality of approximation is achieved for arguments higher than the point at which linearization has been completed. Furthermore, the membership function is highly asymmetric, which sheds light on the character of the linearization process.

2.2 Fuzzy set as a Descriptor of the Notion of Typicality

Fuzzy sets address an issue of gradual *typicality* of elements when being evaluated with respect to a certain concept. They stress the fact that there are elements that fully satisfy the concept (are *typical* for it), and there are various elements that are allowed to belong to the concept only with some membership degrees below 1. The form of the membership function is reflective of the semantics of the concept. Its details could be captured by adjusting the parameters of the membership function, or by choosing its form depending upon experimental data. For instance, consider a fuzzy set of circles. Formally, an ellipse includes a circle shape as its special example when the axes

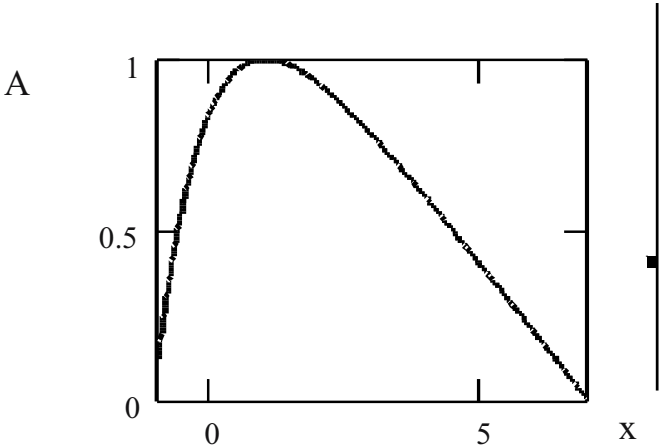


Fig. 2. Fuzzy set-*A* representing the quality of linearization of the function $exp(-x)$ around the point $x_0 = 1$

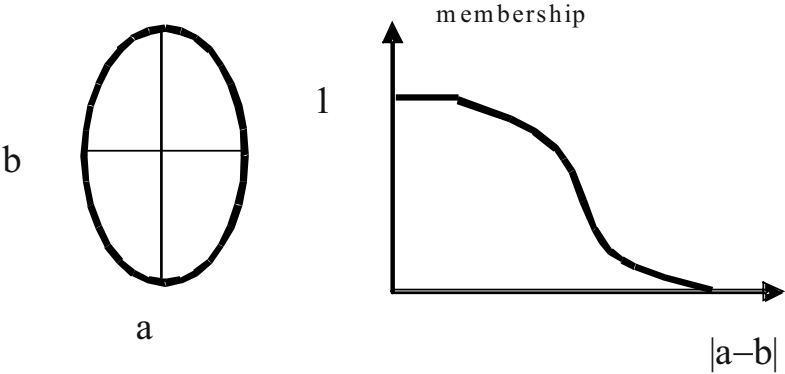


Fig. 3. Perception of the geometry of a circle, and its quantification in the form of membership function of the concept ‘fuzzy circle’

are equal $a = b$ (Fig. 3). What if $a = b + \epsilon$, where ϵ is a very small positive number? Could this figure come with the membership value of the corresponding membership function equal to 0.99. Our perception, which comes with some level of tolerance to imprecision, does not allow us to tell apart this figure from the ideal circle, Fig. 3.

Higher differences between axes of the ellipses – in other words, ‘a’ and ‘b’, could result in lower values of the membership function. The definition of the fuzzy set of circle could be expressed in a number of ways. Prior to the definition or even visualization of the membership function, it is important to formulate a space over which it will be defined. There are several quite intuitive alternatives worth considering:

- (a) for each pair of values of the axes (a and b), collect an experimental assessment of membership of the ellipse to the category of circles. Here the membership function is defined over a Cartesian space of the spaces of lengths of axes of the ellipses. While selecting a form of the membership we require that it assumes values at $a = b$ and is gradually reduced when the arguments start getting more distinct;
- (b) we can define an absolute distance between a and b , $|a - b|$, and form a fuzzy set over this space \mathbf{X} ; $\mathbf{X} = \{x \mid x = |a - b|\} \mathbf{X} \in \mathbf{R}_+$. The semantic constraints translate into the condition of $A(0) = 1$. For higher values of x we may consider monotonically decreasing values of A ;
- (c) we can envision ratios of a and b , say $x = a/b$, and construct a fuzzy set over the subset of space of positive reals \mathbf{R}_+ such that $\mathbf{X} = \{x \mid x = a/b\}$. Here we require that $A(1) = 1$. We also anticipate lower values of membership grades when moving to the left and to the right from $x = 1$. Note that the membership function could be asymmetric as we allow for different membership values for the same length of the axes, say $a = 6$, $b = 5$ and $a = 5$ and $b = 6$ (the phenomenon of asymmetry could be quite apparent due to the visual effects when perceiving geometric phenomena). The previous model of X as outlined in (a) cannot capture this effect;

Once the form of the membership function has been defined, it could be further adjusted by modifying the values of its parameters, which is done on the basis of experimental findings. The parameters come in the form of ordered triples or pairs, say (a, b, m) , $(a/b, m)$ or $(|a - b|, \mu)$, depending on the previously accepted definition of the universe of discourse. The membership values μ are those available from the expert offering an assessment of the likeness (resemblance) of the corresponding geometric figure.

2.3 Membership Functions in the Visualization of Preferences of Solutions

Fuzzy sets could be used as an effective vehicle to use membership functions to visualize and quantify levels of preference of a family of possible solutions to a given problem. A simple electrical circuit – Fig. 4 – helps us illustrate the underlying idea. Consider the problem of power maximization in the external resistance in the circuit.

The voltage source E is characterized by some internal resistance equal to r . The external resistance R is the one on which we want to maximize power dissipation. By straightforward calculations we determine that the power dissipated on R is given as

$$P = i^2 R = \left(\frac{E}{R + r} \right)^2 R \tag{3}$$

where E is the voltage of the source. The maximization of P with respect to R is determined by zeroing the derivative of $P(dP/dR) = 0$, which leads

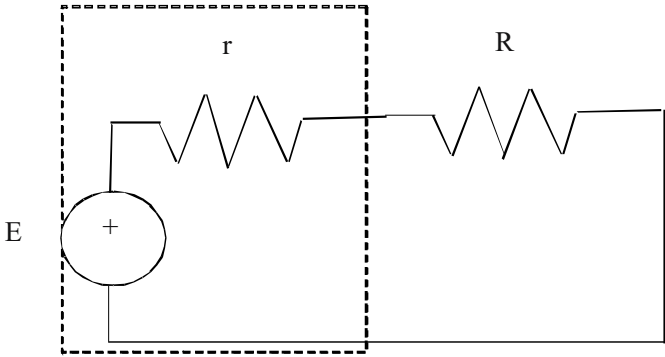


Fig. 4. A simple electrical circuit and the problem of maximization of power dissipation in the external resistance- R

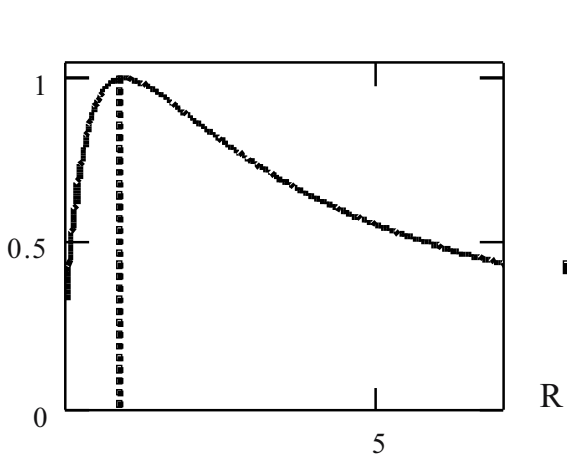


Fig. 5. Membership function of the optimal power dissipation in external resistance R ; the maximal value is achieved for $R = 1$ (the internal resistance r is equal to 1)

to determination of the optimal value of the resistance R_{opt} . Through simple derivations we obtain $R_{opt} = r$.

It becomes evident that while the condition $R = r$ produces the maximum of P , this solution is not technically feasible as there is a substantial level of power dissipation on the internal resistance. If we plot the relationship of P versus R (Fig. 5), and treat it as a membership function of R (which requires a simple normalization of P by dividing it by the maximal value obtained for $R = r$), we note that the shape of this relationship is highly asymmetric: when increasing the value of resistance over the optimal value (R_{opt}), the values of the membership function change quite smoothly and the reduction of the membership grades becomes quite limited.

On the other hand, when moving towards lower values of R such that $R < R_{opt}$, the reduction in the membership grades is quite substantial. We can say that the membership function of the optimal resistance offers a highly visible and very much intuitive quantification of the notion of optimality. The asymmetric shape of the resulting fuzzy set delivers some guidance in the selection of possible suboptimal solutions, while the membership degree serves as an indicator of the suitability (degree of optimality) of the individual value of R .

3 User-Centric Estimation of Membership Functions

Experts, human observers, users, and designers can provide an important insight into the quantification of membership grades. Several systematic ways of estimation of fuzzy sets have been proposed in this regard. The vertical and horizontal modes of membership estimation are two standard approaches used in the determination of fuzzy sets. They reflect distinct ways of looking at fuzzy sets whose membership functions at some finite number of points are quantified by experts. The pairwise method of membership estimation is another interesting alternative.

3.1 Horizontal Membership Function Estimation Scheme

In the horizontal approach we identify a collection of elements in the universe of discourse X and request that an expert answers the question: “Does x belong to concept-A?”

The answers are allowed to be in a binary (yes-no) format. The concept A defined in \mathbf{X} could be any linguistic notion, say *high* speed, *low* temperature, or similar. Given a group of ‘ n ’ experts whose answers for a given point of \mathbf{X} form a mix of yes-no replies, we count the number of affirmative answers and compute the ratio of the positive answers (p) versus the total number of replies (n) – that is p/n . This ratio (likelihood) is treated as a membership degree of the concept at the given point of the universe of discourse. When all experts accept that the element belongs to the concept, then its membership degree is equal to 1. Higher disagreement between the experts (quite divided opinions) results in lower membership degrees. The concept A defined in \mathbf{X} requires collecting results for some other elements of \mathbf{X} and determining the corresponding ratios, as outlined in Fig. 6.

As the replies follow some binomial distribution, we could easily determine a confidence interval of the individual membership grade. The standard deviation of the estimate of the positive answers associated with the point x , denoted here by σ is given in the form

$$\sigma = \sqrt{\frac{p(1-p)}{n}} \quad (4)$$

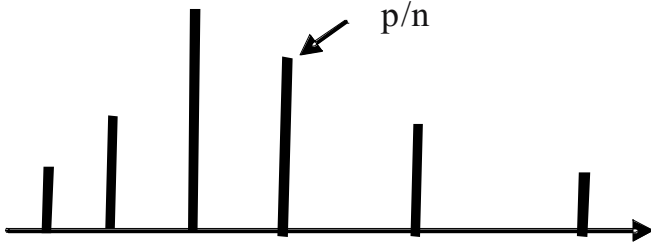


Fig. 6. Horizontal method of the estimation of the membership function: observe a series of estimates determined for selected elements of \mathbf{X} (note also that the elements of \mathbf{X} need not to be evenly distributed)

The associated confidence interval which describes a range of membership values is then determined as

$$[p - \sigma, p + \sigma] \tag{5}$$

In essence, when the confidence intervals are taken into consideration, the membership estimates become intervals of possible membership values and this leads to the concept of so-called ‘interval-valued’ fuzzy sets. By assessing the width of the estimates, we could control the execution of the experiment: when the ranges are too wide, one could re-design the experiment and monitor closely the consistency of the responses collected.

The advantage of the method comes with its simplicity, as the technique relies explicitly upon a direct counting of responses. The concept is also intuitively appealing. The probabilistic nature of the replies is useful in forming build confidence intervals that are essential to the assessment of the specificity of the membership quantification. However one drawback is related to the local character of the construct: as the estimates of the membership function are completed separately for each element of the universe of discourse, they could exhibit a lack of continuity when moving from a specific point to its neighbor. Obviously, this concern is particularly valid in the case when \mathbf{X} is a subset of real numbers.

3.2 Vertical Membership Function Estimation Scheme

The vertical mode of membership estimation is concerned with the estimation of the membership function by focusing on the determination of successive α -cuts. The experiment focuses on the unit interval of membership grades. The experts involved in the experiment are asked questions of the form:

“what are the elements of \mathbf{X} which belong to fuzzy set A at degree not lower than α ?”

where α is a certain level (threshold) of membership grades in $[0,1]$.

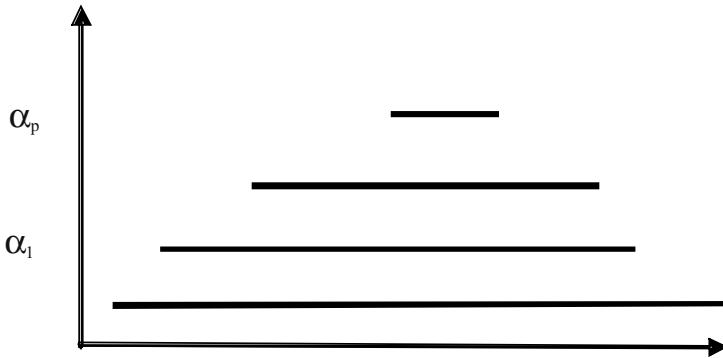


Fig. 7. Vertical approach of membership estimation through the reconstruction of a fuzzy set through its estimated α -cuts

The essence of the method is illustrated in Fig. 7. Note that the satisfaction of the inclusion constraint has a straightforward interpretation: we envision that for higher values of α , the expert is going to provide more limited subsets of \mathbf{X} . In a nutshell, the vertical approach leads to the fuzzy set by combining the estimates of the corresponding α -cuts. Given the nature of this method, we are referring to the collection of random sets as these estimates appear in succeeding stages of the estimation process.

The elements are identified by the expert as they form the corresponding α -cuts of A . By repeating the process for several selected values of α we end up with the α -cuts, and using them we reconstruct the fuzzy set. This could be done by solving the corresponding approximation problem. The simplicity of the method is its genuine advantage. As with the horizontal method of membership estimation, a possible lack of continuity is a particular disadvantage one has to be aware of. Here the selection of suitable levels of needs to be carefully investigated. Similarly, the order in which different levels of α are used in the experiment could impact the estimate of the membership function.

3.3 Pairwise Membership Function Estimation Scheme

The priority method introduced by [29, 30] forms another interesting alternative used to estimate the membership function. To explain the essence of the method, let us consider a collection of elements x_1, x_2, \dots, x_n (which could be, for instance, some alternatives whose allocation to a certain fuzzy set is sought), with corresponding membership grades $A(x_1), A(x_2), \dots, A(x_n)$. Let us organize them into a so-called reciprocal matrix of the following form:

$$R = [r_{ij}] = \begin{bmatrix} \frac{A(x_1)}{A(x_1)} & \frac{A(x_1)}{A(x_2)} & \dots & \frac{A(x_1)}{A(x_n)} \\ \frac{A(x_2)}{A(x_1)} & \frac{A(x_2)}{A(x_2)} & \dots & \frac{A(x_2)}{A(x_n)} \\ \dots & \dots & \dots & \dots \\ \frac{A(x_n)}{A(x_1)} & \frac{A(x_n)}{A(x_2)} & \dots & \frac{A(x_n)}{A(x_n)} \end{bmatrix} = \begin{bmatrix} 1 & \frac{A(x_1)}{A(x_2)} & \dots & \frac{A(x_1)}{A(x_n)} \\ \frac{A(x_2)}{A(x_1)} & 1 & \dots & \frac{A(x_2)}{A(x_n)} \\ \dots & \dots & \dots & \dots \\ \frac{A(x_n)}{A(x_1)} & \frac{A(x_n)}{A(x_2)} & \dots & 1 \end{bmatrix} \quad (6)$$

Noticeably, the diagonal values of R are equal to 1. The entries that are symmetrically positioned with respect to the diagonal satisfy the condition of reciprocity, namely that $r_{ij} = \frac{1}{r_{ji}}$. Furthermore an important transitivity property holds – $r_{ik}r_{kj} = r_{ij}$ – for all indexes i, j , and k . This property is obvious. By plugging in the ratios one gets $r_{ik}r_{kj} = \frac{A(x_i)}{A(x_k)} \frac{A(x_k)}{A(x_j)} = \frac{A(x_i)}{A(x_j)} = r_{ij}$. Let us now multiply the matrix by the vector of the membership grades $A = [A(x_1)A(x_2) \dots A(x_n)]^T$. For the i -th row of R (that is, the i -th entry of the resulting vector of results) we obtain

$$R = [r_{ij}] = \left[\frac{A(x_i)}{A(x_1)} \frac{A(x_i)}{A(x_2)} \dots \frac{A(x_i)}{A(x_n)} \right] \begin{bmatrix} A(x_1) \\ A(x_2) \\ \dots \\ A(x_n) \end{bmatrix} \tag{7}$$

$i = 1, 2, \dots, n$. Thus the i -th element of the vector is equal to ‘ n ’. Overall, once completing the calculations for all ‘ i ’ this leads us to the expression $RA = nA$. In other words, we conclude that A is the eigenvector of R associated with the largest eigenvalue of R , which is equal to ‘ n ’. Obviously in the above scenario, we have assumed that the membership values $A(x_i)$ are given and then showed what form of results could they lead to. In practice the membership grades are not given and have to be determined.

The starting point of the estimation process are entries of the reciprocal matrix which are obtained through collecting results of pairwise evaluations offered by an expert, designer or user (depending on the character of the task at hand). Prior to making any assessment, the expert is provided with a finite scale with values spread in between 1 and 7. Alternatively, scales involving 5 (1–5) or 9 (1–9) levels could be sought. If x_i is strongly preferred over x_j when being considered in the context of the fuzzy set whose membership function we would like to estimate, then this judgment is expressed by assigning high values of the available scale, say 6 or 7. If we still sense that x_i is preferred over x_j yet the strength of this preference is lower in comparison with the previous case, then this is quantified using some intermediate values of the scale, say 3 or 4. If no difference is sensed, values close to 1 are the preferred choice, say 2 or 1. The value of 1 indicates that x_i and x_j are equally preferred. On the other hand, if x_j is preferred over x_i , the corresponding entry assumes values below one.

Given the reciprocal character of the assessment, once the preference of x_i over x_j has been quantified, the inverse of this number is plugged into the entry of the matrix that is located at the (j, i) -th coordinate. As indicated earlier, the elements on the main diagonal are equal to 1. Next the maximal eigenvalue is computed along with its corresponding eigenvector. The normalized version of the eigenvector is then the membership function of the fuzzy set we considered when doing all pairwise assessments of the elements of its universe of discourse.

The pairwise evaluations are far more convenient and manageable in comparison to any effort we make when assigning membership grades to all elements of the universe in a single step. Practically, pairwise comparison helps the expert focus only on two elements one at a time, thus reducing uncertainty and hesitation while leading to a higher level of consistency. The assessments are not free of bias and could exhibit some inconsistent evaluations. In particular, we cannot expect that the transitivity requirement could be fully satisfied. Fortunately, the lack of consistency could be quantified and monitored. The largest eigenvalue computed for R is always greater than the dimensionality of the reciprocal matrix, $\lambda_{max} > n$, where the equality $\lambda_{max} = n$ occurs only if the results are fully consistent. The ratio

$$\nu = (\lambda_{max} - n)/(n - 1) \quad (8)$$

can be treated as an index of inconsistency of the data; the higher its value, the less consistent are the collected experimental results. This expression can be sought as the indicator of the quality of the pairwise assessments provided by the expert. If the value of ν is too high (exceeding a certain superimposed threshold), the experiment may need to be repeated. Typically if ν is less than 0.1 the assessment is sought to be consistent, while higher values of ν call for re-examination of the experimental data and the experiment to be re-run. To quantify how much the experimental data deviates from the transitivity requirement, we calculate the absolute differences between the corresponding experimentally obtained entries of the reciprocal matrix, namely r_{ik} and $r_{ij}r_{jk}$. The sum expressed in the form

$$V(i, k) = \sum_{j=1}^n | r_{ij}r_{jk} - r_{ik} | \quad (9)$$

serves as a useful indicator of the lack of transitivity of the experimental data for the given pair of elements (i, k) . If required, we may repeat the experiment if the above sum takes high values. The overall sum $\sum_{j=1}^n V(i, k)$ then becomes a global evaluation of the lack of transitivity of the experimental assessment. There have been a number of generalizations of the generic method – see, for example [3, 11, 15, 28, 34].

4 Fuzzy Sets as Granular Representatives of Numeric Data

In general, a fuzzy set is reflective of some numeric data that are put together in some context. Using its membership function we attempt to embrace them in some concise manner. The development of the fuzzy set is supported by

the following experiment-driven and intuitively appealing rationale [2, 27]:

- (a) we expect that A ‘reflects’ (or matches) the available experimental data to the highest extent, and
- (b) the fuzzy set is kept specific enough so that it comes with a well-defined semantics. Furthermore, which is quite legitimate, we assume that the fuzzy set to be constructed has a unimodal membership function or its maximal membership grades occupy a contiguous region in the universe of discourse in which this fuzzy set has been defined. This helps us build a membership function separately for its rising and decreasing sections. The core of the fuzzy set is determined first. Next, assuming the simplest scenario when using the linear type of membership functions, the essence of the optimization is such that we rotate the linear section of the membership function around the point of the core of A , as indicated in Fig. 8. Note that the core of A is built on the basis of two data points. The point of rotation of the linear segment of the membership function is marked by an empty circle

Before moving on with the determination of the membership function, we concentrate on the location of its numeric representative. Typically, one could view an average of the experimental data x_1, x_2, \dots, x_n to be its sound representative. While its usage is quite common in practice, a better representative of the numeric data is a median value. There is a compelling reason behind this choice. The median is a robust statistic meaning that it allows for a high level of tolerance to potential noise existing in the data. Its important ability is to ignore outliers. Given that the fuzzy set is sought to be a granular and ‘stable’ representation of the numeric data, our interest is in the robust

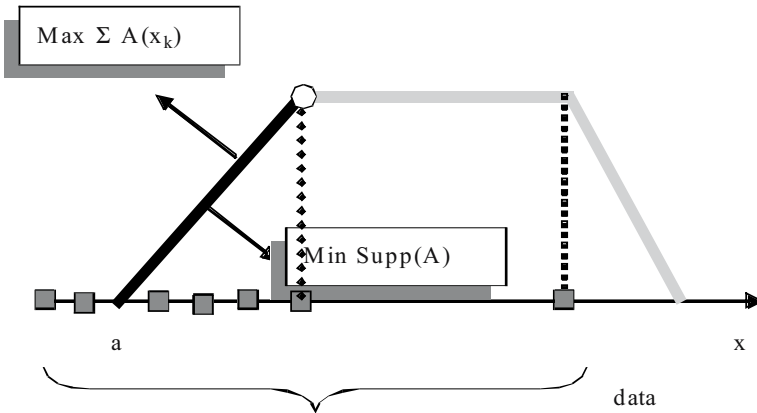


Fig. 8. Optimization of the linear increasing section of the membership function of A ; highlighted are the positions of the membership function originating from the realization of the two conflicting criteria (that is, a maximization of the experimental evidence behind the fuzzy set and a minimization of its spread)

development not being affected by noise. Undoubtedly, the use of the median is a good starting point. Let us recall that the median is an order statistic and is formed on the basis of an ordered set of numeric values. In the case of an odd number of data in the data set, the point located in the middle of this ordered sequence is the median. When we encounter an even number of data in the granulation window, instead of picking up an average of the two points located in the middle, we consider these two points to form a core of the fuzzy set. Thus, depending upon the number of data points, we either end up with a triangular or trapezoidal membership function.

Having fixed the modal value of A (that could be a single numeric value, ‘ m ’ or a certain interval $[m, n]$), the optimization of the spreads of the linear portions of the membership functions are carried out separately for their increasing and decreasing portions. We consider the increasing part of the membership function (the decreasing part is handled in an analogous manner). Referring to Fig. 8, the two requirements guiding the design of the fuzzy set are transformed into the corresponding optimization problem:

- (a) maximize the experimental evidence of the fuzzy set; this implies that we tend to ‘cover’ as many numeric data as possible – in other words, the coverage has to be made as high as possible. Graphically, in the optimization of this requirement, we rotate the linear segment up (clockwise), as illustrated in Fig. 8. Formally, the sum of the membership grades $A(x_k) \sum_k A(x_k)$ (where A is the linear membership function to be optimized and x_k is located to the left to the modal value) has to be maximized.
- (b) Simultaneously, we would like to make the fuzzy set as specific as possible so that it comes with some well defined semantics. This requirement is met by making the support of A as small as possible, that is $\min_a |m - a|$.

To accommodate these two conflicting requirements, we combine (a) and (b) in the form of the ratio that is maximized with respect to the unknown parameter of the linear section of the membership function

$$Max_a = \frac{\sum_k A(x_k)}{|m - a|} \tag{10}$$

The linearly decreasing portion of the membership function is optimized in the same manner. The overall optimization returns the parameters of the fuzzy number in the form of the lower and upper bound (a and b , respectively) and its support (m or $[m, n]$). We can write down the resulting fuzzy number in the form $A(a, m, n, b)$, which underlines the parameters determining this fuzzy set.

Example

As an illustration, let us consider a scenario where experimental numeric data are governed by some uniform probability density function defined over the

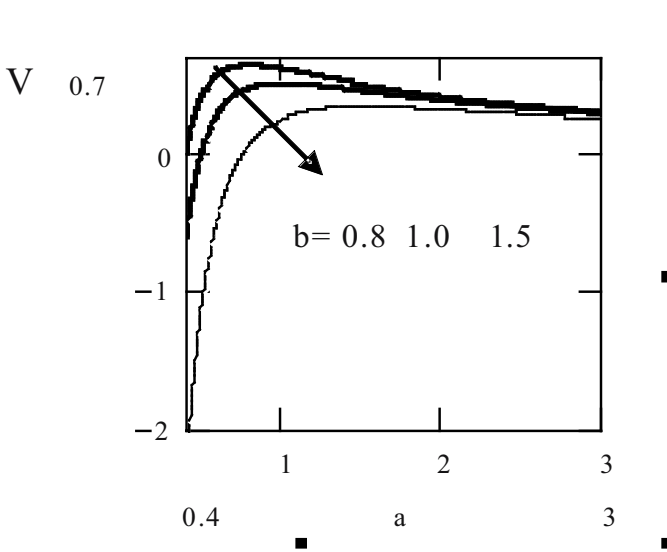


Fig. 9. Plots of V versus ‘a’ for selected values of ‘b’

range $[0, b]$, $b > 0$ – that is $p(x) = 1/b$ over $[0, b]$, and 0 otherwise. The linear membership function of A is of the form $A(x) = \max(0, 1 - x/a)$. The modal value of A is equal to zero. The above optimization criterion Eqn. (10) now reads as

$$V(a) = \frac{\int_0^a A(x)p(x)dx}{a} = \frac{1}{ab} \int_0^a \left(1 - \frac{x}{a}\right) dx = \frac{1}{ab} \left(b - \frac{b^2}{2a}\right) = \frac{2a - b}{2a^2} \quad (11)$$

The plot of V regarded as a function of the optimized slope of A is shown in Fig. 9; here we use different values of ‘b’ to visualize the effect of this parameter on the behavior of V .

The optimal value of ‘a’ results from the relationship $dV/da = 0$ and this leads to the intuitively appealing result of $a = b$. The form of the relationship $V = V(a)$ is highly asymmetric; while values of ‘a’ higher than the optimal value (a_{opt}) lead to a very slow degradation of performance (V changing slowly). Rapid changes in V are noted for the values of ‘a’ which are lower than the optimal value.

Example

We show the details on how the data-driven triangular fuzzy set is being formed. The data set under discussion consists of the following numeric data

$$\{-2.00 \ 0.80 \ 0.90 \ 1.00 \ 1.30 \ 1.70 \ 2.10 \ 2.60 \ 3.30\}$$

The values of the performance index obtained during optimization of the left and right part of the slope of the triangular membership function and

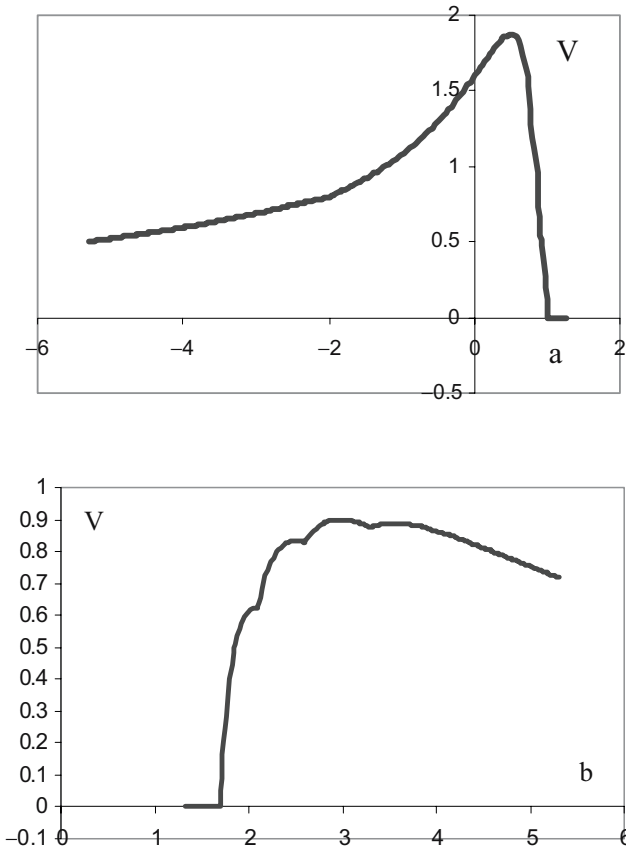


Fig. 10. Values of the performance index V optimized for the linear sections of the membership function; in both cases we note a clearly visible maximum occurring at both sides of the modal value of the fuzzy set that determine the location of the bounds of the membership function ($a = 0.51$ and $b = 2.96$)

being viewed as a function of the intercept are shown in Fig. 10. The performance index shows a clear maximum for both linear parts of the membership function. The final result coming in the form of the triangular fuzzy set (fuzzy number) is uniquely described by its bounds and the modal value – altogether described as $A(0.51 \ 1.30 \ 2.96)$. This shows us how a sound compromise has been reached between the spread of the fuzzy set that helps us assure a solid coverage of the data while retaining its high specificity (limited spread). The result is quite appealing as the fuzzy set formed in this way nicely captures the core part of the numeric data.

So far we have discussed only linear types of membership functions (in other words, those with linearly increasing or decreasing sections) however *any*

monotonically increasing or decreasing function could be sought as a viable alternative. In particular, a polynomial (monomial, to be more precise) type of relationships, say x^p with ‘ p ’ being a positive integer could be of interest. The values of ‘ p ’ impact the shape and more importantly, the spread of the resulting fuzzy set.

5 From Multidimensional Numeric Data to Fuzzy Sets: Membership Estimation via Fuzzy Clustering

Fuzzy sets can be formed on the basis of numeric data through their clustering (groupings). These groups of data give rise to membership functions that convey a global, more abstract view of the available data. In this regard, Fuzzy C-Means (FCM, for short) is one of the commonly used mechanisms for fuzzy clustering [1, 17, 21, 24].

Let us review its formulation, develop the algorithm, and highlight the main properties of fuzzy clusters. Given a collection of n -dimensional data $\mathbf{x}_k, k = 1, 2, \dots, N$, the task of determining its structure – a collection of ‘ c ’ clusters – is expressed as a minimization of the following objective function (performance index), Q being regarded as a sum of the weighted squared distances

$$Q = \sum_{i=1}^c \sum_{k=1}^N u_{ik}^m \|\mathbf{x}_k - \mathbf{v}_i\|^2 \tag{12}$$

where \mathbf{v}_i are the prototypes of the clusters, $i = 1, 2, \dots, c$ and $U = [u_{ik}]$ stands for a partition matrix expressing a way of allocating the data to the corresponding clusters. The distance between the data \mathbf{x}_k and prototype \mathbf{v}_i is denoted by $\|\cdot\|$. The fuzzification coefficient $m(>1.0)$ quantifies the impact of the membership grades on the individual clusters.

A partition matrix satisfies two important and intuitively appealing properties

$$0 < \sum_{k=1}^N u_{ik} < N, i = 1, 2, \dots, c \tag{13}$$

and

$$\sum_{i=1}^c u_{ik} = 1, k = 1, 2, \dots, N \tag{14}$$

Let us denote by \mathbf{U} a family of matrices satisfying Eqns. (13) and (14). The first requirement states that each cluster has to be nonempty and different from the entire set (which is straightforward). The second requirement states that the sum of the membership grades should be confined to 1.

The minimization of Q is completed with respect to $U \in \mathbf{U}$ and the prototypes of the clusters. More explicitly, we write it down as follows: $\min Q$ with respect to $U \in \mathbf{U}, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_c$.

From the optimization standpoint, there are two individual optimization tasks to be carried out separately for the partition matrix and the prototypes. The first one concerns the minimization with respect to the constraints given the requirement of the form Eqn. (14), which holds for each data point \mathbf{x}_k . The use of Lagrange multipliers converts the problem into its constraint-free version. The augmented objective function formulated for each data point, $k = 1, 2, \dots, N$, reads as

$$V = \sum_{i=1}^c u_{ik}^m d_{ik}^2 + \lambda (\sum_{i=1}^c u_{ik} - 1) \tag{15}$$

where $d_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$. Proceeding with the necessary conditions for the minimum of V for $k = 1, 2, \dots, N$, one has

$$\frac{\partial V}{\partial u_{st}} = 0; \quad \frac{\partial V}{\partial \lambda} = 0; \quad \text{and} \quad u_{st} = \frac{1}{\sum_{j=1}^c \left(\frac{d_{st}^2}{d_{jt}^2}\right)^{\frac{1}{m-1}}} \tag{16}$$

Optimization of the prototypes \mathbf{v}_i is carried out assuming the Euclidean distance between the data and the prototypes that is $\|\mathbf{x}_k - \mathbf{v}_i\|^2 = \sum_{j=1}^n (x_{kj} - v_{ij})^2$. The objective function reads now as follows $Q = \sum_{i=1}^c \sum_{k=1}^N u_{ik}^m \sum_{j=1}^n (x_{kj} - v_{ij})^2$, and its gradient with respect to $\mathbf{v}_i - \nabla_{v_i} Q$ - made equal to zero yields the system of linear equations

$$\sum_{j=1}^n u_{ik}^m (x_{kt} - v_{st}) = 0 \tag{17}$$

$s = 1, 2, \dots, c; t = 1, 2, \dots, n$.

Thus

$$v_{st} = \frac{\sum_{k=1}^N u_{ik}^m x_{kt}}{\sum_{k=1}^N u_{ik}^m} \tag{18}$$

Overall, the FCM clustering is completed through a sequence of iterations where we start from some random allocation of data (a certain randomly initialized partition matrix) and carry out the following updates by adjusting the values of the partition matrix and the prototypes:

Algorithm 1

repeat

(i) update the prototypes

$$\mathbf{v}_i = \frac{\sum_{k=1}^N \mathbf{u}_{ik}^m \mathbf{x}_k}{\sum_{k=1}^N \mathbf{u}_{ik}^m}; \quad i = 1, 2, \dots, c \quad (19)$$

(ii) update the partition matrix

$$\mathbf{u}_{ik} = \frac{1}{\sum_{j=1}^c \left(\frac{\|\mathbf{x}_k - \mathbf{v}_i\|}{\|\mathbf{x}_k - \mathbf{v}_j\|} \right)^{2/(m-1)}}; \quad i = 1, 2, \dots, c; \quad k = 1, 2, \dots, N \quad (20)$$

until some termination criterion has been satisfied

The iteration loop (i)–(ii) is repeated until a certain termination criterion has been satisfied. Typically, the termination condition is quantified by looking at the changes in the membership values of the successive partition matrices. Denote by $U(iter)$ and $U(iter + 1)$ the two partition matrices produced in two consecutive iterations of the algorithm. If the distance $\|U(iter + 1) - U(iter)\|$ is less than a small predefined threshold ε , then we terminate the algorithm. Typically, one considers the Tchebyshev distance between the partition matrices meaning that the termination criterion reads as follows

$$\max_{i,k} |u_{ik}(iter + 1) - u_{ik}(iter)| \leq \varepsilon \quad (21)$$

The key components of the FCM and a quantification of their impact on the form of the produced results are summarized in Table 1.

The fuzzification coefficient exhibits a direct impact on the geometry of fuzzy sets generated by the algorithm. Typically, the value of ‘ m ’ is assumed to be equal to 2.0. Lower values of m (that are closer to 1) yield membership functions that start resembling characteristic functions of sets; most of the membership values become localized around 1 or 0. An increase of the fuzzification coefficient ($m = 3, 4$, and so on) produces ‘spiky’ membership functions with membership grades equal to 1 at the prototypes and a fast decline of the values when moving away from the prototypes. Several illustrative examples of membership functions are included in Fig. 11. The prototypes here are equal to 1, 3.5 and 5, while the fuzzification coefficient assumes values of 1.2 (top), 2.0 (middle) and 3.5 (bottom), respectively. The intensity of the rippling effect is affected by ‘ m ’, increasing with higher values of ‘ m ’. In addition to the varying shape of the membership functions, observe that the requirement put on the sum of membership grades imposed on the fuzzy sets yields some rippling effect; the membership functions are not unimodal but

Table 1. Main features of the Fuzzy C-Means (FCM) clustering algorithm

FCM algorithm feature	Representation and optimization aspects
Number of clusters (c)	Structure in the data set and the number of fuzzy sets estimated by the method; an increase in the number of clusters produces lower values in the objective function, however given the semantics of fuzzy sets one should maintain this number quite low (5–9 information granules)
Objective function Q	Develops a structure aimed at the minimization of Q ; iterative process supports the determination of the local minimum of Q
Distance function $\ \cdot \ $	Reflects (or imposes) a geometry of the clusters one is looking for – an essential design parameter affecting the shape of the membership functions
Fuzzification coefficient (m)	Implies a certain shape of membership functions present in the partition matrix; essential design parameter. Low values of ‘ m ’ (being close to 1.0) induce characteristic function. Values higher than 2.0 yield spiky membership functions
Termination criterion	The distance between partition matrices in two successive iterations; the algorithm terminates once the distance falls below some assumed positive threshold (ε), in other words, $\ U(iter + 1) - U(iter) \ < \varepsilon$

may exhibit some ripples whose intensity depends upon the distribution of the prototypes and the values of the fuzzification coefficient.

These membership functions offer an interesting feature of evaluating the extent to which a certain data point is ‘shared’ between different clusters, and in this sense become difficult to allocate to a *single* cluster (fuzzy set). Let us introduce the following index which serves as a certain separation measure

$$\varphi(u_1, u_2, \dots, u_c) = 1 - c^c \prod_{i=1}^c u_i \tag{22}$$

where u_1, u_2, \dots, u_c are the membership degrees for some data point. If one of membership degrees, say $u_i = 1$, then the separation index attains its maximum equal to 1. At the other extreme, when the data point is shared by all clusters to the same degree (equal to $1/c$), then the value of the index drops down to zero. This means that there is *no* separation between the clusters for this specific point.

For instance, if $c = 2$, Eqn. (22) relates directly to the entropy of fuzzy sets. We have $\varphi(u) = 1 - 4u(1 - u)$. The term $H(u) = 4u(1 - u)$ is one version of the entropy function (recall that in the general form this function satisfies the

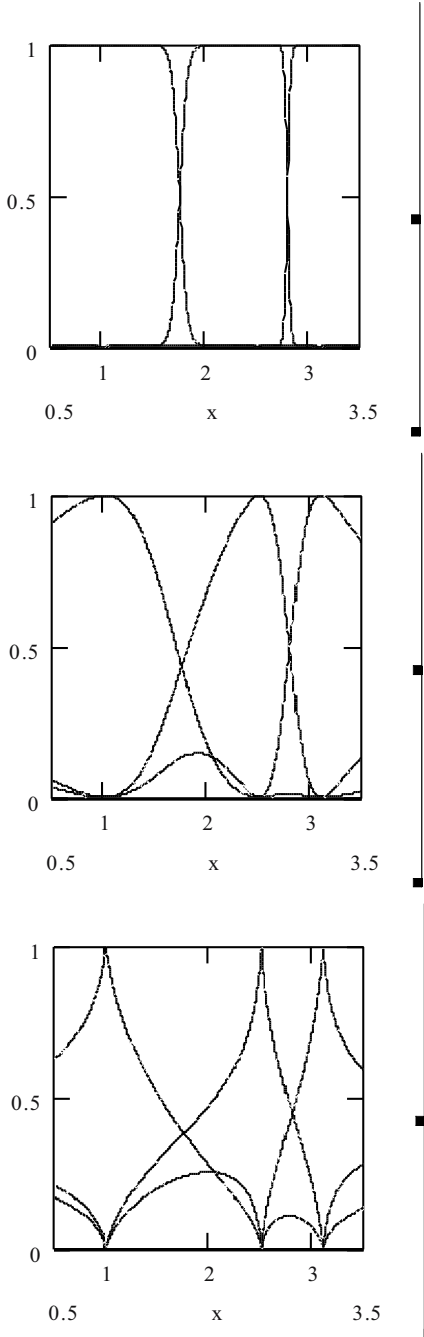


Fig. 11. Examples of membership functions of fuzzy sets; the prototypes are equal to 1, 3.5 and 5, while the fuzzification coefficient assumes values of 1.2 (a), 2.0 (b) and 3.5 (c), respectively (the intensity of the rippling effect increases as a function of ‘m’)

set of requirements: (a) boundary conditions $H(0) = H(1) = 0$, H is monotonically increasing over $[0, 1/2]$ and monotonically decreasing over $[1/2, 1]$.

While the number of clusters is typically limited to a few information granules, we can easily proceed with successive refinements of fuzzy sets. This can be done by splitting fuzzy clusters of the highest heterogeneity [18]. Let us assume that we have already constructed ‘ c ’ fuzzy clusters. Each of them can be characterized by the performance index

$$V_i = \sum_{k=1}^N u_{ik}^m \| \mathbf{x}_k - \mathbf{v}_i \|^2 \tag{23}$$

$i = 1, 2, \dots, c$. The higher the value of V_i , the more heterogeneous the i -th cluster. The one with the highest value of V_i – that is, the one for which we have $i_0 = \arg \max_i V_i$ – is refined by being split into two clusters. Denote the set of data associated with the i_0 -th cluster by $\mathbf{X}(i_0)$,

$$\mathbf{X}(i_0) = \{ \mathbf{x}_k \in \mathbf{X} \mid u_{i_0 k} = \max_i u_{ik} \} \tag{24}$$

We cluster the elements in $\mathbf{X}(i_0)$ by forming two clusters which leads to two more specific (detailed) fuzzy sets. This gives rise to a hierarchical structure of the family of fuzzy sets, as illustrated in Fig. 12. The relevance of this construct in the setting of fuzzy sets is that it emphasizes the essence of forming a hierarchy of fuzzy sets rather than working with a single-level structure of a large number of components whose semantics could not be retained.

The process of further refinement is realized in the same way by picking up the cluster of highest heterogeneity and splitting it into two consecutive clusters.

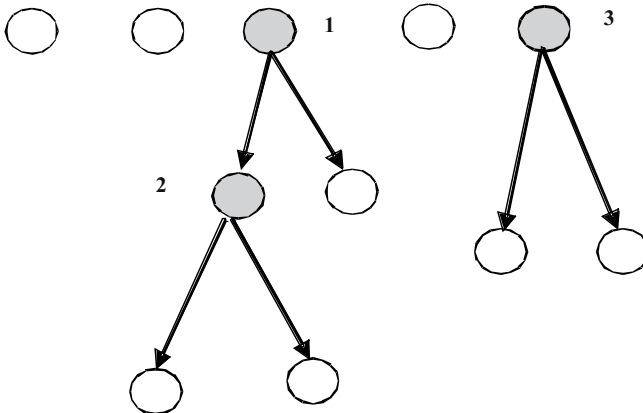


Fig. 12. Successive refinements of fuzzy sets through fuzzy clustering applied to the clusters of the highest heterogeneity (the numbers indicate the order of the splits)

It is worth emphasizing that the FCM algorithm is a highly representative method of membership estimation that profoundly relies on the use of experimental data. In contrast to some other data-driven techniques presented so far, FCM can easily cope with multivariable experimental data.

6 Main Design Guidelines

The considerations presented above give rise to a number of general guidelines supporting the development of fuzzy sets.

- *Highly visible and well-defined semantics of information granules.* No matter what the determination technique is, one has to become cognizant of the semantics of the resulting fuzzy sets. Fuzzy sets are interpretable information granules of a well-defined meaning and this aspect needs to be fully captured. Given this, the number of information granules has to be kept quite small with their number being restricted to 7 ± 2 fuzzy sets;
- *There are several fundamental views of fuzzy sets, and depending upon them we could consider the use of various estimation techniques* (for example, by accepting a horizontal or vertical view of fuzzy sets and adopting a pertinent technique);
- *Fuzzy sets are context-sensitive constructs and as such require careful calibration.* This feature of fuzzy sets should be treated as their genuine advantage. The semantics of fuzzy sets can be adjusted through shifting and/or adjusting their membership functions. The nonlinear transformation we introduced previously helps complete an effective adjustment by making use of some ‘standard’ membership functions. The calibration mechanisms being used in the design of the membership function are reflective of the human-centricity of fuzzy sets.
- *We have delineated between the two major categories of approaches supporting the design of membership functions, that is data-driven and expert (user)-based.* There are very different in the sense of the origin of the supporting evidence. Fuzzy clustering is a fundamental mechanism of the development of fuzzy sets. It is important in the sense that the method is equally suitable for one-dimensional and multivariable cases. The expert or simply user-based methods of membership estimation are important in the sense they offer some systematic and coherent mechanisms of elicitation of membership grades. With regard to consistency of the elicited membership grades, the pairwise estimation technique is of particular interest in that it provides well quantifiable mechanisms for the assessment of the consistency of the resulting membership grades. The estimation procedures underline some need for further development of higher types of constructs, such as fuzzy sets of type-2 or higher, and fuzzy sets of higher order that may be ultimately associated with constructs such as type-2 fuzzy sets or interval-valued fuzzy sets (this particular construct is visible when dealing

with the horizontal method of membership estimation, which comes with associated confidence intervals).

- *User-driven membership estimation uses the statistics of data yet in an implicit manner.* The granular terms – that is, fuzzy sets – come into existence once there is some experimental evidence behind them (otherwise there is no point forming such fuzzy sets).
- *The development of fuzzy sets can be carried out in a stepwise manner.* For instance, a certain fuzzy set can be further refined, if required by the problem at hand. This could lead to several more specific fuzzy sets that are associated with the fuzzy set formed at the higher level. Being aware of the complexity of the granular descriptors, we should resist the temptation of forming an excessive number of fuzzy sets at a single level, as such fuzzy sets could be easily lacking any sound interpretation.

7 Nonlinear Transformation of Fuzzy Sets

In many problems, we encounter a family of fuzzy sets defined in the same space. The family of fuzzy sets $\{A_1, A_2, \dots, A_c\}$ is referred to as referential fuzzy sets. To form a family of semantically meaningful descriptors of the variable at hand, we usually require that these fuzzy sets satisfy the requirements of unimodality, limited overlap, and coverage. Technically, all of these features are reflective of our intention to endow this family of fuzzy sets with some semantics. These fuzzy sets could be sought as generic descriptors (say, *small*, *medium*, *high*, and so forth) being described by some typical membership functions. For instance, those could be uniformly distributed triangular or Gaussian fuzzy sets with some ‘standard’ level of overlap between the successive terms (descriptors).

As mentioned, fuzzy sets are usually subject to some calibration depending upon the character of the problem at hand. We may use the same terms of *small*, *medium*, and *large* in various contexts yet their detailed meaning (that is, membership degrees) has to be adjusted (calibrated). For the given family of referential fuzzy sets, their calibration could be accomplished by taking the space $\mathbf{X} = [a, b]$ over which they are originally defined and transforming it into $[a, b]$ through some nondecreasing monotonic and continuous function $\Phi(\mathbf{x}, \mathbf{p})$ where \mathbf{p} is a vector of some adjustable parameters bringing the required flexibility of the mapping [24]. The nonlinearity of the mapping is such that some regions of \mathbf{X} are contracted and some of them are stretched (expanded), and in this manner capture the required local context of the problem. This affects the membership functions of the referential fuzzy sets $\{A_1, A_2, \dots, A_c\}$ whose membership functions are expressed now as $A_i(\Phi(\mathbf{x}))$. The construction of the mapping Φ is optimized, taking into account some experimental data concerning membership grades given at some points of \mathbf{X} . More specifically, the experimental data come in the form of the input–output pairs

$$\begin{aligned}
 &x_1 - \mu_1(1), \mu_2(1), \dots, \mu_c(1) \\
 &x_2 - \mu_1(2), \mu_2(2), \dots, \mu_c(2) \\
 &\quad \quad \quad \vdots \\
 &x_N - \mu_1(N), \mu_2(N), \dots, \mu_c(N)
 \end{aligned} \tag{25}$$

where the k -th input–output pair consists of x_k which denotes some point in \mathbf{X} , while $\mu_1(1), \mu_2(1), \dots, \mu_c(k)$ are the numeric values of the corresponding membership degrees. The objective is to construct a nonlinear mapping that is optimizing it with respect to the available parameters \mathbf{p} . More formally, we could translate the problem into the minimization of the following sum of squared errors

$$\begin{aligned}
 &\sum_{i=1}^c (A_i(\Phi(x_1; \mathbf{p}) - \mu_i(1))^2 + \sum_{i=1}^c (A_i(\Phi(x_2; \mathbf{p}) - \mu_i(2))^2 + \\
 &\quad \quad \quad \dots + \sum_{i=1}^c (A_i(\Phi(x_N; \mathbf{p}) - \mu_i(N))^2
 \end{aligned} \tag{26}$$

One feasible mapping comes in the form of the piecewise linear function shown in Fig. 13. Here the vector of the adjustable parameters \mathbf{p} involves a collection of the split points r_1, r_2, \dots, r_L and the associated differences D_1, D_2, \dots, D_L ; hence $\mathbf{p} = [r_1, r_2, \dots, r_L, D_1, D_2, \dots, D_L]$. The regions of expansion (or compression) are used to affect the referential membership functions and adjust their values given the experimental data.

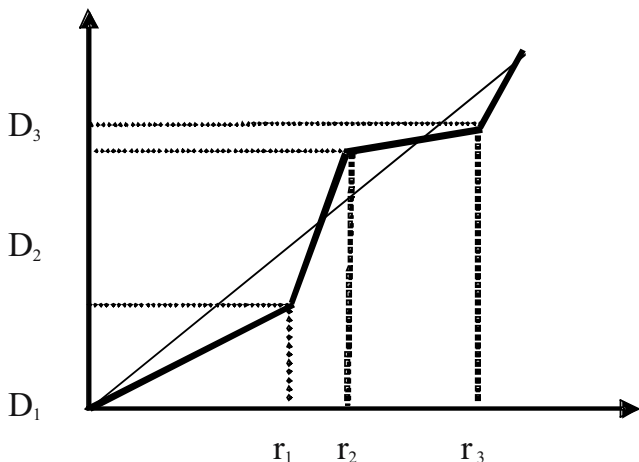


Fig. 13. A Piecewise linear transformation function Φ ; also shown is a linear mapping not affecting the universe of discourse and not exhibiting any impact on the referential fuzzy sets (the proposed piece-wise linear mapping is fully invertible)

Example

We consider some examples of nonlinear transformations of Gaussian fuzzy sets, Fig.15(a), through the piecewise linear transformations (here $L = 3$) shown in Fig.14.

Note that, as indicated in Fig. 15, some fuzzy sets or their segments become more specific, while the others are made more general and expanded over some regions of the universe of discourse.

Considering the same nonlinear mapping as before, two triangular fuzzy sets are converted into fuzzy sets described by piecewise membership functions, as illustrated in Fig. 16.

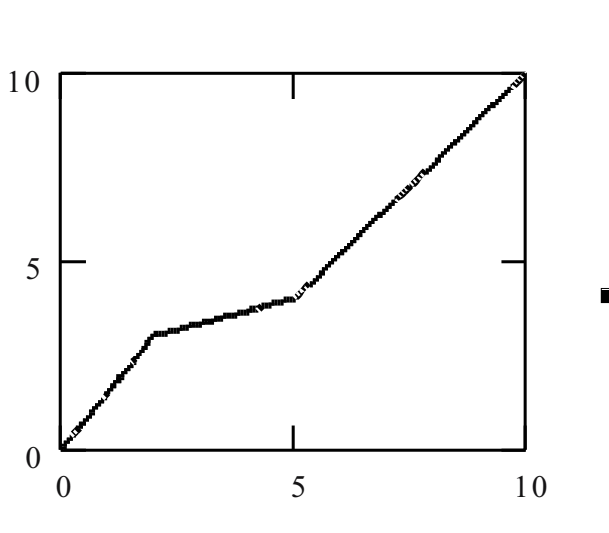


Fig. 14. An example of the piecewise linear transformation

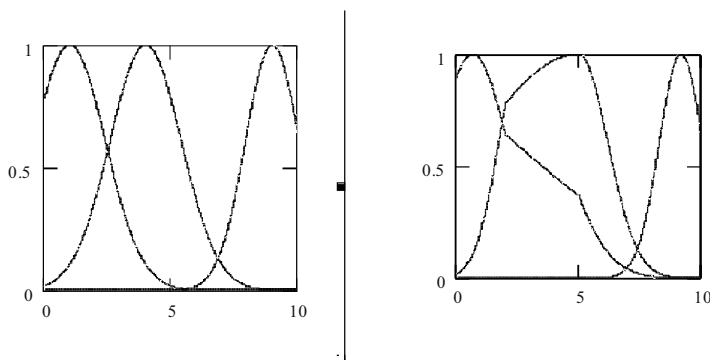


Fig. 15. Examples of original membership functions (a) and the resulting fuzzy sets (b) after the piecewise linear transformation

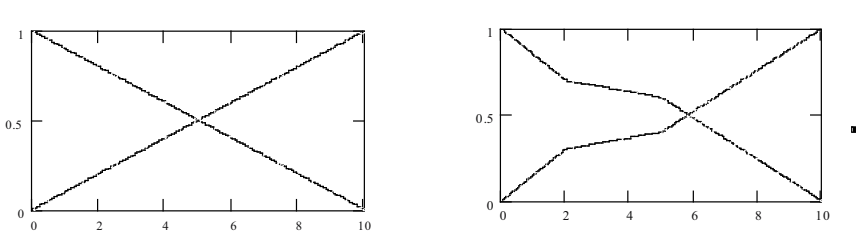


Fig. 16. Two triangular fuzzy sets along with their piecewise linear transformation

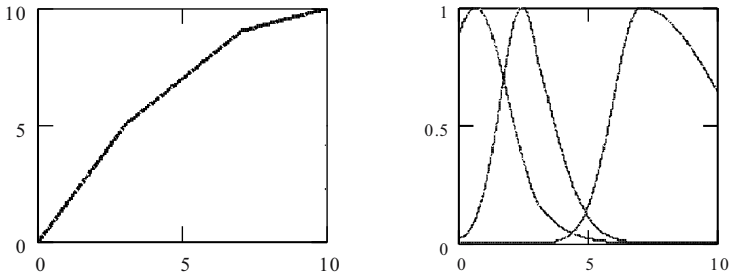


Fig. 17. The piecewise linear mapping (a) and the transformed Gaussian fuzzy set (b)

Some other examples of the transformation of fuzzy sets through piecewise mapping are shown in Fig. 17.

Information granules – and fuzzy sets in particular – capture pieces of domain knowledge and represent them in a formal setting of membership functions. The same concept – fuzzy set – can be perceived in many different ways, thus leading to the development of several specific membership functions of the same fuzzy set. For instance, the concept of *high* inflation treated as a fuzzy set and coming with its underlying membership function can be perceived by different human observers quite differently, and subsequently lead to several fuzzy sets. The corresponding membership functions could then be viewed as some modified or distorted versions of the original membership function.

Our objective is to reconcile these various perception views by forming some common view of the concept resulting in the form of some fuzzy set. In a nutshell, this leads to the formation of some optimal fuzzy set that takes into consideration the variety of perceptions. This reconciliation of perceptions is achieved through the formation of a certain optimization problem involving the individual fuzzy sets. The reconciliation of various perceptions could also involve fuzzy mappings. More specifically, some relationships between two spaces can be described by a family of fuzzy relations representing the

way in which they are perceived by various observers. Reconciliation of these relationships produces an optimal fuzzy relation being reflective of some essential commonalities of existing relationships. This optimal relation is a result of solution to the pertinent optimization problem.

8 Reconciliation of Information Granule Perception

Consider a certain fuzzy set A defined in some space (universe) X . It is perceived from different standpoints where each of these perceptions is quantified by ‘ c ’ human observers. In this regard, the observers provide some numeric confidence levels z_1, z_2, \dots, z_c where $z_i \in [0, 1]$. These specific confidence levels are translated into a form in which A becomes effectively perceived by these observers. Let us introduce the complements of $z_i, w_i = 1 - z_i$. The conjecture is that our perception transforms A into a less specific construct where a lack of confidence translates into a reduced level of specificity of A . This leads to the disjunctive model of perception in which the perceived information granule A comes with a membership function of the form $A_i(x) = A(x) s w_i$, where ‘ s ’ denotes a certain t-conorm (s-norm) [10] [32]. This model exhibits several interesting properties. As mentioned, the perceived granule cannot gain in its specificity but rather, depending upon the confidence, may exhibit some reduction of detail (lack of specificity). If the confidence about A is high – say $z_i = 1$ – then $w_i = 0$ and $A_i(x) = A(x)$ so the original fuzzy set is left unaffected. Conversely, if $z_i = 0$ then $w_i = 1$ and subsequently $A_i(x) = 1$. This new membership function demonstrates that the perceived information granule A is modeled as ‘unknown’ (being effectively the entire universe of discourse). In other words, the complete lack of confidence transforms A into the information granule which does not bear any specific meaning. The way of perceiving A from different standpoints (quantified by different values of the confidence values) is illustrated in Fig. 18. Note the collection of perceived information granules A_i resulting from A being affected by the associated levels of confidence $z_i(w_i)$. The result of the reconciliation comes through the optimization of the confidence level $z(z; \text{ where } z = 1 - w)$.

Given a family of fuzzy sets A_i , we are interested in reconciling the variety of the perception standpoints and on this basis construct a certain general (reconciled) viewpoint at A , say \tilde{A} whose membership function is expressed in the form

$$\tilde{A} = A(x)sw \tag{27}$$

where the weight $w \in [0, 1]$ is reflective of the reconciliation process. By adjusting the values of ‘ w ’ we can effectively capture the contributing components of the process. Graphically, we can envision the overall process described above as illustrated in Fig. 18.

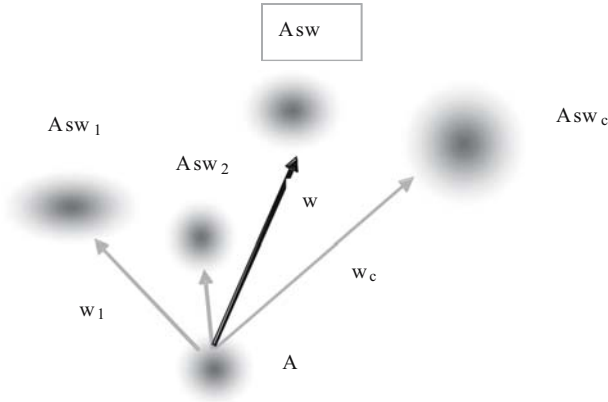


Fig. 18. Reconciliation of perceptions of information granule A

9 The Optimization Process

The above reconciliation can be transformed into the following optimization problem in which we are looking for the most suitable realization of perception so that all the individual views are taken into consideration. We introduce the following performance index

$$Q = \sum_{i=1}^c \int_X [A(x)sw_i - A(x)sw]^2 dx \tag{28}$$

The minimization of Q is carried out with respect to the values of ‘ w ’; $Min_w Q$. The necessary condition of the minimum is set as $\frac{dQ}{dw} = 0$. Proceeding with the detailed computation, we obtain

$$\begin{aligned} \frac{d}{dw} \sum_{i=1}^c \int_X [A(x)sw_i - A(x)sw]^2 dx = \\ -2 \sum_{i=1}^c \int_X [A(x)sw_i - A(x)sw] \frac{d(A(x)sw)}{dw} dx = 0 \end{aligned} \tag{29}$$

Let us denote by $\Phi(x, w)$ the derivative in the above expression, namely $\Phi(x, w) = \frac{dA(x)sw}{dw}$. Then we have

$$\sum_{i=1}^c \int_X [A(x)sw_i - A(x)sw] \Phi(x, w) dx = 0 \tag{30}$$

and

$$\sum_{i=1}^c \int_X A(x)sw_i = \sum_{i=1}^c \int_X [A(x)sw]\Phi(x, w)dx \tag{31}$$

Further calculations are possible once we have accepted a certain form of the t-conorm. For instance, let it be a probabilistic sum ($a \text{ s } b = a + b - ab$). This implies that the above expression for $\Phi(x, w)$ reads as

$$\frac{dA(x)sw}{dw} = \frac{d}{dw}(A(x) + w - A(x)w) = 1 - A(x) \tag{32}$$

Next we obtain

$$\sum_{i=1}^c \int_X A(x)sw_i = \sum_{i=1}^c \int_X A(x)(1 - A(x))dx + w \sum_{i=1}^c \int_X (1 - A(x))^2 dx \tag{33}$$

Let us rewrite the above expression by re-arranging and grouping terms and setting up its value to zero. We obtain

$$\int_X (1 - A(x))^2 \sum_{i=1}^c (w_i - w)dx = 0 \tag{34}$$

As the function under the integral is nonnegative, to make the value of $A(x)$ equal to zero, we require that the term $\sum_{i=1}^c (w_i - w)$ becomes zero, and this happens when ‘w’ is the average of the weights, $w = \frac{1}{c} \sum_{i=1}^c w_i$.

10 An Application of the Perception Mechanism to Rule-Based Systems

In a nutshell, fuzzy rule-based systems can be represented as a network of logic relationships (dependencies) between fuzzy sets existing in some input and output spaces. These fuzzy sets form the corresponding conditions and conclusions of the rules. As an illustration, let us consider the rule of the form

$$\begin{aligned} &\mathbf{if} && (input\#1 \text{ is } A1 & \mathbf{and} & input\#2 \text{ is } B1) \\ &\mathbf{or} && (input\#1 \text{ is } A2 & \mathbf{and} & input\#2 \text{ is } B2) \\ &\mathbf{or} && (input\#1 \text{ is } A3 & \mathbf{and} & input\#2 \text{ is } B3) \\ &\mathbf{then} && conclusion \text{ is } D \end{aligned} \tag{35}$$

Such a rule is directly mapped onto a network of AND and OR computing nodes. Furthermore, the nodes are equipped with some weights (connections) whose role is to calibrate the impact of the individual fuzzy sets standing in the rules and thus affect the results of the rule-based computation. The

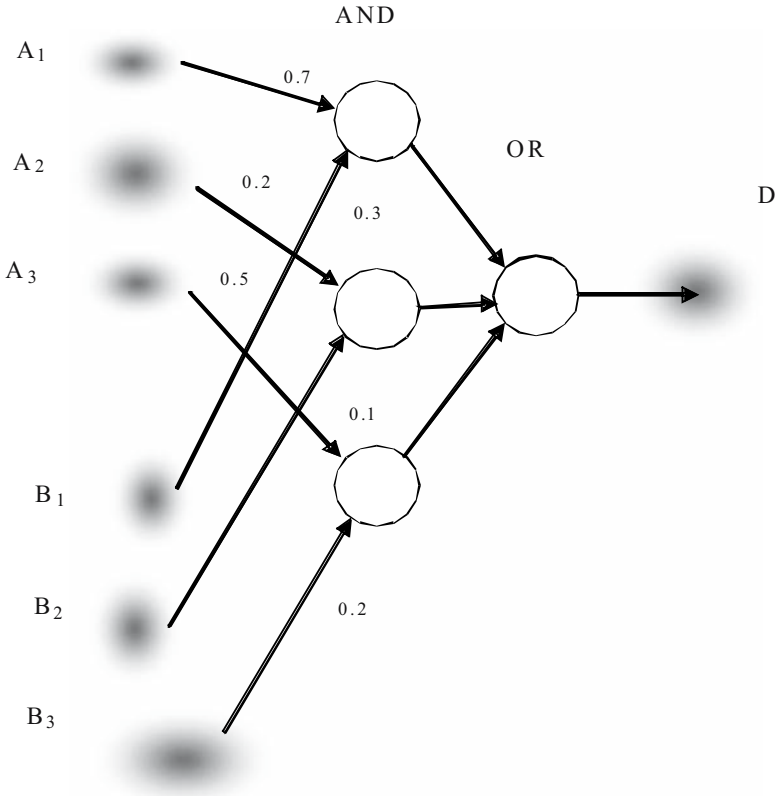


Fig. 19. A logic network realizing the rule presented by Eqn. (35) (note that the AND- and OR-nodes are endowed with some numeric connections)

network used to realize the above part of the rule-based system is illustrated in Fig. 19.

Alluding to the realization of the network illustrated in Fig. 19 composed of a collection of the fuzzy neurons with some specific numeric connection values [18, 19], we can write down the following detailed and numerically quantified **if...then** compound expression

$$\begin{aligned}
 & \text{if} \\
 & \quad \{[(A_1 \text{ or } 0.7) \text{ and } (B_1 \text{ or } 0.3)] \text{ and } 0.9\} \text{ or} \\
 & \quad \{[(A_2 \text{ or } 0.2) \text{ and } (B_2 \text{ or } 0.5)] \text{ and } 0.7\} \text{ or} \\
 & \quad \{[(A_3 \text{ or } 0.1) \text{ and } (B_3 \text{ or } 0.2)] \text{ and } 1.0\} \text{ or} \\
 & \text{then } D \tag{36}
 \end{aligned}$$

Each input fuzzy set (A_1, B_1, \dots) is ‘perceived’ by the corresponding AND-nodes through their weights (connections). There is an obvious analogy

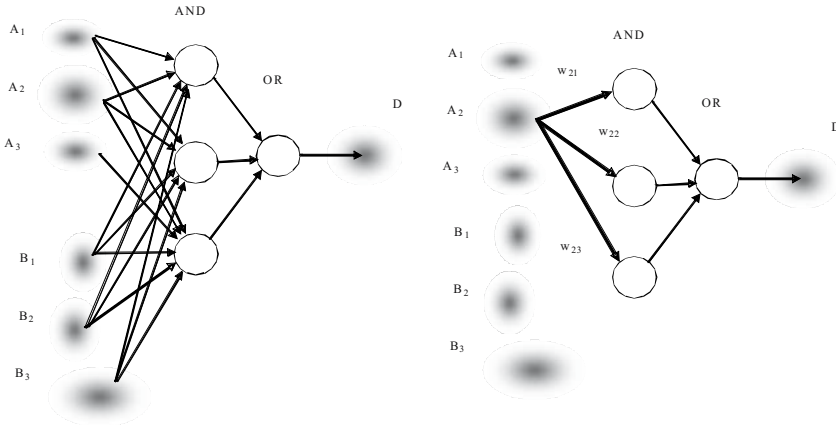


Fig. 20. Reconciliation of the impact of the input on individual AND-nodes through optimization of the corresponding connections

between the problem we formulated in Sect.9 and the formal description of the network formed by the logic neurons. By solving Eqn.(28) we arrive at a single value of the connection between a certain input and all the AND neurons, as displayed in Fig. 20.

By repeating the same reconciliation process for all input fuzzy sets, we arrive at the collection of optimized weights $w[1], w[2], w[3]$ (where $w[1]$ comes as a solution of the optimization task in which A_1 is involved, and so forth). They serve as a direct mechanism of establishing importance of the input information granule: the one of the lowest value of $w[i]$ points at the most relevant input (ii).

11 Reconciliation of Granular Mappings

So far we have discussed a way of reconciling perceptions of the same fuzzy set viewed from different standpoints. The problem under consideration is concerned with the reconciliation of relational mappings. The problem is formulated as follows: given relational mappings (fuzzy relations) R_1, R_2, \dots, R_c from space \mathbf{X} to \mathbf{Y} . More specifically, for given A in \mathbf{X} , the result of mapping comes in the form $A \circ R_1, A \circ R_2, \dots, A \circ R_c$, where \circ is a certain composition operator (in particular, it could be the one such as the sup-min, sup-t, inf-s, inf-min, or similar). Determine R such that it forms a reconciliation of the individual fuzzy relations. The reconciliation involves a certain fuzzy set in \mathbf{X} , denote it by A or may deal with a family of fuzzy sets in \mathbf{X} , say A_1, A_2, \dots, A_N . Formally speaking, we are looking for R defined in $\mathbf{X} \times \mathbf{Y}$ such that it results from a minimization of the following expression

$$Q = \sum_{i=1}^c \| A \circ R_i - A \circ R \|^2 \tag{37}$$

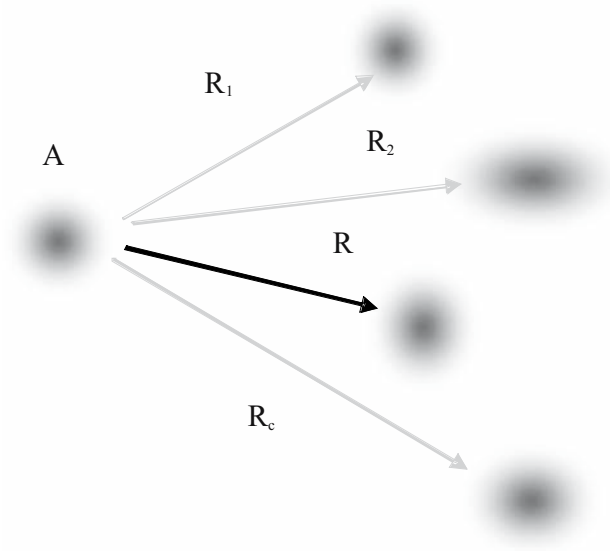


Fig. 21. Reconciliation of the relational mappings from \mathbf{X} to \mathbf{Y} (the result of this process is a certain fuzzy relation R)

for a given A in X or

$$Q = \sum_{l=1}^N \sum_{i=1}^c \| A_l \circ R_k - A_l \circ R \|^2 \tag{38}$$

for a family of A_l s. $\| \cdot \|$ denotes a certain distance function (in particular, it could come as the Euclidean one).

The essence of the reconciliation process of Eqn.(37) is visualized in Fig. 21. The optimization problem presented earlier in the form of Eqn. (11) can also be regarded as the reconciliation of relational models (expressed by fuzzy relations R_1, R_2, \dots, R_c) being completed in the context of some granular probes (fuzzy sets).

There is an interesting interpretation of this reconciliation process. Let \mathbf{X} and \mathbf{Y} denote a space of symptoms and diagnoses, respectively. The same fuzzy set of symptoms A leads to different interpretations (diagnoses) depending upon the fuzzy relations R_1, R_2, \dots, R_c , modeling the relational mappings expressed by different experts. Reconciliation is concerned with the development of the fuzzy relation that expresses the relationships between symptoms and diagnoses.

Minimization of Eqns.(37) and (38) can be realized for a given A once we have agreed upon the form of the composition operator. Similarly the optimization of the fuzzy relation depends upon the fuzzy set(s) available in \mathbf{X} .

In what follows, we consider finite spaces \mathbf{X} and \mathbf{Y} , $\mathbf{X} = \{x_1, x_2, \dots, x_n\}$, $\mathbf{Y} = \{y_1, y_2, \dots, y_m\}$. These cases make sense from a practical standpoint as quite often the spaces of symptoms and diagnoses are of finite dimensionality. Then the fuzzy sets and fuzzy relations can be represented in vector and matrix form. Considering the general case – Eqn. (38) – accepting a certain $s - t$ composition of A and R that is $\sum_{i=1}^n (A(x_i)tR(x_i, y_j))$, and adopting a Euclidean distance $\| \cdot \|$, we rewrite the performance index in the following format

$$\begin{aligned}
 Q &= \sum_{l=1}^N \sum_{k=1}^c \| A_l \circ R_k - A_l \circ R \|^2 \\
 &= \sum_{l=1}^N \sum_{k=1}^c \sum_{j=1}^m \left(\sum_{i=1}^n (A_l(x_i)tR_k(x_i, y_j)) - \sum_{i=1}^n (A_l(x_i)tR(x_i, y_j)) \right)^2 \quad (39)
 \end{aligned}$$

Minimization of Q is performed through a gradient-based optimization of R – in other words, a series of iterations (updates) of the values of R undertaken in successive iterations ($iter, iter + 1$) of form

$$R(iter + 1) = R(iter) - \alpha \nabla_R Q \quad (40)$$

where α is a positive learning rate and ∇ denotes a gradient of Q computed with respect to the fuzzy relation. Proceeding with the details, we carry out computations for all elements of the fuzzy relation which leads us to the expression

$$R(x_s, y_t)(iter + 1) = R(x_s, y_t)(iter) - \alpha \frac{\partial Q}{\partial R(x_s, y_t)(iter)} \quad (41)$$

$s = 1, 2, \dots, n; t = 1, 2, \dots, m.$

Realization of the learning scheme can be completed once the triangular norm and co-norm have been specified. In what follows, we consider the product and probabilistic sum. Then the derivative in Eqn. (38) can be expressed as follows

$$\frac{\partial Q}{\partial R(x_s, y_t)} = \sum_{l=1}^N \sum_{j=1}^m \left(\sum_{i=1}^n (A_l(x_i)tR_k(x_i, y_j)) - \sum_{i=1}^n (A_l(x_i)tR(x_i, y_j)) \right) \quad (42)$$

The derivative in the above expression can be written down in more detailed form

$$\frac{\partial}{\partial R(x_s, y_t)} (B_{l,s,t} + A_l(x_s)R(x_s, y_t) - B_{l,s,t}A_l(x_s)R(x_s, y_t)) \quad (43)$$

where $B_{l,s,t} = \sum_{i=1, i \neq s}^n (A_l(x_i)tR(x_i, y_j)).$

Finally after some rearrangements we obtain

$$\begin{aligned} \frac{\partial Q}{\partial R(x_s, y_t)} &= (B_{l,s,t} + A_l(x_s)R(x_s, y_t) - B_{l,s,t}A_l(x_s)R(x_s, y_t)) \\ &= A_l(x_s)(1 - B(l, s, t)) \end{aligned} \tag{44}$$

Example

As a numeric illustration, let us consider three fuzzy relations representing several mappings between \mathbf{X} and \mathbf{Y} ; here $\text{card}(\mathbf{X}) = \text{card}(\mathbf{Y}) = 4$,

$$R_1 = \begin{bmatrix} 1.0 & 0.7 & 0.3 & 0.0 \\ 0.9 & 1.0 & 0.4 & 0.1 \\ 0.4 & 0.8 & 1.0 & 0.6 \\ 0.0 & 0.2 & 0.7 & 1.0 \end{bmatrix} \quad R_2 = \begin{bmatrix} 1.0 & 0.7 & 0.8 & 0.0 \\ 0.9 & 1.0 & 0.4 & 0.1 \\ 0.4 & 0.8 & 1.0 & 1.0 \\ 0.0 & 0.5 & 0.2 & 0.3 \end{bmatrix} \quad R_3 = \begin{bmatrix} 0.5 & 0.7 & 0.1 & 1.0 \\ 0.9 & 1.0 & 0.4 & 0.1 \\ 0.4 & 0.8 & 1.0 & 0.6 \\ 1.0 & 0.9 & 0.6 & 0.0 \end{bmatrix} \tag{45}$$

The fuzzy sets $A_l, l = 1, 2, 3, 4$ defined in \mathbf{X} have the following membership functions

$$\begin{aligned} A_1 &= [1.0 \ 0.1 \ 0.3 \ 0.2] \\ A_2 &= [0.2 \ 1.0 \ 0.2 \ 0.0] \\ A_3 &= [0.0 \ 0.2 \ 0.9 \ 0.1] \\ A_4 &= [0.1 \ 0.3 \ 0.2 \ 1.0] \end{aligned} \tag{46}$$

We consider the s-t composition realized using the product and probabilistic sum, respectively. The learning rate α is equal to 0.05. Starting with random entries of the fuzzy relation R, the learning converged after about 30 learning steps (iterations) (Fig. 22).

The resulting fuzzy relation has the following entries

$$R = \begin{bmatrix} 0.85 & 0.7 & 0.41 & 0.38 \\ 0.9 & 1.0 & 0.4 & 0.09 \\ 0.4 & 0.8 & 1.0 & 0.74 \\ 0.34 & 0.53 & 0.51 & 0.45 \end{bmatrix} \tag{47}$$

By direct inspection, we note that R forms a certain compromise between the individual mappings between the spaces.

Let us now consider a different family of fuzzy sets $\{A_l\}$ that are used in the construction of the fuzzy relation

$$\begin{aligned} A_1 &= [1.0 \ 0.7 \ 0.5 \ 0.3] \\ A_2 &= [0.8 \ 1.0 \ 0.6 \ 0.0] \\ A_3 &= [0.4 \ 0.5 \ 0.9 \ 0.1] \\ A_4 &= [0.1 \ 0.6 \ 0.8 \ 1.0] \end{aligned} \tag{48}$$

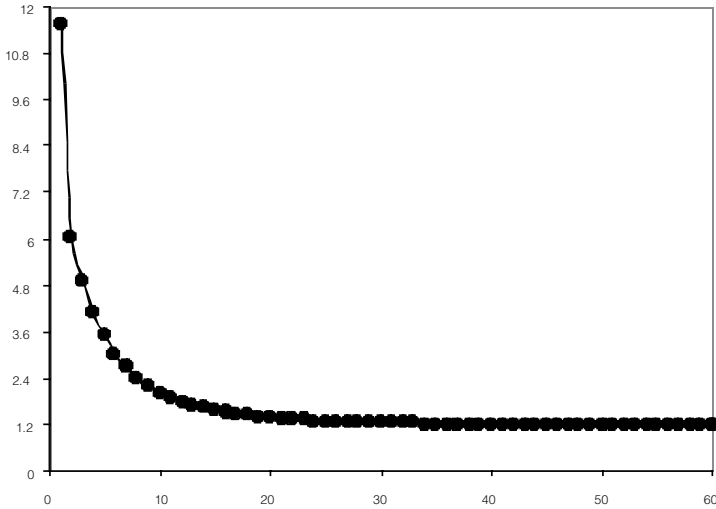


Fig. 22. Performance index Q in successive learning epochs

These fuzzy sets are far less specific than those used previously. They exhibit a substantial level of overlap (expressed in the values of the possibility measure computed for these pairs of fuzzy sets). The resulting fuzzy relation now has the form

$$R = \begin{bmatrix} 0.88 & 0.75 & 0.36 & 0.44 \\ 0.88 & 0.97 & 0.46 & 0.0 \\ 0.4 & 0.8 & 0.98 & 0.75 \\ 0.35 & 0.53 & 0.53 & 0.5 \end{bmatrix} \tag{49}$$

where $Q = 0.482$. The use of the more specific fuzzy sets A_l produces the fuzzy relation

$$R = \begin{bmatrix} 0.83 & 0.7 & 0.4 & 0.33 \\ 0.9 & 1.0 & 0.4 & 0.1 \\ 0.4 & 0.8 & 1.0 & 0.73 \\ 0.33 & 0.53 & 0.5 & 0.43 \end{bmatrix} \tag{50}$$

which comes with a higher value of the performance index, $Q = 1.92$. The input fuzzy sets with the singleton form of membership functions, that is

$$\begin{aligned} A_1 &= [1.0 \ 0.0 \ 0.0 \ 0.0] \\ A_2 &= [0.0 \ 1.0 \ 0.0 \ 0.0] \\ A_3 &= [0.0 \ 0.0 \ 1.0 \ 0.0] \\ A_4 &= [0.0 \ 0.0 \ 0.0 \ 1.0] \end{aligned} \tag{51}$$

give rise to the same fuzzy relation as before, however the performance index assumes a higher value ($Q = 2.78$).

We can conclude that the granular probes (input fuzzy sets) A_l play an important role in the reconciliation process and the formation of the fuzzy relation, however the results are not affected once the probes become sufficiently specific.

12 Conclusions

We have presented various approaches and algorithmic aspects of the design of fuzzy sets. The estimation of membership functions is a multifaceted problem and the selection of a suitable method relies on the choice of available experimental data and domain knowledge. For the user-driven approaches, it is essential to evaluate and flag the consistency of the results. This becomes well supported by the pairwise comparison method. Furthermore, we have introduced the idea of reconciliation of fuzzy set perception and granular mappings. It is shown how such reconciliation is expressed in the form of a certain optimization problem leading to the minimization of the connection (in the case of fuzzy sets) and the fuzzy relation (when we are concerned with the granular mapping formed by some fuzzy relations). We demonstrated the application of the approach to rule-based systems by demonstrating how it could be used in the assessment of relevance of input information granules in the rules. The role of reconciliation of the granular mappings is illustrated using problems of expressing fuzzy mappings between symptoms and diagnoses.

References

1. Bezdek JC (1981) *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, NY.
2. Bortolan G, Pedrycz W (2002) An interactive framework for an analysis of ECG signals. *Artificial Intelligence in Medicine*, 24(2): 109–132.
3. Buckley JJ, Feuring T, Hayashi Y (2001) Fuzzy hierarchical analysis revisited. *European J. Operational Research*, 129(1): 48–64.
4. Ciaramella A, Tagliaferri R, Pedrycz W, Di Nola A (2006) A Fuzzy relational neural network. *Intl. J. Approximate Reasoning*, 41(2): 146–163.
5. Civanlar MR, Trussell HJ (1986) Constructing membership functions using statistical data. *Fuzzy Sets and Systems*, 18(1): 1–13.
6. Chen MS, Wang SW (1999) Fuzzy clustering analysis for optimizing fuzzy membership functions. *Fuzzy Sets and Systems*, 103(2): 239–254.
7. Dishkant CH (1981) About membership functions estimation. *Fuzzy Sets and Systems*, 5(2): 141–147.
8. Dombi J (1990) Membership function as an evaluation. *Fuzzy Sets and Systems*, 35(1): 1–21.
9. Hong TP, Lee CY (1996) Induction of fuzzy rules and membership functions from training examples. *Fuzzy Sets and Systems*, 84(1): 389–404.
10. Klement E, Mesiar R, Pap E (2000) *Triangular Norms*. Kluwer Academic Publishers Dordrecht, The Netherlands.

11. Kulak O, Kahraman C (2005) Fuzzy multi-attribute selection among transportation companies using axiomatic design and analytic hierarchy process. *Information Sciences*, 170(2–4): 191–210.
12. Masson MH, Denoeux T (2006) Inferring a possibility distribution from empirical data. *Fuzzy Sets and Systems*, 157(3): 319–340.
13. Medaglia AL, Fang SC, Nuttle HLW, Wilson JR (2002) An efficient and flexible mechanism for constructing membership functions. *European J. Operational Research*, 139(1): 84–95.
14. Medasani S, Kim J, Krishnapuram R (1998) An overview of membership function generation techniques for pattern recognition. *Intl. J. Approximate Reasoning*, 19(3–4): 391–417.
15. Mikhailov L, Tsvetinov P (2004) Evaluation of services using a fuzzy analytic hierarchy process. *Applied Soft Computing*, 5(1): 23–33.
16. Miller GA (1956) The magical number seven plus or minus two: some limits of our capacity for processing information. *Psychological Review*, 63: 81–97.
17. Pedrycz W, Rocha A (1993) Hierarchical FCM in a stepwise discovery of structure in data. *Soft Computing*, 10: 244–256.
18. Pedrycz A, Reformat M (2006) Knowledge-based neural networks. *IEEE Trans. Fuzzy Systems*, 1: 254–266.
19. Pedrycz W (1993) Fuzzy neural networks and neurocomputations. *Fuzzy Sets and Systems*, 56: 1–28.
20. Pedrycz W (1994) Why triangular membership functions? *Fuzzy Sets and Systems*, 64: 21–30.
21. Pedrycz W (1995) *Fuzzy Sets Engineering*. CRC Press, Boca Raton, FL.
22. Pedrycz W, Valente de Oliveira J (1996) An algorithmic framework for development and optimization of fuzzy models. *Fuzzy Sets and Systems*, 80: 37–55.
23. Pedrycz W, Gudwin R, Gomide F (1997) Nonlinear context adaptation in the calibration of fuzzy sets. *Fuzzy Sets and Systems*, 88: 91–97.
24. Pedrycz W, Gomide F (1998) *An Introduction to Fuzzy Sets: Analysis and Design*. MIT Press, Cambridge, MA.
25. Pedrycz W (2001) Fuzzy equalization in the construction of fuzzy sets. *Fuzzy Sets and Systems* 119: 329–335 of fuzzy sets. *Fuzzy Sets and Systems*, 88: 91–97.
26. Pedrycz W (ed) (2001) *Granular Computing: An Emerging Paradigm*. Physica Verlag, Heidelberg, Germany.
27. Pedrycz W, Vukovich G (2002) On elicitation of membership functions. *IEEE Trans. Systems, Man, and Cybernetics – Part A*, 32(6): 761–767.
28. Pendharkar PC (2003) Characterization of aggregate fuzzy membership functions using Saaty's eigenvalue approach. *Computers and Operations Research*, 30(2): 199–212.
29. Saaty TL (1980) *The Analytic Hierarchy Process*. McGraw Hill, New York, NY.
30. Saaty TL (1986) Scaling the membership functions. *European J. Operational Research*, 25(3): 320–329.
31. Schweizer B, Sklar A (1983) *Probabilistic Metric Spaces* North-Holland, New York, NY.
32. Simon D (2005) H_∞ Estimation for fuzzy membership function optimization. *Intl. J. Approximate Reasoning* 40(3): 224–242.
33. Turksen IB (1991) Measurement of membership functions and their acquisition. *Fuzzy Sets and Systems*, 40(1): 5–138.

34. van Laarhoven PJM, Pedrycz W (1983) A fuzzy extension of Saaty's priority theory. *Fuzzy Sets and Systems*, 11(1-3): 199-227.
35. Yang CC, Bose NK (2006) Generating fuzzy membership function with self-organizing feature map. *Pattern Recognition Letters*, 27(5): 356-365.
36. Zadeh LA (1996) Fuzzy logic = computing with words. *IEEE Trans. Fuzzy Systems*, 4: 103-111.
37. Zadeh LA (1997) Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 90: 111-117.
38. Zadeh LA (2005) Toward a generalized theory of uncertainty (GTU) – an outline. *Information Sciences*, 172: 1-40.

Resources

1 Key Books

Pedrycz W, Gomide F (2007) *Fuzzy Systems Engineering: Toward Human-Centric Computing*. Wiley, New York, NY.

Pedrycz W, Gomide F (1998) *An Introduction to Fuzzy Sets: Analysis and Design*. MIT Press, Cambridge, MA.

Ross TJ (2004) *Fuzzy Logic with Engineering Applications (2nd ed)*. Wiley, New York, NY.

Zimmermann HJ (2005) *Fuzzy Set Theory and its Applications (4th ed)*. Springer-Verlag, Berlin.

2 Key Survey/Review Articles

Ekel P Y (2002) Fuzzy sets and models of decision making. *Computers & Mathematics with Applications*, 44(7): 863–875.

Frattale Mascioli FM, Rizzi A, Panella M, Martinelli G (2000) Scale-based approach to hierarchical fuzzy clustering. *Signal Processing*, 80(6): 1001–1016.

Kacprzyk J, Zadrozny S (2001) Computing with words in intelligent database querying: standalone and Internet-based applications. *Information Sciences*, 134(1–4): 71–109.

Guo P, Tanaka H (2001) Fuzzy DEA: a perceptual evaluation method. *Fuzzy Sets and Systems*, 119(1): 149–160.

Pei Z, Resconi G, Van Der Wal AJ, Qin K, Xu Y (2006) Interpreting and extracting fuzzy decision rules from fuzzy information systems and their inference. *Information Sciences*, 176(13): 1869–1897.

Wolff KE (2002) Concepts in fuzzy scaling theory: order and granularity. *Fuzzy Sets and Systems*, 132(1): 63–75.

Zadeh LA (2006) Generalized theory of uncertainty (GTU)-principal concepts and ideas. *Computational Statistics & Data Analysis*, 51(1) 15–46.

Zadeh LA (2005) Toward a generalized theory of uncertainty (GTU)—an outline. *Information Sciences*, 172(1–2): 1–40.

Zadeh LA (2004) A note on web intelligence, world knowledge and fuzzy logic. *Data & Knowledge Engineering*, 50(3): 291–304.

3 Organisations, Societies, Special Interest Groups

IEEE Computational Intelligence Society:

<http://iee-cis.org/>

International Fuzzy Systems Association (IFSA):

<http://www.cmplx.cse.nagoya-u.ac.jp/~ifsa/>

North American Fuzzy Information Processing Society (NAFIPS):

<http://www.cmplx.cse.nagoya-u.ac.jp/~ifsa/>

Japan Society for Fuzzy Theory and intelligent Informatics (SOFT):

<http://www.cmplx.cse.nagoya-u.ac.jp/~ifsa/>

European Society for Fuzzy Logic and Technology (EUSFLAT):

<http://www.cmplx.cse.nagoya-u.ac.jp/~ifsa/>

4 Research Groups

Refer to http://cswww.essex.ac.uk/staff/hhu/fuzzy_neural.html for a comprehensive list of research groups in fuzzy sets.

5 Key International Conferences/Workshops

IFSA Congress:

<http://www.cmplx.cse.nagoya-u.ac.jp/~ifsa/>

NAFIPS conferences:

<http://nafips.ece.ualberta.ca/nafips07/call.htm>

IEEE-Fuzz:

<http://fuzzieee2007.org/>

IEEE World Congress on Computational Intelligence:

<http://www.wcci2008.org/>

Evolutionary Multiobjective Design of Fuzzy Rule-Based Classifiers

Hisao Ishibuchi, Yusuke Nojima, and Isao Kuwajima

Department of Computer Science and Intelligent Systems, Graduate School
of Engineering, Osaka Prefecture University, Sakai, Osaka 599-8531, Japan,
hisaoi@cs.osakafu-u.ac.jp, nojima@cs.osakafu-u.ac.jp,
kuwajima@ci.cs.osakafu-u.ac.jp

1 Introduction

The main goal in classifier design has been accuracy maximization on unseen patterns [23]. A number of learning algorithms have been proposed to minimize the classification errors on training patterns in various fields such as neural networks [84], fuzzy systems [76] and machine learning [81]. It is well-known that neural networks are universal approximators of nonlinear functions [34, 35]. Since fuzzy systems are also universal approximators [70, 73, 92], we can design fuzzy rule-based classifiers that can correctly classify all training patterns. Such a fuzzy rule-based classifier, however, does not usually have high accuracy on test patterns, as shown in Fig. 1, where a typical accuracy-complexity tradeoff relation is depicted. We can decrease the error rate of classifiers on training patterns by increasing their complexity (for example, by increasing the number of fuzzy rules in fuzzy rule-based classifiers), as shown by the dotted curve in Fig. 1. The classification accuracy on test patterns is, however, degraded by increasing the complexity too much, as shown by the solid curve in Fig. 1. Such an undesirable deterioration in the classification accuracy on test patterns is known as ‘overfitting to training patterns’ [23]. Finding the optimal complexity with the maximum accuracy on test patterns (that is, S^* in Fig. 1) is one of the main research issues in machine learning, especially in the field of statistical learning theory [10].

The main advantage of fuzzy rule-based systems over other nonlinear systems such as neural networks is their linguistic interpretability [48]. That is, each fuzzy rule is easily understood by human users through linguistic interpretation. In this sense, fuzzy rule-based systems can be viewed as being transparent models (that is, ‘white-box’ models) whereas other nonlinear systems such as neural networks are usually ‘black-box’ models. Examples of fuzzy rules for function approximation problems are “If x_1 is *small* and x_2 is *small* then y is *large*” and “If x_1 is *large* and x_2 is *large* then y is *small*”.

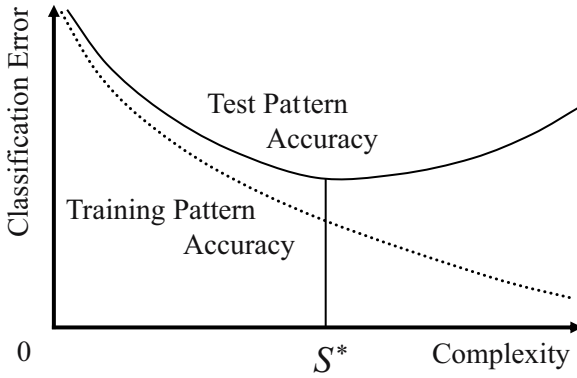


Fig. 1. A typical relation between the classification accuracy of classifiers and their complexity

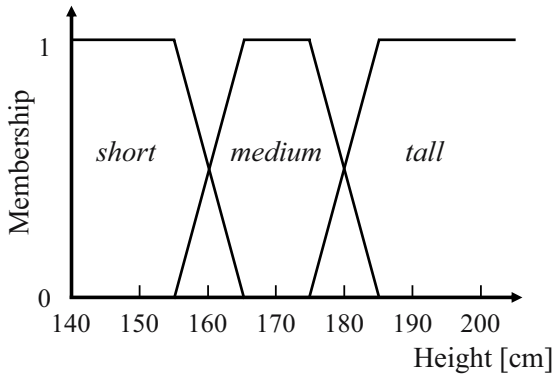


Fig. 2. Examples of membership functions of *short*, *medium* and *tall*

In these fuzzy rules, *small* and *large* are linguistic values. The meaning of each linguistic value is mathematically defined by its membership function in fuzzy logic. The membership function of each linguistic value is specified so that its meaning coincides with our intuition. For pattern classification problems, examples of fuzzy rules are “If x_1 is *small* and x_2 is *small* then Class 1” and “If x_1 is *large* and x_2 is *large* then Class 2”.

An example of fuzzy discretization is shown in Fig. 2, where membership functions of three linguistic values for the height (namely, *short*, *medium* and *tall*) are depicted. Of course, the definition of each membership function depends on the situation. When we talk about basketball players, we may have totally different membership functions from Fig. 2. Moreover we do not have the same set of membership functions even when the situation is clearly specified (for instance, the height of an NBA basketball player). Each

of us may have a different set of membership functions. Nevertheless we can usually communicate with each other using linguistic values in everyday situations without causing any serious inconvenience. These observations suggest that we may be able to easily understand fuzzy rules with linguistic values. That is, fuzzy rule-based systems have high interpretability. Of course, fuzzy rules with linguistic values are not always appropriate for knowledge representation. In some cases, interval discretization is much more appropriate. For example, consider the following knowledge: “People under 20 are not allowed to smoke”. In this case, interval discretization is appropriate since no fuzziness is involved in this knowledge.

A large number of learning algorithms were proposed during the 1990s to improve the accuracy of fuzzy rule-based systems by adjusting the membership function of each linguistic value. Some techniques use neural network learning algorithms [84] to fine-tune the membership function of each linguistic value [1, 33, 60, 76]. Others use genetic algorithms [26, 31] to optimize fuzzy rule-based systems [15, 16, 67]. Whereas the accuracy of fuzzy rule-based systems is significantly improved by such a learning algorithm, their interpretability is often degraded. This is mainly because we cannot simultaneously perform accuracy maximization and complexity minimization, due to the existence of the accuracy-complexity tradeoff relation in the design of fuzzy rule-based systems as shown in Fig. 1. In the field of fuzzy rule-based systems, accuracy-complexity tradeoff is often referred to as ‘interpretability-accuracy tradeoff’ [7, 8]. In this Chapter, we use these two terms interchangeably.

In order to find interpretable fuzzy rule-based systems with high accuracy, some approaches in the late 1990s took into account the existence of the accuracy-complexity tradeoff relation [52, 64, 66, 83, 86, 87]. In these approaches, an accuracy measure and a complexity measure are combined into a single scalar objective function to be optimized. A single fuzzy rule-based system, which can be viewed as a good tradeoff (that is, a good compromise) between accuracy and complexity, is obtained by these approaches on the accuracy-complexity tradeoff curve. Recently the existence of the accuracy-complexity tradeoff relation in the design of fuzzy rule-based systems has been widely recognized [7, 8, 48].

As is well-known in the field of multiobjective optimization [18, 74], it is very difficult to combine multiple objectives into a single scalar objective function. That is, an appropriate specification of a scalar objective function using multiple objectives is very difficult, whereas the finally obtained fuzzy rule-based system strongly depends on its specification. In order to avoid this difficulty, a multiobjective approach was proposed where various Pareto-optimal fuzzy rule-based systems were searched for along the accuracy-complexity tradeoff curve by an evolutionary multiobjective optimization (EMO) algorithm [42]. Since this study, a number of multiobjective approaches have been proposed to search for Pareto-optimal fuzzy rule-based systems [47, 50, 55, 63, 90, 91]. It is usually assumed in these approaches that a

single fuzzy rule-based system is chosen by a human user based on his/her preference from the obtained non-dominated alternatives. Recently multiobjective approaches have been used in various areas in machine learning [65].

In this Chapter, we explain EMO-based approaches to the multiobjective design of fuzzy rule-based classifiers. First we briefly explain fuzzy rules, their heuristic extraction from numerical data, and fuzzy reasoning for pattern classification problems. We also compare fuzzy and interval rules using illustrative numerical examples. Next we briefly explain multiobjective optimization and EMO algorithms. Then we explain two approaches to the multiobjective design of fuzzy rule-based classifiers. One is evolutionary multiobjective fuzzy rule selection and the other is multiobjective fuzzy genetics-based machine learning (GBML). Through computational experiments on data sets in the UCI Machine Learning repository (<http://www.ics.uci.edu/~mllearn/MLSummary.html>), we demonstrate the effectiveness of our multiobjective approaches to the design of fuzzy rule-based classifiers. Finally we indicate some future research directions and conclude the Chapter.

2 Fuzzy Rule-Based Classifiers

In this Section, we explain fuzzy rules, fuzzy rule extraction and fuzzy reasoning for pattern classification problems. We also show some characteristic features of fuzzy rule-based classifiers by comparing them with interval rule-based classifiers.

2.1 Pattern Classification Problems

Let us assume that we have m training patterns $\mathbf{x}_p = (x_{p1}, x_{p2}, \dots, x_{pn})$, $p = 1, 2, \dots, m$ from M classes in an n -dimensional pattern space. The classification of each training pattern is known (in other words, we have m labeled patterns). For simplicity of explanation, we assume that all attribute values have already been normalized into real numbers in the unit interval $[0, 1]$. Thus the pattern space of our classification problem is an n -dimensional unit hypercube $[0, 1]^n$. An example of such a pattern classification problem is shown in Fig. 3, where $m = 30$ (30 patterns), $n = 2$ (2 attributes), and $M = 3$ (3 classes).

2.2 Fuzzy Rules

Fuzzy rules for an n -dimensional pattern classification problem are written as

$$\text{Rule } R_q : \text{ If } x_1 \text{ is } A_{q1} \text{ and } \dots \text{ and } x_n \text{ is } A_{qn} \text{ then Class } C_q \quad (1)$$

where R_q is the label of the q th fuzzy rule, $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is an n -dimensional pattern vector, A_{qi} is an antecedent fuzzy set associated with a

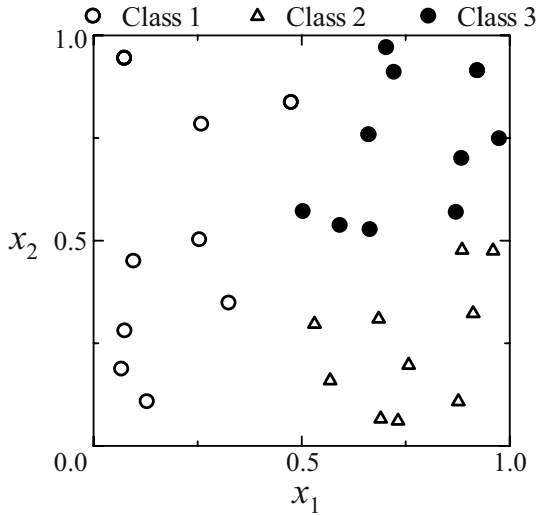


Fig. 3. Example pattern classification problem in the 2D pattern space $[0, 1]^2$

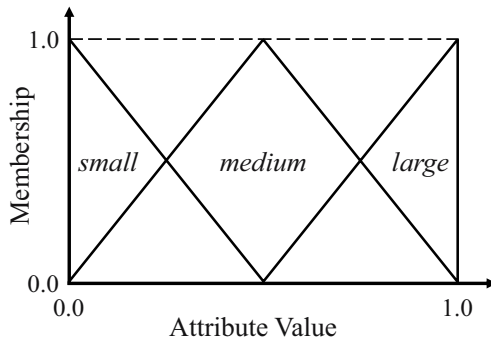


Fig. 4. Fuzzy partition of the unit interval into three linguistic terms *small*, *medium*, and *large*

linguistic term (in other words, A_{qi} is a linguistic value), and C_q is a consequent class. An example of such a fuzzy rule is “If x_1 is *small* and x_2 is *small* then Class 1”.

In some studies (for instance, [27]), a disjunctive combination of multiple linguistic terms is used as an antecedent fuzzy set A_{qi} , such as “If x_1 is *small* or *medium* and x_2 is *medium* or *large* then Class 1”. The disjunctive combination of all linguistic terms can be interpreted as *don't care*. For example, “*small* or *medium* or *large*” can be interpreted as *don't care* when the corresponding attribute is divided into the three linguistic terms *small*, *medium* and *large*, as shown in Fig. 4. In this case, the disjunctive combination “*small* or *medium*” can be interpreted as the negation of *large* (in other words,

“not *large*”). It should be noted that *don't care* can also be directly used as a special antecedent fuzzy set. We will further examine the use of *don't care* as a special antecedent fuzzy set in Sect. 4.

Fuzzy rules in Eqn. (1) have high interpretability because their antecedent part is specified by linguistic values such as *small* and *large*. Whereas fuzzy rules in Eqn. (1) have a single-dimensional fuzzy set A_{q_i} for each attribute in their antecedent part, it is also possible to use a multi-dimensional antecedent fuzzy set as follows:

$$\text{Rule } R_q : \text{ If } \mathbf{x} \text{ is } \mathbf{A}_q \text{ then Class } C_q \tag{2}$$

where \mathbf{x} is an n -dimensional pattern vector, and \mathbf{A}_q is an n -dimensional antecedent fuzzy set. Fuzzy rules of this type have often been used in clustering-based rule generation methods (for instance, [2, 3]). A typical example of a multi-dimensional fuzzy set \mathbf{A}_q is the ellipsoidal membership function illustrated in Fig. 5(b). When the multi-dimensional fuzzy set \mathbf{A}_q can be represented by a fuzzy vector as $\mathbf{A}_q = (A_{q1}, A_{q2}, \dots, A_{qn})$, there is no difference between fuzzy rules in Eqn. (1) and Eqn. (2). This situation is illustrated in Fig. 5(a) where the two-dimensional fuzzy set \mathbf{A}_q can be represented as $\mathbf{A}_q = (A_{q1}, A_{q2})$. On the other hand, the ellipsoidal fuzzy set \mathbf{A}_q in Fig. 5(b) cannot be represented as a fuzzy vector.

Fuzzy rules in Eqn. (2) with multi-dimensional antecedent fuzzy sets such as Fig. 5(b) usually have high classification accuracy. Their interpretability, however, is usually low due to the difficulty in linguistic interpretation of multi-dimensional antecedent fuzzy sets. One common approach for improving their interpretability is to project multi-dimensional antecedent fuzzy sets onto each axis of the pattern space [77, 79, 83, 86, 87]. Fuzzy rules in Eqn. (1) with single-dimensional antecedent fuzzy sets are derived from fuzzy rules

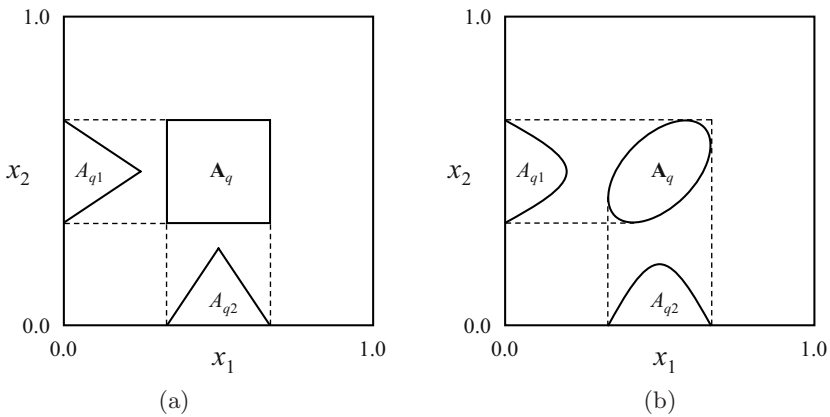


Fig. 5. Two-dimensional antecedent fuzzy sets: (a) 2D fuzzy vector; (b) ellipsoidal antecedent fuzzy set

in Eqn. (2) by the projection of their multi-dimensional antecedent fuzzy sets. Whereas such projection improves the interpretability of fuzzy rules, it degrades their classification accuracy. This is because information loss is involved in the projection, as we can see from Fig. 5(b). The information loss accompanying the projection of multi-dimensional antecedent fuzzy sets was discussed in [79].

Since the early 1990s [51], fuzzy rules of the following type have often been used for the design of fuzzy rule-based classifiers [48]:

$$\text{Rule } R_q : \text{ If } x_1 \text{ is } A_{q1} \text{ and } \cdots \text{ and } x_n \text{ is } A_{qn} \text{ then Class } C_q \text{ with } w_q \quad (3)$$

where w_q is a rule weight (that is, certainty grade). Usually w_q is a real number in the unit interval $[0, 1]$. As we will show later, rule weights have a large effect on the accuracy of fuzzy rule-based classifiers [44]. In other words, we can improve the accuracy of fuzzy rule-based classifiers by modifying only the rule weight of each fuzzy rule (without modifying the membership function of each antecedent fuzzy set). The use of rule weights, however, degrades the interpretability of fuzzy rule-based classifiers [78]. Several rule weight specification methods have been proposed [56, 89]. Their appropriate specification strongly depends on the choice of a fuzzy reasoning method for classifying new patterns. For the specification of rule weights, see [56]; their adjustment was discussed in [80].

It is also possible to use fuzzy rules with M consequent classes for an M -class pattern classification problem as follows:

$$\begin{aligned} \text{Rule } R_q : \text{ If } x_1 \text{ is } A_{q1} \text{ and } \cdots \text{ and } x_n \text{ is } A_{qn} \text{ then Class 1 with } w_{q1} \\ \text{and } \cdots \text{ and Class } M \text{ with } w_{qM} \end{aligned} \quad (4)$$

where w_{qh} is a weight of Class h ($h = 1, 2, \dots, M$). Usually w_{qh} is a real number in the unit interval $[0, 1]$. Fuzzy rules of this type are actually the same as those for a function approximation problem of an n -input and M -output function. More specifically, fuzzy rules in Eqn. (4) can be viewed as being the same as the following fuzzy rules with no rule weights for function approximation:

$$\begin{aligned} \text{Rule } R_q : \text{ If } x_1 \text{ is } A_{q1} \text{ and } \cdots \text{ and } x_n \text{ is } A_{qn} \text{ then } y_1 \text{ is } b_{q1} \\ \text{and } \cdots \text{ and } y_M \text{ is } b_{qM} \end{aligned} \quad (5)$$

where y_h is the h th output variable and b_{qh} is a consequent real number ($h = 1, 2, \dots, M$). Fuzzy rules in Eqn. (4) were examined in [13], together with those in Eqns. (1) and (3). The fuzzy rules in Eqn. (4) have more classification capability and less interpretability than those in Eqns. (1) and (3). In this Chapter, we use fuzzy rules in Eqn. (3), since they seem to be a good compromise between interpretability and accuracy.

2.3 Fuzzy Reasoning

Let S be a set of fuzzy rules – in other words, S is a fuzzy rule-based classifier. When an input pattern $\mathbf{x}_p = (x_{p1}, x_{p2}, \dots, x_{pn})$ is presented to the fuzzy rule-based classifier S , its compatibility grade with each fuzzy rule in S needs to be calculated. The following product and minimum operators are often used to calculate the compatibility grade:

$$\mathbf{A}_q(\mathbf{x}_p) = A_{q1}(x_{p1}) \cdot A_{q2}(x_{p2}) \cdot \dots \cdot A_{qn}(x_{pn}) \quad (6)$$

$$\mathbf{A}_q(\mathbf{x}_p) = \min\{A_{q1}(x_{p1}), A_{q2}(x_{p2}), \dots, A_{qn}(x_{pn})\} \quad (7)$$

where $\mathbf{A}_q(\mathbf{x}_p)$ is the compatibility of \mathbf{x}_p with the antecedent part $\mathbf{A}_q = (A_{q1}, A_{q2}, \dots, A_{qn})$ of the fuzzy rule R_q , and $A_{qi}(\cdot)$ is the membership function of the antecedent fuzzy set A_{qi} . In this Chapter, we use the product operator because it is more popular than the minimum operator in recently developed fuzzy rule-based systems.

First we explain fuzzy reasoning for pattern classification using fuzzy rules with no rule weights in Eqn.(1). When we use a winner-take-all scheme, the maximum compatibility grade of the input pattern \mathbf{x}_p for each class is calculated, as follows [51]:

$$\alpha_h(\mathbf{x}_p) = \max\{\mathbf{A}_q(\mathbf{x}_p) \mid C_q = h; R_q \in S\}, \quad h = 1, 2, \dots, M \quad (8)$$

The input pattern \mathbf{x}_p is assigned to the class with the maximum value of $\alpha_h(\mathbf{x}_p)$ over the M classes. In this case, \mathbf{x}_p can be viewed as being classified by the following winner rule R_W in the fuzzy rule-based classifier S :

$$\mathbf{A}_W(\mathbf{x}_p) = \max\{\mathbf{A}_q(\mathbf{x}_p) \mid R_q \in S\} \quad (9)$$

When multiple classes have the same maximum value (that is, when multiple fuzzy rules with different consequent classes have the same maximum value in Eqn. (9)), the classification of \mathbf{x}_p is rejected. Of course, it is possible to randomly assign \mathbf{x}_p to one of those classes with the maximum value of $\alpha_h(\mathbf{x}_p)$ in Eqn. (8).

Instead of the maximum compatibility in Eqn.(8), we can also define $\alpha_h(\mathbf{x}_p)$ by the total compatibility grade as follows [45]:

$$\alpha_h(\mathbf{x}_p) = \sum_{R_q \in S; C_q = h} \mathbf{A}_q(\mathbf{x}_p), \quad h = 1, 2, \dots, M \quad (10)$$

As in the case of Eqn. (8), \mathbf{x}_p is assigned to the class with the maximum value of $\alpha_h(\mathbf{x}_p)$ over the M classes. Whereas only the winner rule with the maximum compatibility grade is responsible for the classification result in the case of Eqn. (8), all compatible rules vote for their consequent classes in Eqn. (10).

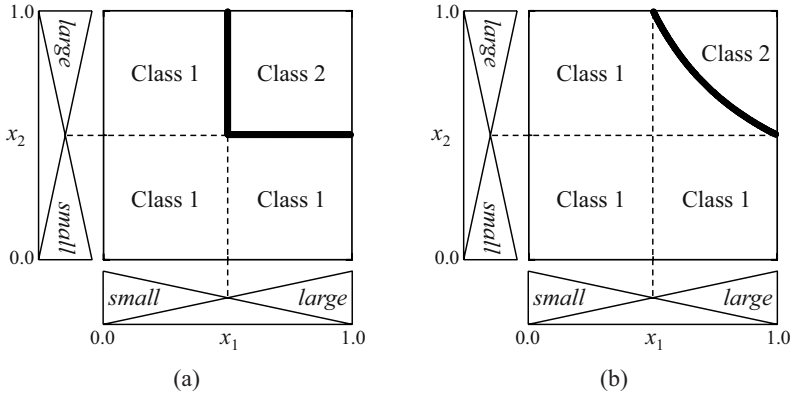


Fig. 6. Classification results by the four fuzzy rules with no rule weights: (a) the winner-take-all scheme in Eqn. (8); (b) the voting-based scheme in Eqn. (10)

For illustration, let us assume that we have the following four fuzzy rules:

- If x_1 is *small* and x_2 is *small* then Class 1**
- If x_1 is *small* and x_2 is *large* then Class 1**
- If x_1 is *large* and x_2 is *small* then Class 1**
- If x_1 is *large* and x_2 is *large* then Class 2**

where *small* and *large* are linguistic terms of antecedent fuzzy sets. These fuzzy rules are shown in Fig. 6. The bold lines in Fig. 6 show the classification boundaries between the two classes by the winner-take-all scheme in Eqn. (8) in Fig. 6(a) and by the voting-based scheme in Eqn. (10) in Fig. 6(b).

As we can see from Fig. 6(a), the classification boundary by the winner-take-all scheme is the same as the result by the lookup table approach with interval rules, as was pointed out by [71, 72]. On the other hand, we observe a nonlinear boundary in Fig. 6(b) when we use the voting-based scheme. One advantage of the winner-take-all scheme over the voting-based one is its high explanation capability for classification results. That is, we can easily explain why an input pattern is classified as a particular class since only a single fuzzy rule is responsible for the classification of each input pattern in the case of the winner-take-all scheme. In this Chapter, we use the winner-take-all scheme due to its high explanation capability for classification results.

For fuzzy rules with rule weights in Eqn. (3), the maximum weighted compatibility grade for each class is calculated as follows:

$$\alpha_h(\mathbf{x}_p) = \max\{\mathbf{A}_q(\mathbf{x}_p) \cdot w_q \mid C_q = h; R_q \in S\}, \quad h = 1, 2, \dots, M \quad (11)$$

In this case, the winning rule R_W for \mathbf{x}_p is identified as

$$\mathbf{A}_W(\mathbf{x}_p) \cdot w_W = \max\{\mathbf{A}_q(\mathbf{x}_p) \cdot w_q \mid R_q \in S\} \quad (12)$$

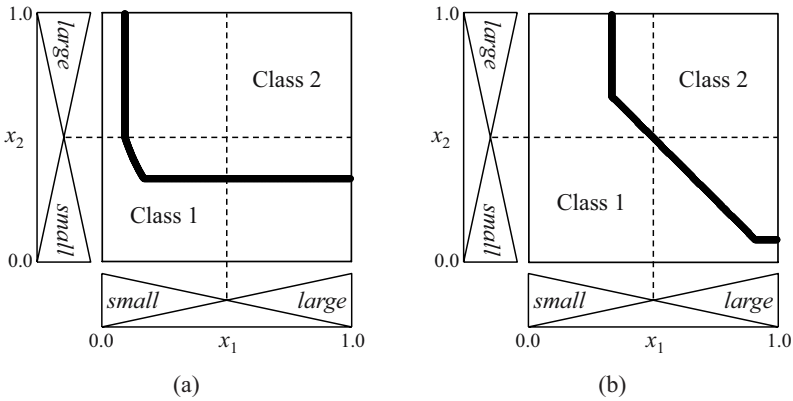


Fig. 7. Classification results by the four fuzzy rules with rule weights: (a) $(w_1, w_2, w_3, w_4) = (0.1, 0.1, 0.5, 1.0)$; (b) $(w_1, w_2, w_3, w_4) = (1.0, 0.5, 0.1, 1.0)$

Let us consider the following four fuzzy rules with rule weights:

- If** x_1 is *small* **and** x_2 is *small* **then** Class 1 with w_1
- If** x_1 is *small* **and** x_2 is *large* **then** Class 1 with w_2
- If** x_1 is *large* **and** x_2 is *small* **then** Class 1 with w_3
- If** x_1 is *large* **and** x_2 is *large* **then** Class 2 with w_4

Figure 7 shows two examples of classification boundaries obtained from these fuzzy rules by the winner-take-all scheme. We can see from a comparison between Fig. 6(a) and Fig. 7 that rule weights have a large effect on the classification results by a fuzzy rule-based classifier. Totally different classification boundaries were obtained in Fig. 7 using different values of rule weights.

2.4 Fuzzy Rule Extraction

Let us assume that the i th attribute is divided into K_i antecedent fuzzy sets ($i = 1, 2, \dots, n$). This means that the n -dimensional pattern space is divided into $K_1 \times K_2 \times \dots \times K_n$ fuzzy subspaces. [51] proposed the idea of determining the consequent class of a fuzzy rule by the majority class in the corresponding fuzzy subspace. First the total compatibility grade with the antecedent vector \mathbf{A}_q is calculated for each class using the given training patterns $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$ as follows:

$$\beta_h(\mathbf{A}_q) = \sum_{\mathbf{x}_p \in \text{Class } h} \mathbf{A}_q(\mathbf{x}_p), \quad h = 1, 2, \dots, M \tag{13}$$

Then the consequent class C_q is specified as the class with the maximum value of $\beta_h(\mathbf{A}_q)$ over the M classes for the antecedent vector \mathbf{A}_q . In this manner, we have a fuzzy rule “ $\mathbf{A}_q \Rightarrow C_q$ ”. When multiple classes have the same maximum value, we do not generate any fuzzy rules with the antecedent vector \mathbf{A}_q . When no training patterns are compatible with \mathbf{A}_q (that is, when

the right-hand side of Eqn. (13) is zero for all classes), neither do we generate any fuzzy rules with the antecedent vector \mathbf{A}_q .

This specification method can be explained as choosing the class with the maximum fuzzy conditional probability [58, 89]. The fuzzy conditional probability is specified for each class as follows:

$$\Pr(\text{Class } h \mid \mathbf{A}_q) = \frac{\sum_{\mathbf{x}_p \in \text{Class } h} \mathbf{A}_q(\mathbf{x}_p)}{\sum_{p=1}^m \mathbf{A}_q(\mathbf{x}_p)} \quad (14)$$

This fuzzy conditional probability is referred to as the confidence of the fuzzy rule “ $\mathbf{A}_q \Rightarrow \text{Class } h$ ” in the field of fuzzy data mining [32, 48, 57]. The confidence of “ $\mathbf{A}_q \Rightarrow \text{Class } h$ ” is written as

$$c(\mathbf{A}_q \Rightarrow \text{Class } h) = \frac{\sum_{\mathbf{x}_p \in \text{Class } h} \mathbf{A}_q(\mathbf{x}_p)}{\sum_{p=1}^m \mathbf{A}_q(\mathbf{x}_p)} \quad (15)$$

This formulation of the fuzzy confidence has often been used to evaluate a fuzzy rule together with the fuzzy support:

$$s(\mathbf{A}_q \Rightarrow \text{Class } h) = \frac{\sum_{\mathbf{x}_p \in \text{Class } h} \mathbf{A}_q(\mathbf{x}_p)}{m} \quad (16)$$

It should be noted that the same consequent class is obtained even when we use the fuzzy conditional probability in Eqn. (14), the fuzzy confidence in Eqn. (15) and the fuzzy support in Eqn. (16) instead of $\beta_h(\mathbf{A}_q)$ in Eqn. (13).

One of the most popular fuzzy rule extraction methods is that of [93]. Whereas this method was originally proposed for function approximation, it is also applicable to pattern classification with minor modifications. The basic idea of this method is to generate the most compatible fuzzy rule with each training pattern. Let $q(\mathbf{x}_p)$ be the index of the most compatible antecedent vector \mathbf{A}_q with the training pattern \mathbf{x}_p . Using $q(\mathbf{x}_p)$, we define $\mathbf{A}_q^*(\mathbf{x}_p)$ as follows:

$$\mathbf{A}_q^*(\mathbf{x}_p) = \begin{cases} \mathbf{A}_q(\mathbf{x}_p), & \text{if } q = q(\mathbf{x}_p), \quad q = 1, 2, \dots, K_1 \times K_2 \times \dots \times K_n \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

That is, $\mathbf{A}_q^*(\mathbf{x}_p)$ is the same as the compatibility grade $\mathbf{A}_q(\mathbf{x}_p)$ only when \mathbf{A}_q is the most compatible antecedent vector with \mathbf{x}_p among all $K_1 \times K_2 \times \dots \times K_n$ combinations of the antecedent fuzzy sets (it is assumed that K_i antecedent fuzzy sets are given for the i th attribute). The rule generation method of [93] can be used for pattern classification by specifying $\beta_h(\mathbf{A}_q)$ as follows:

$$\beta_h(\mathbf{A}_q) = \max\{\mathbf{A}_q^*(\mathbf{x}_p) \mid \mathbf{x}_p \in \text{Class } h\}, \quad h = 1, 2, \dots, M \quad (18)$$

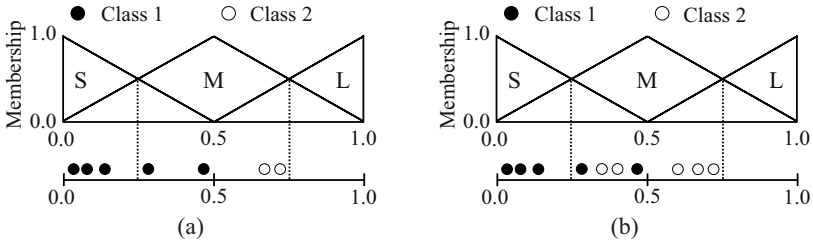


Fig. 8. Two data sets in the single-dimensional pattern space $[0, 1]$; the three antecedent fuzzy sets *small*, *medium* and *large* are denoted by S, M and L, respectively

The consequent class C_q is specified as the class with the maximum value of $\beta_h(\mathbf{A}_q)$ over the M classes.

We explain these two methods – namely the total compatibility method of [51] and the maximum compatibility method of [93] – using Fig. 8 with a single-dimensional pattern space $[0, 1]$. In Fig. 8, both methods choose Class 1 as the consequent class for the antecedent part “If x is *small*”. Class 1 is also chosen by both methods for “If x is *medium*” in Fig. 8(a). In Fig. 8(b), Class 1 is still chosen by the maximum compatibility method for “If x is *medium*”, while Class 2 is chosen by the total compatibility method for the same antecedent. In Fig. 8, no fuzzy rules with the antecedent part “ x is *large*” are generated by the maximum compatibility method while “If x is *large* then Class 2” is generated by the total compatibility method. As shown in Fig. 8, the maximum compatibility method is sensitive to a single training pattern with a high compatibility grade while the total compatibility method depends on all compatible patterns. In this Chapter, we use the total compatibility method in Eqn. (13) due to its robustness with noisy patterns.

It should be noted that we can generate “If x is *large* then Class 2” if we modify Eqn. (18) in the maximum compatibility method of [93] as follows:

$$\beta_h(\mathbf{A}_q) = \max\{\mathbf{A}_q(\mathbf{x}_p) \mid \mathbf{x}_p \in \text{Class } h\}, \quad h = 1, 2, \dots, M \quad (19)$$

In this case, the consequent class of each fuzzy rule is specified by the most compatible training pattern with its antecedent part. On the other hand, the consequent class has the largest sum of compatibility grades with its antecedent part in the total compatibility method of [51].

As we have already shown in Fig. 7, rule weights have a large effect on the classification performance of a fuzzy rule-based classifier. It has been reported that good results were obtained from the following specification of rule weights when we used the winner-take-all scheme [48, 56].

$$w_q = c(\mathbf{A}_q \Rightarrow \text{Class } C_q) - \sum_{h=1; h \neq C_q}^M c(\mathbf{A}_q \Rightarrow \text{Class } h) \quad (20)$$

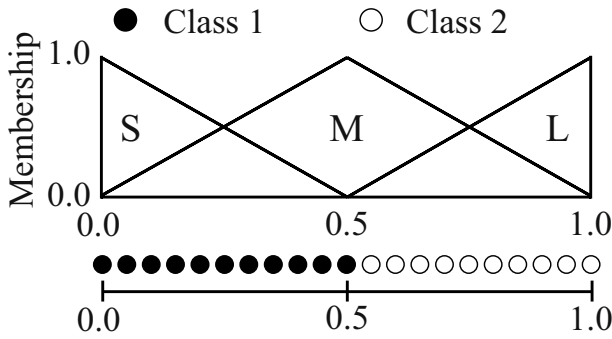


Fig. 9. Training patterns in the single-dimensional pattern space [0, 1]

For example, we can generate the following fuzzy rules from the training patterns in Fig. 9 using this specification method:

- If x is *small* then Class 1 with 1.00
- If x is *medium* then Class 1 with 0.05
- If x is *large* then Class 2 with 1.00

The classification boundary is calculated as $x = 0.524$ from the generated fuzzy rules using the winner-take-all scheme. This classification boundary seems to be acceptable in Fig. 9. On the other hand, the following fuzzy rules are obtained from the same training patterns if we use the confidence of each fuzzy rule *directly* as its rule weight:

- If x is *small* then Class 1 with 1.00
- If x is *medium* then Class 1 with 0.55
- If x is *large* then Class 2 with 1.00

In this case, the classification boundary is calculated as $x = 0.677$. This classification boundary seems to be inappropriate in Fig. 9.

2.5 Comparison Between Fuzzy and Interval Rules

In this Section, we explain some characteristic features of fuzzy rule-based classifiers by comparing them with interval rule-based classifiers. The main difference between fuzzy and interval rules is that each axis of the pattern space is divided into overlapping regions in the case of fuzzy rules as shown in Fig. 9. Thus an input pattern is covered by *multiple* fuzzy rules, whereas it is usually covered by a *single* interval rule. This is illustrated in Fig. 10, where an input pattern is covered by a single interval rule in Fig. 10(a), and by four fuzzy rules in Fig. 10(b).

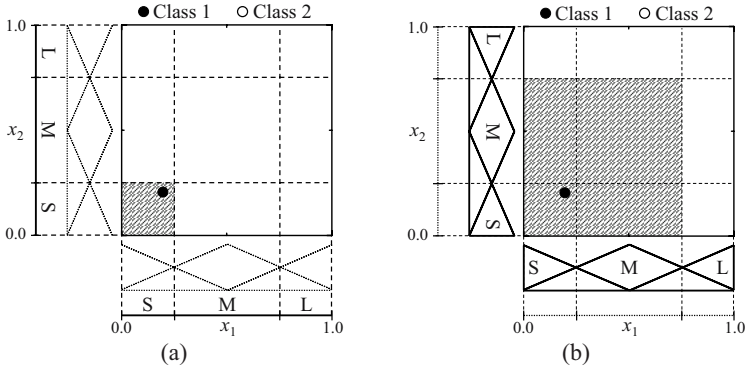


Fig. 10. A single interval rule (a), and four fuzzy rules (b), that cover an input pattern

For illustrative purposes, let us assume a very special situation where only a single Class 1 pattern in Fig. 10 is given as a training pattern. In this situation, a single interval rule with Class 1 consequent at the lower left corner is generated in Fig. 10(a). This interval rule classifies new patterns in the lower left cell as Class 1. The classification of new patterns in the other eight cells is rejected because there are no compatible rules with those patterns in the eight non-shaded cells in Fig. 10(a). On the other hand, four fuzzy rules with Class 1 consequent are generated from the same training pattern in Fig. 10(b). Almost all new patterns – except for those on the top and left edges of the pattern space – are classified as Class 1 by the four generated fuzzy rules. That is, more fuzzy rules are generated from the same training data than interval rules. As a result, a larger region of the pattern space is covered by the generated fuzzy rules than the case of interval rules.

The above-mentioned difference in the number of generated rules between interval and fuzzy rules is also illustrated in Fig. 11. In Fig. 11(a), six interval rules are generated from the 30 given training patterns. Classification of new patterns in the three shaded cells around the top-right corner is rejected in Fig. 11(a). On the other hand, nine fuzzy rules are generated from the same 30 training patterns in Fig. 11(b); all the pattern space is covered by at least one of the generated fuzzy rules in Fig. 11(b).

In Figure 11, we can also observe the difference in the shape of classification boundaries between interval and fuzzy rules. In the case of interval rules, classification boundaries are always piece-wise linear and axis-parallel, as shown in Fig. 11(a). The location of classification boundaries totally depends on the choice of threshold values for interval discretization of each axis of the pattern space. On the other hand, classification boundaries are not always axis-parallel in the case of fuzzy rules, as shown in Fig. 11(b). The location of classification boundaries depends on rule weights as well as fuzzy discretization of each

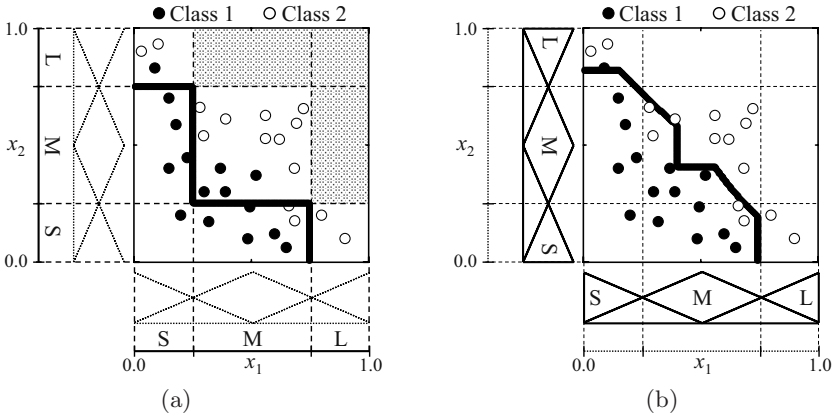


Fig. 11. Classification boundaries generated from the given training patterns: (a) interval rules; (b) fuzzy rules

axis of the pattern space. That is, the location of classification boundaries can be adjusted by rule weights as we have already illustrated in Fig. 7. For the learning of rule weights, see [48, 80]. Of course, the location of classification boundaries can also be adjusted by the tuning of the membership function of each antecedent fuzzy set [76, 77].

Better results were reported using fuzzy rules than interval rules when each axis of the pattern space was uniformly divided into antecedent intervals and antecedent fuzzy sets [48, 54]. Better results were also reported by fuzzy rules when training patterns were very sparse. Interval rules outperformed fuzzy rules only when the threshold values for interval discretization of each axis were carefully specified using a large number of training patterns in the computational experiments reported in [48, 54].

3 Evolutionary Multiobjective Optimization (EMO)

Before discussing evolutionary multiobjective design of fuzzy rule-based classifiers, we briefly explain genetic algorithms (GAs), multiobjective optimization (MO), and evolutionary multiobjective optimization (EMO) algorithms in this Section.

3.1 Genetic Algorithms (GAs)

A general outline of evolutionary algorithms, including genetic algorithms [26, 31], can be written as follows:

Algorithm 1 Generic Evolutionary Algorithm (EA)

Step 1. $P := \text{Initialize } (P)$
 Step 2.
while stopping condition not satisfied **do**
 (a) $P' := \text{Parent Selection } (P)$
 (b) $P'' := \text{Genetic Operations } (P')$
 (c) $P := \text{Replace } (P \cup P'')$

In Step 1, an initial population P is randomly generated in many cases. Whereas a binary string is often used in genetic algorithms to represent an individual (that is, to represent a solution of an optimization problem), other types of strings such as an integer string and a real number string are also used in evolutionary algorithms.

In Step 2(a), a set of pairs of strings (namely, a parent population P') is constructed from the current population P . One of the most popular selection schemes is binary tournament selection where two strings are randomly generated from the current population with replacement and the better one is chosen as a parent. A pair of parents is chosen by iterating this procedure twice. A pre-specified number of pairs of parents are chosen from the current population P in Step 2(a) to construct a parent population P' .

An offspring is generated from each pair of parents by crossover and mutation to construct an offspring population P'' in Step 2(b). Crossover is a genetic operation to generate an offspring from a pair of parents. One-point crossover and uniform crossover in Fig. 12 are often used in genetic algorithms with binary strings. In one-point crossover – Fig. 12(a) – a crossover point is randomly chosen to divide each parent into two substrings. A substring in one part of the offspring comes from one parent while the remaining part of the offspring comes from the other parent. On the other hand, one of the two parents is randomly chosen for each site (in other words, for each locus) of the offspring in uniform crossover. In Fig. 12(b), genes at the marked sites are inherited from Parent A to the offspring. Crossover is applied to each pair of parents with a pre-specified crossover probability. The crossover probability is usually specified as a real number in the interval $[0.5, 1.0]$. When crossover

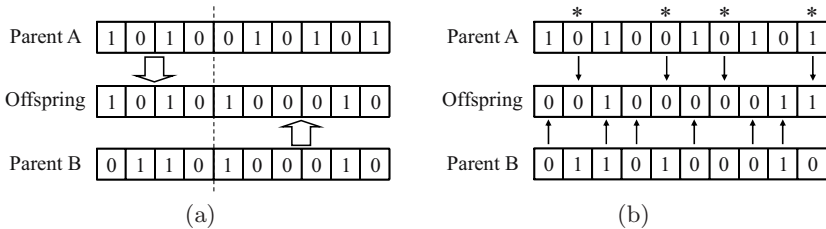


Fig. 12. Typical crossover operations in genetic algorithms with binary strings: (a) one-point crossover; (b) uniform crossover

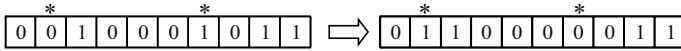


Fig. 13. Bit-flip mutation

is not applied, one of the two parents is randomly chosen and handled as an offspring in the next operation (that is, mutation).

Mutation is another genetic operation, and is used to partially and randomly modify each offspring after crossover. Bit-flip mutation in Fig. 13 is usually used in genetic algorithms with binary strings. A bit value in each site is changed with a pre-specified mutation probability. A typical value of the mutation probability for binary strings is $1/N$, where N is the string length.

In Step 2(c), the next population is constructed from the current population P and the offspring population P'' . Various generation update schemes have been proposed in the literature. Usually a pre-specified number of elite strings (that is, the best strings) are inherited from the current population to the next population with no modifications. The remaining slots in the next population are filled with newly generated offspring. One extreme implementation is to use the offspring population P'' as the next population (namely, $P := P''$). In this case, all the strings in the current population are replaced with the newly generated offspring – in other words, the number of elite strings is zero. Thus the generation gap between the current and next populations is largest. Another extreme implementation is to replace the worst string in the current population with a single offspring (namely, $|P''| = 1$). In this case, the number of elite strings is the population size minus one ($|P| - 1$); thus the generation gap is smallest.

Steps 2(a)–(c) are iterated until a pre-specified termination condition is satisfied. Since better strings are selected in the parent selection phase in Step 2(a) and the generation update phase in Step 2(c), we can expect that the current population is gradually improved by iterating Steps 2(a)–(c).

3.2 Multiobjective Optimization (MO)

In general, a k -objective maximization problem can be written as:

$$\text{Maximize } \mathbf{f}(\mathbf{y}) = (f_1(\mathbf{y}), f_2(\mathbf{y}), \dots, f_k(\mathbf{y})) \text{ subject to } \mathbf{y} \in \mathbf{Y} \quad (21)$$

where $\mathbf{f}(\mathbf{y})$ is a k -dimensional objective vector, $f_i(\mathbf{y})$ is the i th objective to be maximized, \mathbf{y} is a decision vector, and \mathbf{Y} is a feasible region in the decision space.

If there exists a solution \mathbf{y}^* that optimizes all objectives, \mathbf{y}^* is said to be the absolutely optimal solution. In the case of the k -objective maximization problem in Eqn. (21), \mathbf{y}^* is the absolutely optimal solution if the following relations hold:

$$f_i(\mathbf{y}^*) = \max\{f_i(\mathbf{y}) \mid \mathbf{y} \in \mathbf{Y}\} \text{ for } i = 1, 2, \dots, k \quad (22)$$

In general, multiobjective optimization problems do not have such an absolutely optimal solution that is optimal with respect to all objectives. This is because some objectives usually conflict with each other. Thus a different concept of optimal solutions, which is defined based on a dominance relation between two solutions, is often used in the field of multiobjective optimization.

Let \mathbf{y} and \mathbf{z} be two feasible solutions of the multiobjective maximization problem in Eqn. (21). The solution \mathbf{z} is said to dominate the solution \mathbf{y} (that is, \mathbf{z} is better than \mathbf{y}) when the following relations hold:

$$\forall i, f_i(\mathbf{y}) \leq f_i(\mathbf{z}) \text{ and } \exists i, f_i(\mathbf{y}) < f_i(\mathbf{z}) \tag{23}$$

If there exists no feasible solution \mathbf{z} that dominates \mathbf{y} , \mathbf{y} is said to be a Pareto-optimal solution. In this case, \mathbf{y} is optimal in the sense that \mathbf{y} is not dominated by any other feasible solutions. The set of all Pareto-optimal solutions is the Pareto-optimal solution set. The image of the Pareto-optimal solution set onto the objective space is called the Pareto front. That is, the Pareto front is the Pareto-optimal solution set in the objective space.

3.3 Evolutionary Multiobjective Optimization (EMO)

Evolutionary multiobjective optimization (EMO) algorithms try to find a good solution set that well approximates the Pareto-optimal solution set. In Figure 14, we show an example of the search for the Pareto-optimal solution set by an EMO algorithm in a two-dimensional objective space. An important issue in the implementation of good EMO algorithms is to find a good balance between convergence improvement (that is, search direction

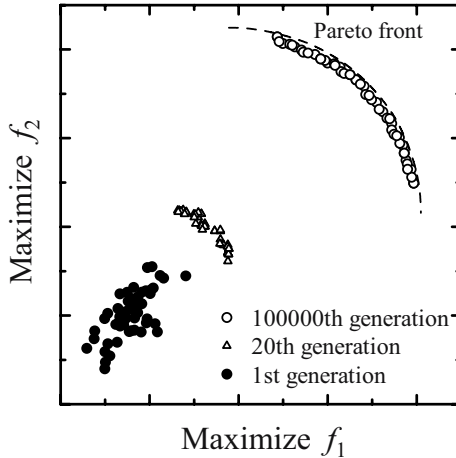


Fig. 14. An example search for the Pareto-optimal solution set by an EMO algorithm

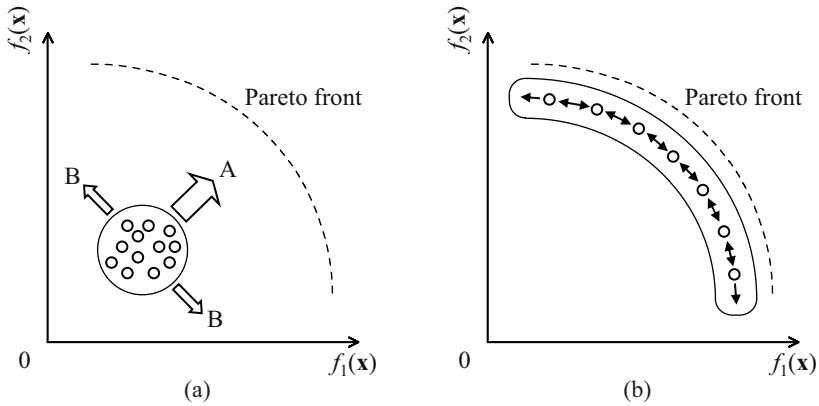


Fig. 15. (a) search direction A for convergence improvement, and search direction B for diversity improvement; (b) the search for uniformity improvement

A in Fig. 15(a)) and diversity improvement (that is, search direction B in Fig. 15(a)). High convergence ability is necessary for EMO algorithms to efficiently find Pareto-optimal or near Pareto-optimal solutions. On the other hand, diversity improvement mechanisms are necessary for EMO algorithms to find various solutions with a wide range of values of each objective. For details of the balance between convergence and diversity in EMO algorithms, see [53]. It is also desired to find a set of uniformly distributed solutions along the Pareto front (see Fig. 15(b)).

Since Schaffer's pioneering work in the mid 1980s [85], a large number of EMO algorithms have been proposed in the literature [11, 12, 18]. Currently EMO is one of the most active research areas in the field of evolutionary computation. Most EMO algorithms are based on the Pareto dominance relation in Eqn. (23) according to Goldberg's suggestion [26]. Pareto dominance relation is used to drive the population to the Pareto front (that is, search direction A in Fig. 15(a)). Most EMO algorithms also have some sort of diversity improvement mechanism to find a set of uniformly distributed solutions with a wide range of values of each objective. Diversity improvement mechanisms are used to widen the population (that is, search direction B in Fig. 15(a)) and uniformly distribute each solution (see Fig. 15(b)). Elitism has been used in recently developed EMO algorithms because its importance was clearly demonstrated by [94]. SPEA [94], PAES [69] and NSGA-II [19] are well-known and frequently-used elitist EMO algorithms. In some studies, local search has been incorporated into EMO algorithms in order to improve their search ability especially for combinatorial optimization problems [41, 59, 61].

In this Chapter, we use NSGA-II (fast elitist non-dominated sorting genetic algorithm) [19] due to its simplicity, popularity and high search ability. NSGA-II may be the most frequently-used and well-known EMO algorithm in

the literature. *NSGA-II* has the same basic framework as the genetic algorithms in Sect. 3.1. That is, *NSGA-II* iterates selection, genetic operations and generation update until a pre-specified termination condition is satisfied. Whereas a scalar fitness function is used in standard single-objective genetic algorithms, multiple objectives are used to evaluate each solution in EMO algorithms. For example, *NSGA-II* uses non-dominated sorting as the primary criterion and a crowding distance as the secondary criterion in binary tournament selection and generation update.

Non-dominated sorting in *NSGA-II* is performed in the following manner to evaluate each solution in the current population using multiple objectives. First, Rank 1 is assigned to those solutions that are not dominated by any other solutions in the current population (more specifically, Rank 1 is assigned to all non-dominated solutions in the current population). Rank 2 is assigned to all non-dominated solutions among solutions with no ranks. This ranking procedure is iterated until ranks are assigned to all solutions in the current population. This procedure is illustrated in Fig. 16(a). Solutions with higher ranks are viewed as better solutions (that is, Rank 1 is the best, Rank 2 is the second, and so forth).

When two solutions have the same rank, a crowding distance is used as the secondary criterion to compare them with each other in *NSGA-II*. When we calculate the crowding distance of a solution, all solutions with the same rank as that solution are projected to each axis of the objective space. Then the distance between two adjacent solutions of the solution is calculated on each axis. The crowding distance is the sum of the distances between two adjacent solutions over all objectives (see Fig. 16(b), where $a + b$ is assigned to a solution with Rank 1). An infinitely large value is assigned to extreme solutions with the minimum or maximum objective value of at least one objective among solutions with the same rank as shown in Fig. 16(b).

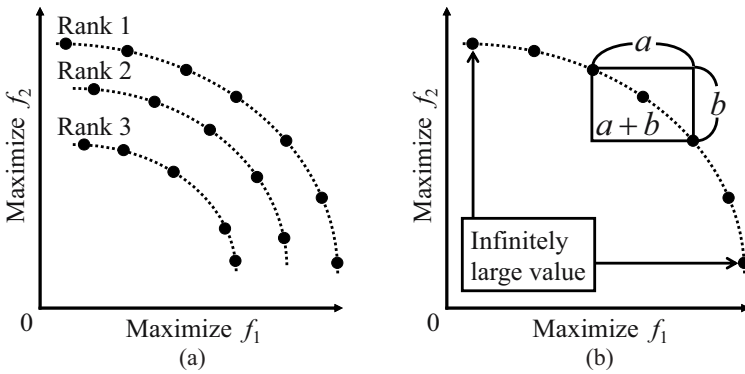


Fig. 16. Two important mechanisms in *NSGA-II*: (a) non-dominated sorting and (b) calculation of the crowding distance

In the parent selection phase of *NSGA-II*, two solutions are randomly drawn from the current population to choose a parent by binary tournament selection. If the two solutions have different ranks, the better solution with the higher rank is chosen as a parent. If they have the same rank, the better solution with a larger value of crowding distance is chosen as a parent. By iterating this binary tournament selection procedure, a pre-specified number of pairs of parents is selected. An offspring is generated from each pair of parents by genetic operations in our implementation of *NSGA-II* (it is also possible to generate two offsprings from each pair of parents). The number of newly generated offspring is the same as the population size in *NSGA-II*.

In the generation update phase of *NSGA-II*, the current and offspring populations are merged to generate an enlarged population. Non-dominated sorting is applied to the enlarged population. Then the crowding distance is calculated for each solution in the enlarged population. A pre-specified number of the best solutions is chosen from the enlarged population using the assigned rank of each solution as the primary criterion and the crowding distance as the secondary criterion. Since the number of offspring is the same as the population size, *NSGA-II* can be viewed as an elitist EMO algorithm with the $(\mu + \lambda)$ ES generation update mechanism, where $\mu = \lambda$.

The convergence of solutions to the Pareto front is realized in *NSGA-II* by choosing better solutions with respect to their ranks in the parent selection phase and the generation update phase. The diversity of solutions is maintained by choosing better solutions with respect to the crowding distance among solutions with the same rank. Elitism, which is realized by the $(\mu + \lambda)$ ES generation update mechanism, has positive effects on both the convergence and the diversity. For more details of each step of *NSGA-II*, see [18] and [19].

4 Two Approaches to Evolutionary Multiobjective Design of Fuzzy Rule-Based Classifiers

In this Section, we explain evolutionary multiobjective design of fuzzy rule-based classifiers. We assume that we have m training patterns $\mathbf{x}_p = (x_{p1}, x_{p2}, \dots, x_{pn})$, $p = 1, 2, \dots, m$ from M classes in an n -dimensional pattern space, which is an n -dimensional unit hypercube $[0, 1]^n$. Since we usually have no *a priori* information about an appropriate granularity (that is, the number of antecedent fuzzy sets) of fuzzy discretization for each attribute, we simultaneously use multiple fuzzy partitions with different granularities, as shown in Fig. 17, where S, MS, M, ML and L mean *small*, *medium small*, *medium*, *medium large* and *large*, respectively. The superscript of each fuzzy set shows the granularity of the fuzzy partition. Each of the 14 fuzzy sets is denoted by one of the 14 symbols: 1, 2, ..., 9, a, b, ..., e.

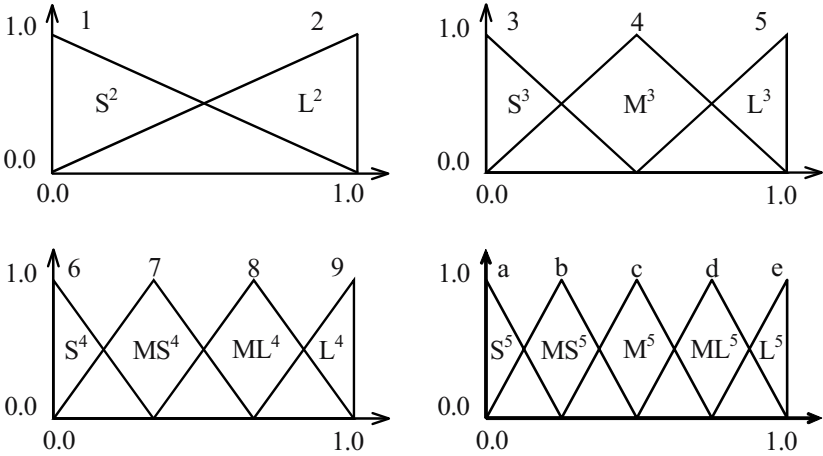


Fig. 17. The four fuzzy partitions used in our computational experiments

In addition to the 14 fuzzy sets of Fig. 17, we also use the domain interval $[0, 1]$ itself as an antecedent fuzzy set in order to represent a *don't care* condition. Thus we have the 15 possible fuzzy sets as the antecedent fuzzy set for each attribute. As a result, the total number of possible combinations of the antecedent fuzzy sets (namely, the total number of possible fuzzy rules) is 15^n for an n -dimensional pattern classification problem.

Of course, we can use other antecedent fuzzy sets with different types of membership functions (for instance, asymmetric, trapezoidal, Gaussian, and so forth) if they are appropriate for the particular pattern classification problem at hand. We can also use tailored antecedent fuzzy sets, tuned for each attribute, if they are available.

4.1 Problem Formulation

Let S be a set of fuzzy rules (S being a fuzzy rule-based classifier). We measure its accuracy by the number of correctly classified training patterns. On the other hand, we measure its complexity by the number of fuzzy rules in S and the total number of antecedent conditions of fuzzy rules in S . Thus we use the following three objectives in the formulation of the multiobjective design of fuzzy rule-based classifiers:

- $f_1(S)$: The number of training patterns correctly classified by S ,
- $f_2(S)$: The number of fuzzy rules in S ,
- $f_3(S)$: The total number of antecedent conditions of fuzzy rules in S (that is, the total rule length).

The number of antecedent conditions in each rule is often referred to as the rule length. Thus $f_3(S)$ is also viewed as the total rule length. It should

be noted that *don't care* conditions are not counted among the number of antecedent conditions (in other words, rule length) in the calculation of $f_3(S)$. For example, the rule length of the following fuzzy rule is not three but one: “If x_1 is *don't care* and x_2 is *small* and x_3 is *don't care* then Class 1 with 0.95”. This rule can be written as “If x_2 is *small* then Class 1 with 0.95” by omitting the two *don't care* conditions.

Multiobjective design of fuzzy rule-based classifiers is formulated as follows:

$$\text{Maximize } f_1(S), \text{ minimize } f_2(S), \text{ and minimize } f_3(S) \quad (24)$$

Our task in this Section is to find Pareto-optimal (or near Pareto-optimal) fuzzy rule-based classifiers of this three-objective optimization problem.

When the Pareto dominance relation – Eqn. (23) in Sect. 3 – is applied to this three-objective optimization problem, it is modified as follows: A rule set S_y is said to be dominated by another rule set S_z (that is, S_z dominates S_y : S_z is better than S_y) when all the following inequalities hold:

$$f_1(S_y) \leq f_1(S_z), f_2(S_y) \geq f_2(S_z), f_3(S_y) \geq f_3(S_z) \quad (25)$$

and at least one of the following inequalities holds:

$$f_1(S_y) < f_1(S_z), f_2(S_y) > f_2(S_z), f_3(S_y) > f_3(S_z) \quad (26)$$

Roughly speaking, when a rule set S_y has lower classification accuracy and higher complexity than another rule set S_z , S_y is said to be dominated by S_z .

4.2 Multiobjective Fuzzy Rule Selection

Fuzzy rule selection for the design of fuzzy rule-based classifiers was first formulated in [52] as the following maximization problem:

$$\text{Maximize } w_1 \cdot f_1(S) - w_2 \cdot f_2(S) \quad (27)$$

where w_1 and w_2 are pre-specified non-negative weights (which have nothing to do with rule weights, despite our using the same symbol w). This formulation was generalized as the following two-objective optimization problem by [42]:

$$\text{Maximize } f_1(S) \text{ and minimize } f_2(S) \quad (28)$$

This may be the first formulation of evolutionary multiobjective design of fuzzy rule-based systems. The two-objective formulation in Eqn. (28) was further generalized to the three-objective one in Eqn. (24) in [47], where an EMO-based rule selection algorithm and an EMO-based genetics-based machine learning (GBML) algorithm were compared with each other. A memetic EMO algorithm – a hybrid of EMO and local search – was used for multiobjective fuzzy rule selection in [55]. The three-objective formulation in Eqn. (24) was also used for non-fuzzy rule selection [49].

Evolutionary multiobjective fuzzy rule selection involves the following two-step method:

- Phase 1:* Candidate rule extraction,
- Phase 2:* Evolutionary multiobjective rule selection.

In the first phase, a large number of candidate fuzzy rules are extracted from training patterns using the heuristic rule extraction method explained in Sect. 2. The total number of possible fuzzy rules is 15^n for an n -dimensional pattern classification problem when we use the 14 fuzzy sets in Fig. 17 and *don't care* as antecedent fuzzy sets for each of the n attributes. For low-dimensional pattern classification problems with only a few attributes, we can examine all 15^n combinations of the antecedent fuzzy sets. Then all the generated fuzzy rules can be used as candidate rules. On the other hand, we cannot examine all 15^n combinations in the case of high-dimensional pattern classification problems (that is, when n is large). Thus a heuristic rule evaluation criterion has been used to choose only a tractable number of fuzzy rules as candidate rules [47, 55].

Let us assume that N candidate rules are extracted in the first phase. In the second phase, a subset S of the N candidate rules is represented by a binary string of length N as $S = s_1 s_2 \cdots s_N$, where $s_j = 1$ and $s_j = 0$ mean the inclusion and exclusion from S of the j th candidate rule, respectively. NSGA-II with such a binary string is used to search for Pareto-optimal rule sets of the three-objective optimization problem in Eqn. (24). Since rule sets are coded as binary strings, we can *directly* apply NSGA-II to the three-objective optimization problem using standard genetic operations, such as uniform crossover and bit-flip mutation.

Two tricks have been used to improve the search ability of NSGA-II for evolutionary multiobjective rule selection [47, 55]. One is the use of biased mutation where different mutation probabilities are assigned to the mutation from a 1 to a 0 and that from a 0 to a 1. For example, mutation from a 1 to a 0 may have a probability 0.1, whereas the mutation probability from a 0 to a 1 is 0.001. A larger probability is assigned to the mutation from a 1 to a 0 than that from a 0 to a 1 to efficiently decrease the number of fuzzy rules in each rule set (namely, to efficiently decrease the number of 1's in each string). The use of biased mutation is motivated by the fact that the number of selected fuzzy rules is much smaller than the number of candidate rules. For example, the number of 1's in a string S may be 10, whereas that of 0's is 1000. Suppose that standard non-biased mutation is applied to this string with probability 0.001. In this case, the average number of 1's mutated to a 0 is 0.01 (that is, 10×0.001), while that of 0's mutated to a 1 is 1 (that is, 1000×0.001). In other words, the 0.01 rule is removed from S and one rule included in S , on average. This means that standard non-biased mutation tends to increase the number of 1's in each string (more specifically, increase

the number of fuzzy rules in each rule set). If we use biased mutation with mutation probabilities 0.1 and 0.001, the average number of removed rules from S is the same as that of newly included rules in S on average (that is, $10 \times 0.1 = 1000 \times 0.001$). This means that one rule is removed from S and one rule is included in S on average. In this manner, biased mutation works as local search with no tendency toward larger rule sets, while standard non-biased mutation just increases the number of fuzzy rules. Advantages of biased mutation over standard non-biased mutation have been demonstrated in [52, 55].

The other trick is the removal of unnecessary fuzzy rules. Since we use the winner-take-all scheme, each training pattern is classified by a single winning rule. Thus some fuzzy rules may be chosen as winner rules for no training patterns. We can remove those fuzzy rules from a rule set S without degrading its classification accuracy (that is, without degrading the first objective $f_1(S)$). At the same time, the second objective $f_2(S)$ and third objective $f_3(S)$ are improved by removing unnecessary fuzzy rules. Thus we remove all fuzzy rules that are selected as winner rules for no training patterns from the rule set S . Removal of unnecessary fuzzy rules is performed for each string of the current population by changing the corresponding 1's to 0's. From the combinatorial nature of rule selection, some fuzzy rules with no contribution to pattern classification in one rule set may have a large contribution in another rule set. Thus the removal of unnecessary fuzzy rules is performed *locally* for each string in the current population (not *globally*, by removing those fuzzy rules from the set of candidate rules). The second and third objectives are calculated for each string after unnecessary fuzzy rules are removed. The removal of unnecessary fuzzy rules has a large effect on the decrease in the number of fuzzy rules. This leads to a decrease in the computation time for evolutionary multiobjective fuzzy rule selection [55].

We explain evolutionary multiobjective fuzzy rule selection using a simple numerical example. Let us assume that 20 training patterns are given in Fig. 18. Figure 18(a) shows the classification boundary by a fuzzy rule-based classifier with a 5×5 fuzzy grid. That is, we examined 5×5 combinations of the five antecedent fuzzy sets in the bottom-right fuzzy partition in Fig. 17 using the heuristic rule extraction method of Sect. 2. Then 23 fuzzy rules were generated, but which cannot correctly classify *all* the 20 training patterns as shown in Fig. 18(a). We also applied evolutionary multiobjective fuzzy rule selection to the same classification problem. First we examined 15×15 combinations of the 15 antecedent fuzzy sets using all the four fuzzy partitions in Fig. 17 as well as *don't care*. Using the heuristic rule extraction method in Sect. 2, 220 fuzzy rules were generated. All the generated fuzzy rules were used as candidate rules in evolutionary multiobjective fuzzy rule selection. We applied NSGA-II to the candidate rules using the following parameter specifications:

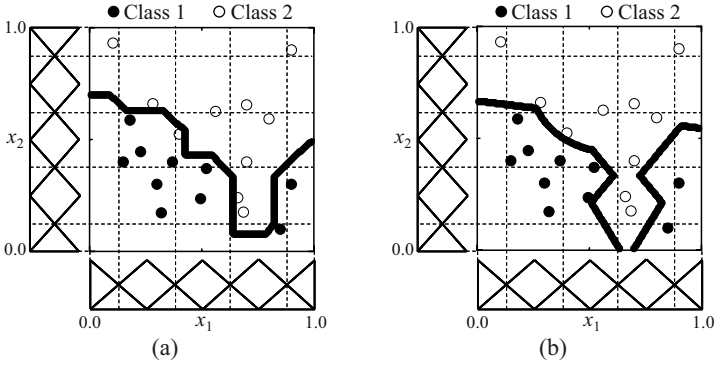


Fig. 18. Training patterns and classification results of two fuzzy rule-based classifiers: (a) 23 fuzzy rules in the 5×5 grid; (b) the 5 fuzzy rules of Fig. 19

Table 1. The obtained non-dominated rule sets

Number of rules	Total rule length	Classification rates (%)
0	0	0
1	1	50
2	2	85
3	3	90
4	4	95
5	6	100

Population size: $N_{pop} = 200$,
 Crossover probability: $P_C = 0.9$,
 Mutation probability: $P_M(1 \rightarrow 0) = 0.1$ and $P_M(0 \rightarrow 1) = 1/N$,
 Stopping (termination) condition: 2,000 generations.

The six non-dominated rule sets of Table 1 were obtained by NSGA-II. We can observe a clear tradeoff between the number of fuzzy rules and the accuracy of fuzzy rule-based classifiers in Table 1. All 20 training patterns can be correctly classified as shown in Fig. 18(b) by the obtained rule set with five fuzzy rules, which are shown in Fig. 19. Real numbers in parentheses in the last column of Fig. 19 show rule weights. Some fuzzy rules have *don't care* conditions denoted by DC in Fig. 19, which have a large effect on the decrease in the number of fuzzy rules. We can see from these experimental results that evolutionary multiobjective fuzzy rule selection is capable of finding a number of non-dominated rule sets with respect to complexity and accuracy.







	x_1	x_2	Consequent
R_1	DC		Class 1 (0.44)
R_2	DC		Class 1 (0.40)
R_3			Class 2 (0.75)
R_4		DC	Class 2 (0.15)
R_5		DC	Class 2 (0.49)

Fig. 19. The selected five fuzzy rules that can correctly classify all the 20 given training patterns

4.3 Multiobjective Fuzzy Genetics-Based Machine Learning

Multiobjective design of fuzzy rule-based classifiers can also be handled within the framework of genetics-based machine learning (GBML). Whereas binary strings were used in rule selection in Sect. 4.2, each fuzzy rule is represented by an integer string of length n using its n antecedent fuzzy sets in fuzzy GBML algorithms. An integer is assigned to each of the 15 antecedent fuzzy sets, as shown in Fig. 17. A rule set is represented by a concatenated integer string, where each substring of length n represents a single fuzzy rule.

GBML algorithms can be roughly classified into one of two approaches: One is the ‘chromosome = rule’ approach, and the other is the ‘chromosome = rule set’ approach. The former includes the Michigan approach [26, 31] and the iterative rule learning approach [9, 14, 27, 30]. The Michigan approach is often called classifier systems [4]. The ‘chromosome = rule set’ approach is often referred to as the Pittsburgh approach [88]. For details of fuzzy versions of these approaches, see [16, 29, 48].

Our multiobjective fuzzy GBML algorithm [50], which is implemented within the framework of *NSGA-II*, is basically the Pittsburgh approach. In other words, each rule set is represented by an integer string where each substring of length n denotes a single fuzzy rule.

The first step in our multiobjective fuzzy GBML algorithm is the generation of an initial population, as in many evolutionary algorithms. More specifically, N_{pop} integer strings are generated to construct an initial population. It was shown in [43, 46] that the search ability of a Michigan-style fuzzy GBML algorithm was drastically improved by generating initial fuzzy rules from training patterns in a heuristic manner. We use a similar heuristic

method to generate an initial population. First we randomly draw a pre-specified number of training patterns (say, N_{rule} patterns). Next we generate a fuzzy rule R_q from each training pattern $\mathbf{x}_p = (x_{p1}, \dots, x_{pn})$ by probabilistically choosing an antecedent fuzzy set A_{qi} for each attribute value x_{pi} from the 14 candidate fuzzy sets B_k ($k = 1, 2, \dots, 9, a, b, \dots, e$) in Fig. 17. Each candidate fuzzy set B_k has the following selection probability for the attribute value x_{pi} :

$$P(B_k) = \frac{B_k(x_{pi})}{\sum_{j=1}^e B_j(x_{pi})}, \quad k = 1, 2, \dots, 9, a, b, \dots, e \quad (29)$$

where $B_k(\cdot)$ denotes the membership function of the antecedent fuzzy set B_k . In Eqn. (29), the selection probability of each antecedent fuzzy set is proportional to its compatibility grade with the attribute value x_{pi} . All the n antecedent fuzzy sets of a fuzzy rule R_q are specified from the training pattern \mathbf{x}_p using Eqn. (29) for $i = 1, 2, \dots, n$. Then each antecedent fuzzy set of the generated fuzzy rule is replaced with *don't care* using a pre-specified probability $P_{\text{don't care}}$. In this manner, N_{rule} fuzzy rules are generated. An initial rule set consists of these fuzzy rules. By iterating this procedure, we generate N_{pop} initial rule sets (in other words, an initial population).

Parent selection is performed in the same manner as in NSGA-II. Crossover is applied to each pair of parents in the following manner. Since we try to decrease the number of fuzzy rules, string length is not constant. The number of fuzzy rules is adjusted by crossover. Let the selected rule sets for crossover be S_1 and S_2 (that is, S_1 and S_2 are a pair of parents). Some fuzzy rules are randomly selected from each parent to construct an offspring by crossover. The number of fuzzy rules to be inherited from each parent to the offspring is randomly specified. Let N_1 and N_2 be the number of fuzzy rules to be inherited from S_1 and S_2 , respectively. We randomly specify N_1 and N_2 in the intervals $[1, |S_1|]$ and $[1, |S_2|]$, respectively, where $|S_i|$ is the number of fuzzy rules in the rule set S_i . In order to generate an offspring, N_1 and N_2 fuzzy rules are randomly drawn from S_1 and S_2 , respectively. The generated offspring has $(N_1 + N_2)$ fuzzy rules. This crossover operation is applied to each pair of parents using a pre-specified crossover probability P_C . In this manner, the number of fuzzy rules in each rule set is varied by the crossover operation. In mutation, each antecedent fuzzy set is randomly replaced with a different one using a pre-specified mutation probability P_M .

The next population is constructed from the current and offspring populations in the same manner as NSGA-II. Our multiobjective fuzzy GBML algorithm searches for Pareto-optimal rule sets of the three-objective optimization problem in Eqn. (24) by iterating selection, crossover, mutation and generation update until a prespecified termination condition is satisfied.

It was demonstrated in [58] that the search ability of a Pittsburgh-style fuzzy GBML algorithm can be improved by combining it with a Michigan-style fuzzy GBML algorithm. Such a hybridization method was used in our multiobjective fuzzy GBML algorithm [50].

4.4 Computational Experiments on Test Problems

In this Section, we explain evolutionary multiobjective design of fuzzy rule-based classifiers through computational experiments on some well-known data sets in the UCI machine learning repository (<http://www.ics.uci.edu/~mllearn/MLSummary.html>). We show experimental results by evolutionary multiobjective fuzzy rule selection. We do not compare evolutionary multiobjective fuzzy rule selection and multiobjective fuzzy GBML, because their performance usually depends on parameter specifications and heuristic tricks incorporated therein; see [47] for a comparison of these two approaches.

We applied multiobjective fuzzy rule selection to the five data sets in Table 2. Computational experiments were performed in the following manner for each data set. First each data set was randomly divided into two subsets of the same size: training patterns and test patterns. Next we extracted 300 fuzzy rules as candidate rules for each class from training patterns using the following rule evaluation criterion, which was originally used in the SLAVE iterative rule learning algorithm [27]:

$$f(R_q) = s(\mathbf{A}_q \Rightarrow \text{Class } C_q) - \sum_{h=1; h \neq C_q}^M s(\mathbf{A}_q \Rightarrow \text{Class } h) \quad (30)$$

where $s(\cdot)$ is the support of a fuzzy rule. We only examined short fuzzy rules with three or less antecedent conditions for candidate rule extraction. That is, the best 300 fuzzy rules among those with three or less antecedent conditions were chosen as candidate rules for each class using the SLAVE criterion in Eqn. (30). Then we applied evolutionary multiobjective fuzzy rule selection to the extracted candidate rules using the same parameter specifications as in Sect. 4.2. A number of non-dominated rule sets were obtained by NSGA-II

Table 2. The five data sets used in our computational experiments

Data set	Attributes	Patterns	Classes
Breast W	9	683†	2
Glass	9	214	6
Heart C	13	297†	5
Iris	4	150	3
Wine	13	178	3

† = incomplete patterns with missing values are not included

for each data set. Finally we calculated the classification rates of each non-dominated rule set for training patterns and test patterns.

Experimental results are summarized in Figs. 20–24. In each figure, the left and right plots show classification rates of obtained non-dominated rule sets on training and test patterns, respectively. Some rule sets (for example, those with only a single fuzzy rule) are not shown in these figures because their classification rates are out of the range of the vertical axis of each plot. It should be noted that all the non-dominated rule sets in each plot were obtained from a single run of evolutionary multiobjective fuzzy rule selection.

We can observe similar tradeoff relations between accuracy and complexity for training patterns in the lefthand plots of Figs. 20–24 for all five data sets. That is, higher classification rates were obtained using more fuzzy rules. On the other hand, a different accuracy-complexity tradeoff relation is observed

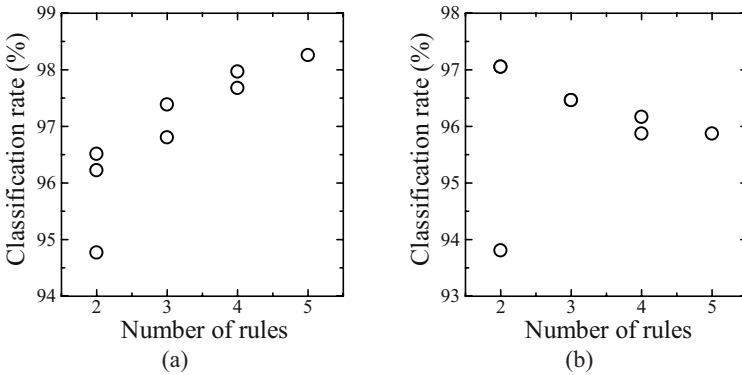


Fig. 20. Obtained non-dominated rule sets for the Wisconsin breast cancer data (Breast W): (a) training data accuracy; (b) test data accuracy

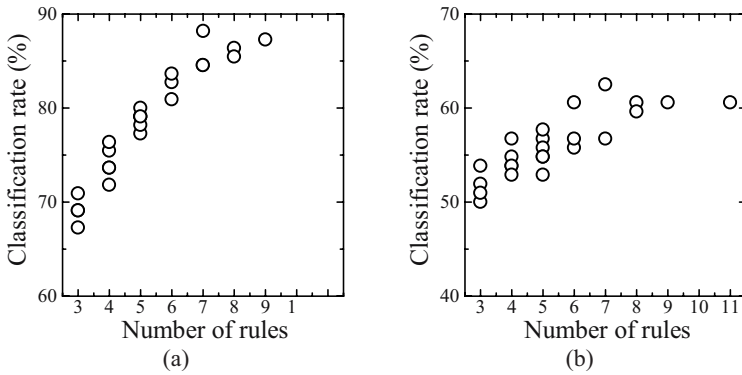


Fig. 21. Obtained non-dominated rule sets for the glass data (Glass): (a) training data accuracy; (b) test data accuracy

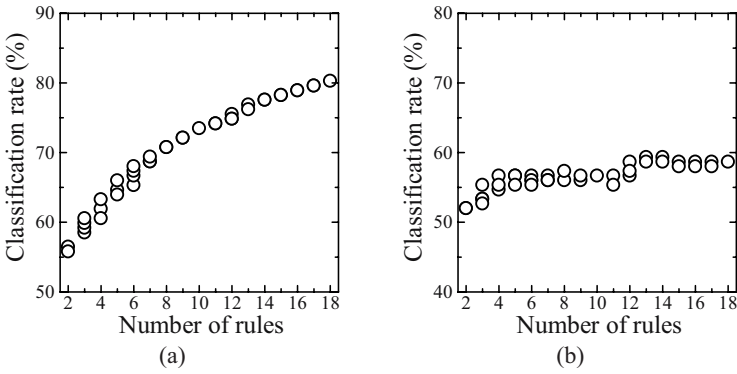


Fig. 22. Obtained non-dominated rule sets for the Cleveland heart disease data (Heart C): (a) training data accuracy; (b) test data accuracy

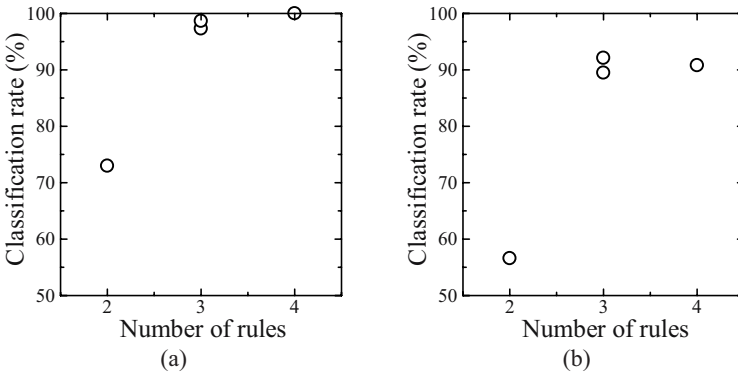


Fig. 23. Obtained non-dominated rule sets for the iris data (Iris): (a) training data accuracy; (b) test data accuracy

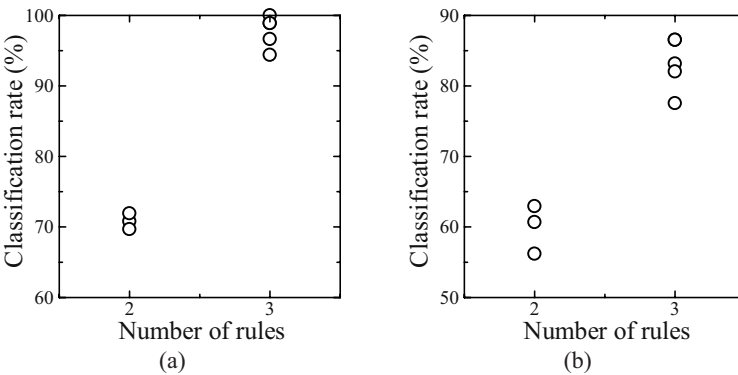


Fig. 24. Obtained non-dominated rule sets for the wine data (Wine): (a) training data accuracy; (b) test data accuracy

for test patterns of each data set in the righthand plots of Figs. 20–24. No overfitting to training patterns is observed in some data sets (for instance, the wine data in Fig. 24), while too many fuzzy rules seem to decrease the classification rates on test patterns in other data sets (for example, the Wisconsin breast cancer data in Fig. 20 and the glass data in Fig. 21). Evolutionary multiobjective fuzzy rule selection can be used to find fuzzy rule-based classifiers with high generalization ability by examining various rule sets obtained during its single run.

Evolutionary multiobjective fuzzy rule selection can also be used to visually show relations to human users, as in Figs. 20–24. Such a visualized tradeoff relation helps human users to choose a single fuzzy rule-based classifier based on their preference with respect to interpretability and accuracy. All human users do not necessarily choose the best fuzzy rule-based classifier with respect to the generalization ability; some human users may prefer simple fuzzy rule-based classifiers with high interpretability (such as Fig. 25 for the Wine data) to complicated ones with high accuracy (for instance, Fig. 26).

Since we use the total rule length as well as the number of fuzzy rules as complexity measures in evolutionary multiobjective fuzzy rule selection, it is also possible to examine the dependency of the accuracy of rule sets on the rule length of each fuzzy rule. In Fig. 27, we show the relation between classification

	x_1	x_{11}	x_{13}	Consequent
R_1				Class 1 (0.82)
R_2				Class 2 (0.77)
R_3				Class 3 (0.43)

Fig. 25. A simple rule set obtained for the Wine data with a 94.4% classification rate on training patterns and an 83.1% classification rate on test patterns

	x_1	x_7	x_{10}	x_{11}	Consequent
R_1					Class 1 (0.44)
R_2					Class 2 (0.73)
R_3					Class 3 (0.76)

Fig. 26. A complicated rule set obtained for the Wine data with a 100% classification rate on training patterns and an 86.5% classification rate on test patterns

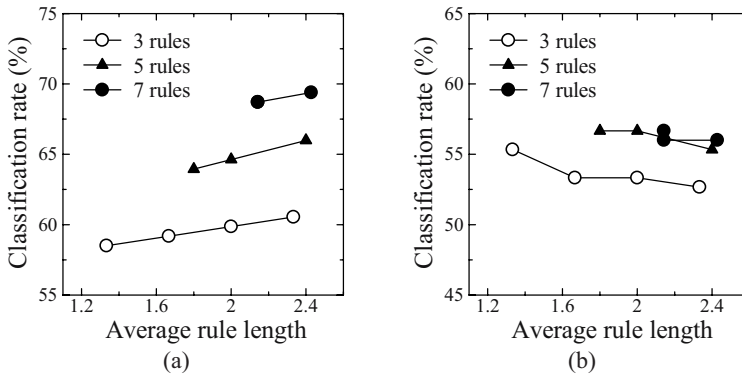


Fig. 27. Relation between classification rates and average rule length in the obtained non-dominated rule sets for the Cleveland heart disease data (Heart C): (a) training data accuracy; (b) test data accuracy

rates and average rule length for some rule sets obtained in Fig. 22 for the Cleveland heart disease data. Among the obtained non-dominated rule sets in Fig. 22, those with three, five and seven fuzzy rules are depicted in Fig. 27 using different horizontal axes (namely, average rule length). We can observe a clear accuracy-complexity tradeoff relation in Fig. 27(a) for training patterns. More specifically, classification rates on training patterns were improved by increasing the average rule length. On the other hand, classification rates on test patterns were not always improved by increasing the average rule length in Fig. 27(b).

5 Future Research Directions

A number of research issues are left for future studies in order to fully utilize the potential advantages of evolutionary multiobjective approaches to the design of fuzzy rule-based systems over single-objective ones. One issue is the mathematical formulation of the interpretability of fuzzy rule-based systems. In this Chapter, we use the number of fuzzy rules and the total rule length as complexity measures. Those complexity measures are minimized in order to maximize the interpretability of fuzzy rule-based systems. There are, however, many other aspects related to the interpretability of fuzzy rule-based systems. Among them are the number of input variables, the number of antecedent fuzzy sets for each input variable, the shape of each antecedent fuzzy set, and the overlapping grade of adjacent antecedent fuzzy sets – see [7, 8, 28, 75] for further discussions on the interpretability of fuzzy rule-based systems. An important research issue is to formulate a tractable multiobjective optimization problem using those aspects related to the interpretability of fuzzy rule-based systems. We cannot simply include all the related aspects

as separate objectives because the increase in the number of objectives makes it very difficult for EMO algorithms to efficiently search for Pareto-optimal solutions of multiobjective optimization problems [38, 39, 62].

Another issue is a theoretical discussion of the accuracy-complexity (or interpretability-accuracy) tradeoff relation of fuzzy rule-based systems. Many studies on multiobjective design of fuzzy rule-based systems do not discuss such theoretical aspects. Theoretical studies will provide us with much clearer insights about the advantages of multiobjective design over single-objective one. They may also give us a sound guideline to the design of an accurate and interpretable fuzzy rule-based system.

Handling of large data sets is also an important research issue. We believe that evolutionary multiobjective design of fuzzy rule-based systems can significantly contribute to the field of data mining (DM). A number of fuzzy data mining approaches have already been proposed [32, 36, 37]. Evolutionary multiobjective fuzzy data mining has also been proposed in some studies (for example, [68]). Whereas we discussed multiobjective optimization of rule sets in this Chapter, [68] discussed multiobjective optimization of fuzzy rules. In the field of data mining, EMO algorithms have been mainly used to search for Pareto-optimal rules [20–22, 25, 68, 82], rather than Pareto-optimal rule sets as in this Chapter. It may be an interesting topic to examine the relation between Pareto-optimal rules and Pareto-optimal rule sets [40].

Whereas a large number of evolutionary approaches to data mining have already been proposed [24], a large computational cost for iterative generation update with many individuals seems to be the main difficulty in applying evolutionary approaches to large data sets. Several tricks have already been proposed to increase the applicability of evolutionary approaches to large data sets [5, 6, 17]. Efficient tricks for decreasing the computational cost of evolutionary multiobjective approaches to the design of fuzzy rule-based systems may be needed in its application to large data sets.

6 Concluding Remarks

As we have already demonstrated in this Chapter, applying evolutionary multiobjective approaches to the design of fuzzy rule-based classifiers has several potential advantages over single-objective ones. The main advantage is that a number of non-dominated classifiers are obtained by a *single* run of a multiobjective approach. The obtained classifiers can be used for accuracy-complexity tradeoff analysis to find the best classifier with respect to generalization ability. They are also used for visualization of the interpretability-accuracy tradeoff relation, which may help human users to choose a fuzzy rule-based classifier according to their preference with respect to interpretability and accuracy. In this manner, evolutionary multiobjective approaches make it

possible to design an accurate and interpretable fuzzy rule-based classifier according to the human users' preference. It is usually much easier for human users to choose a fuzzy rule-based classifier from several alternatives along the interpretability-accuracy tradeoff surface than to represent their preference with respect to interpretability and accuracy as a scalar fitness function for single-objective approaches.

Whereas we concentrate on the design of fuzzy rule-based classifiers in this Chapter, the basic ideas of evolutionary multiobjective design can be applied to the design of fuzzy rule-based systems for other application areas such as modeling, function approximation and data mining. We believe that evolutionary multiobjective approaches will become much more popular in the near future not only in the field of fuzzy rule-based systems but also other research areas such as machine learning and data mining.

References

1. Abe S (2001) *Pattern Classification: Neuro-fuzzy Methods and Their Comparison*. Springer-Verlag, Berlin.
2. Abe S, Lan M-S (1995) A method for fuzzy rules extraction directly from numerical data and its application to pattern classification. *IEEE Trans. Fuzzy Systems*, 3(1): 18–28.
3. Abe S, Thawonmas R (1997) A fuzzy classifier with ellipsoidal regions. *IEEE Trans. Fuzzy Systems*, 5(3): 358–368.
4. Booker LB, Goldberg DE, Holland JH (1989) Classifier systems and genetic algorithms. *Artificial Intelligence*, 40(1–3): 235–282.
5. Cano JR, Herrera F, Lozano M (2005) Stratification for scaling up evolutionary prototype selection. *Pattern Recognition Letters*, 26(7): 953–963.
6. Cano JR, Herrera F, Lozano M (2006) On the combination of evolutionary algorithms and stratified strategies for training set selection in data mining. *Applied Soft Computing*, 6(3): 323–332.
7. Casillas J, Cordon O, Herrera F, Magdalena L (eds.) (2003) *Interpretability Issues in Fuzzy Modeling*. Springer-Verlag, Berlin.
8. Casillas J, Cordon O, Herrera F, Magdalena L (eds.) (2003) *Accuracy Improvements in Linguistic Fuzzy Modeling*. Springer-Verlag, Berlin.
9. Castillo L, Gonzalez A, Perez R (2001) Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm. *Fuzzy Sets and Systems*, 120(2): 309–321.
10. Cherkassky V, Mulier F (1998) *Learning from Data: Concepts, Theory, and Methods*. Wiley, New York, NY.
11. Coello CAC, Lamont GB (2004) *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, Singapore.
12. Coello CAC, van Veldhuizen DA, Lamont GB (2002) *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, Boston, MA.
13. Cordon O, del Jesus MJ, Herrera F (1999) A Proposal on reasoning methods in fuzzy rule-based classification systems. *Intl. J. Approximate Reasoning*, 20(1): 21–45.

14. Cordon O, del Jesus MJ, Herrera F, Lozano M (1999) MOGUL: a methodology to obtain genetic fuzzy rule-based systems under the iterative rule learning approach. *Intl. J. Intelligent Systems*, 14(11): 1123–1153.
15. Cordon O, Gomide F, Herrera F, Hoffmann F, Magdalena L (2004) Ten years of genetic fuzzy systems: current framework and new trends. *Fuzzy Sets and Systems*, 141(1): 5–31.
16. Cordon O, Herrera F, Hoffman F, Magdalena L (2001) *Genetic Fuzzy Systems*. World Scientific, Singapore.
17. Curry R, Heywood MI (2004) Towards efficient training on large datasets for genetic programming. *Lecture Notes in Artificial Intelligence 3060: Advances in Artificial Intelligence – Canadian AI 2004*. Springer-Verlag, Berlin: 161–174.
18. Deb K (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK.
19. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation*, 6(2): 182–197.
20. de la Iglesia B, Philpott MS, Bagnall AJ, Rayward-Smith VJ (2003) Data mining rules using multi-objective evolutionary algorithms. In: Sarker R, Reynolds R, Abbass H, Tan KC, McKay R, Essam D, Gedeon T (eds.) *Proc. 2003 Congress Evolutionary Computation*. 8–12 December, Canberra, Australia. IEEE Press, Piscataway, NJ: 1552–1559.
21. de la Iglesia B, Reynolds A, Rayward-Smith VJ (2005) Developments on a multi-objective metaheuristic (MOMH) algorithm for finding interesting sets of classification rules. *Lecture Notes in Computer Science 3410: Evolutionary Multi-Criterion Optimization – EMO 2005*. Springer-Verlag, Berlin: 826–840.
22. de la Iglesia B, Richards G, Philpott MS, Rayward-Smith VJ (2006) The application and effectiveness of a multi-objective metaheuristic algorithm for partial classification. *European J. Operational Research*, 169(3): 898–917.
23. Duda O, Hart PE, Stork DG (2001) *Pattern Classification*. Wiley, New York, NY.
24. Freitas AA (2002) *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, Berlin.
25. Ghosh A, Nath BT (2004) Multi-objective rule mining using genetic algorithms. *Information Sciences*, 163(1–3): 123–133.
26. Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA.
27. Gonzalez A, Perez R (1999) SLAVE: A genetic learning system based on an iterative approach. *IEEE Trans. Fuzzy Systems*, 7(2): 176–191.
28. Guillaume S (2001) Designing fuzzy inference systems from data: an interpretability-oriented review. *IEEE Trans. Fuzzy Systems*, 9(3): 426–443.
29. Herrera F (2005) Genetic fuzzy systems: status, critical considerations and future directions. *Intl. J. Computational Intelligence Research*, 1(1): 59–67.
30. Herrera F, Lozano M, Verdegay JL (1998) A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems*, 100(1–3): 143–158.
31. Holland JH (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
32. Hong T-P, Kuo C-S, Chi S-C (2001) Trade-off between computation time and number of rules for fuzzy mining from quantitative data. *Intl. J. Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(5): 587–604.

33. Horikawa S, Furuhashi T, Uchikawa Y (1992) On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Trans. Neural Networks*, 3(5): 801–806.
34. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2): 251–257.
35. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5): 359–366.
36. Hu YC, Chen RS, Tzeng GH (2002) Mining fuzzy association rules for classification problems. *Computers & Industrial Engineering* 43(4): 735–750.
37. Hu YC, Chen RS, Tzeng GH (2003) Finding fuzzy classification rules using data mining techniques. *Pattern Recognition Letters*, 24(1–3): 509–519.
38. Hughes EJ, (2005) Evolutionary many-objective optimization: many once or one many? In: Corne D (ed.) *Proc. 2005 Congress Evolutionary Computation 2–5 September*, Edinburgh, UK. IEEE Press, Piscataway, NJ: 222–227.
39. Ishibuchi H, Doi T, Nojima Y (2006) Incorporation of scalarizing fitness functions into evolutionary multiobjective optimization algorithms. *Lecture Notes in Computer Science 4193: PPSN IX*. Springer-Verlag, Berlin: 493–502.
40. Ishibuchi H, Kuwajima I, Nojima Y (2007) Relation between Pareto-optimal fuzzy rules and Pareto-optimal fuzzy rule sets. In: Bonissone P, Coello CAC, Jin Y (eds.) *Proc. 1st IEEE Symp. Computational Intelligence in Multicriteria Decision Making 1–5 April, 2007*, Honolulu, HI. IEEE Press, Piscataway, NJ: 42–49.
41. Ishibuchi H, Murata T (1998) A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Trans. Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 28(3): 392–403.
42. Ishibuchi H, Murata T, Turksen IB (1997) Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems*, 89(2): 135–150.
43. Ishibuchi H, Nakashima T (1999) Improving the performance of fuzzy classifier systems for pattern classification problems with continuous attributes. *IEEE Trans. Industrial Electronics* 46(6): 157–168.
44. Ishibuchi H, Nakashima T (2001) Effect of rule weights in fuzzy rule-based classification systems. *IEEE Trans. Fuzzy Systems*, 9(4): 506–515.
45. Ishibuchi H, Nakashima T, Morisawa T (1999) Voting in fuzzy rule-based systems for pattern classification problems. *Fuzzy Sets and Systems*, 103(2): 223–238.
46. Ishibuchi H, Nakashima T, Murata T (1999) Performance evaluation of fuzzy classifier systems for multi-dimensional pattern classification problems. *IEEE Trans. Systems, Man, and Cybernetics – Part B: Cybernetics*, 29(5): 601–618.
47. Ishibuchi H, Nakashima T, Murata T (2001) Three-objective genetics-based machine learning for linguistic rule extraction. *Information Sciences*, 136(1–4): 109–133.
48. Ishibuchi H, Nakashima T, Nii M (2004) *Classification and Modeling with Linguistic Information Granules: Advanced Approaches to Linguistic Data Mining*. Springer-Verlag, Berlin.
49. Ishibuchi H, Namba S (2004) Evolutionary Multiobjective Knowledge Extraction for High-dimensional Pattern Classification Problems. *Lecture Notes in Computer Science 3242: Parallel Problem Solving from Nature – PPSN VIII* Springer, Berlin: 1123–1132.

50. Ishibuchi H, Nojima Y (2007) Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning. *Intl. J. Approximate Reasoning*, 44(1): 4–31.
51. Ishibuchi H, Nozaki K, Tanaka H (1992) Distributed representation of fuzzy rules and its application to pattern classification. *Fuzzy Sets and Systems*, 52(1): 21–32.
52. Ishibuchi H, Nozaki K, Yamamoto N, Tanaka H (1995) Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Trans. Fuzzy Systems*, 3(3): 260–270.
53. Ishibuchi H, Shibata Y (2004) Mating scheme for controlling the diversity-convergence balance for multiobjective optimization. *Lecture Notes in Computer Science 3102: Genetic and Evolutionary Computation – GECCO 2004*. Springer-Verlag, Berlin: 1259–1271.
54. Ishibuchi H, Yamamoto T (2002) Effect of fuzzy discretization in fuzzy rule-based systems for classification problems with continuous attributes. *Archives of Control Sciences*, 12(4): 351–378.
55. Ishibuchi H, Yamamoto T (2004) Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining. *Fuzzy Sets and Systems*, 141(1): 59–88.
56. Ishibuchi H, Yamamoto T (2005) Rule weight specification in fuzzy rule-based classification systems. *IEEE Trans. Fuzzy Systems*, 13(4): 428–435.
57. Ishibuchi H, Yamamoto T, Nakashima T (2001) Fuzzy data mining: effect of fuzzy discretization. In: Cercone N, Lin TY, Wu X (eds.) *Proc. 1st IEEE Intl. Conf. Data Mining 29 November - 2 December, San Jose, CA*. IEEE Computer Society, Los Alamitos, CA: 241–248.
58. Ishibuchi H, Yamamoto T, Nakashima T (2005) Hybridization of fuzzy GBML approaches for pattern classification problems. *IEEE Trans. Systems, Man, and Cybernetics – Part B: Cybernetics*, 35(2): 359–365.
59. Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Trans. Evolutionary Computation*, 7(2): 204–223.
60. Jang J-S R (1993) ANFIS: adaptive-network-based fuzzy inference system. *IEEE Trans. Systems, Man, and Cybernetics*, 23(3): 665–685.
61. Jaskiewicz A (2002) Genetic local search for multi-objective combinatorial optimization. *European J. Operational Research*, 137(1): 50–71.
62. Jaskiewicz A (2004) On the computational efficiency of multiple objective metaheuristics: the knapsack problem case study. *European J. Operational Research*, 158(2): 418–433.
63. Jimenez F, Gomez-Skarmeta AF, Sanchez G, Roubos H, Babuska R (2001) Accurate, transparent and compact fuzzy models for function approximation and dynamic modeling through multi-objective evolutionary optimization. *Lecture Notes in Computer Science 1993: Evolutionary Multi-Criterion Optimization – EMO 2001*. Springer-Verlag, Berlin: 653–667.
64. Jin Y (2000) Fuzzy modeling of high-dimensional systems: complexity reduction and interpretability improvement. *IEEE Trans. Fuzzy Systems*, 8(2): 212–221.
65. Jin Y (ed.) (2006) *Multi-Objective Machine Learning*. Springer-Verlag, Berlin.
66. Jin Y, von Seelen W, Sendhoff B (1999) On generating FC³ fuzzy rule systems from data using evolution strategies. *IEEE Trans. Systems, Man, and Cybernetics – Part B: Cybernetics*, 29(6): 829–845.

67. Karr CL, Gentry EJ (1993) Fuzzy control of pH using genetic algorithms. *IEEE Trans. Fuzzy Systems*, 1(1): 46–53.
68. Kaya M (2006) Multi-objective genetic algorithm based approaches for mining optimized fuzzy association rules. *Soft Computing*, 10(7): 578–586.
69. Knowles JD, Corne DW (2000) Approximating the nondominated front using Pareto archived evolution strategy *Evolutionary Computation*, 8(2): 149–172.
70. Kosko B (1992) Fuzzy systems as universal approximators. In: Bezdek J (ed.) *Proc. IEEE Intl. Conf. Fuzzy Systems*. 8–12 March, San Diego, CA. IEEE Press, Piscataway, NJ: 1153–1162.
71. Kuncheva LI (2000) *Fuzzy Classifier Design*. Physica-Verlag, Heidelberg.
72. Kuncheva LI (2000) How good are fuzzy if-then classifiers? *IEEE Trans. Systems, Man, and Cybernetics – Part B: Cybernetics*, 30(4): 501–509.
73. Mendel JM (1995) Fuzzy-logic systems for engineering – a tutorial. *Proc. IEEE*, 83(3): 345–377.
74. Miettinen K (1998) *Nonlinear Multiobjective Optimization*. Kluwer, Boston, MA.
75. Mikut R, Jakel J, Groll L (2005) Interpretability issues in data-based learning of fuzzy systems. *Fuzzy Sets and Systems*, 150(2): 179–197.
76. Nauck D, Klawonn F, Kruse R (1997) *Foundations of Neuro-Fuzzy Systems*. Wiley, Chichester, UK.
77. Nauck D, Kruse R (1997) A neuro-fuzzy method to learn fuzzy classification rules from data. *Fuzzy Sets and Systems*, 89(3): 277–288.
78. Nauck D, Kruse R (1998) How the learning of rule weights affects the interpretability of fuzzy systems. In: Keller J (ed.) *Proc. 7th IEEE Intl. Conf. Fuzzy Systems* 4–9 May, Anchorage, AK. IEEE Press, Piscataway, NJ: 1235–1240.
79. Nauck D, Kruse R (1999) Obtaining interpretable fuzzy classification rules from medical data. *Artificial Intelligence in Medicine*, 16(2): 149–169.
80. Nozaki K, Ishibuchi H, Tanaka H (1996) Adaptive fuzzy rule-based classification systems. *IEEE Trans. Fuzzy Systems*, 4(3): 238–250.
81. Quinlan JR (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.
82. Reynolds A, de la Iglesia B (2006) Rule induction using multi-objective meta-heuristics: encouraging rule diversity. In: Wang L (ed.) *Proc. 2006 Intl. Joint Conf. Neural Networks* 16–21 July, British Columbia, Canada. IEEE Press, Piscataway, NJ: 6375–6382.
83. Roubos H, Setnes M (2001) Compact and transparent fuzzy models and classifiers through iterative complexity reduction. *IEEE Trans. Fuzzy Systems*, 9(4): 516–524.
84. Rumelhart DE, McClelland JL, the PDP Research Group (1986) *Parallel Distributed Processing*. MIT Press, Cambridge, MA.
85. Schaffer JD (1985) Multiple objective optimization with vector evaluated genetic algorithms. In: Grefenstette J (ed.) *Proc. 1st Intl. Conf. Genetic Algorithms and Their Applications*. 24–26 July, Pittsburgh, PA. Lawrence Erlbaum Associates, Hillsdale, NJ: 93–100.
86. Setnes M, Babuska K, Verbruggen B (1998) Rule-based modeling: precision and transparency. *IEEE Trans. Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 28(1): 165–169.
87. Setnes M, Roubos H (2000) GA-based modeling and classification: complexity and performance. *IEEE Trans. Fuzzy Systems*, 8(5): 509–522.

88. Smith SF (1980) A Learning System based on Genetic Algorithms. *PhD Dissertation* Department of Computer Science, University of Pittsburgh, PA.
89. van den Berg J, Kaymak U, van den Bergh WM (2002) Fuzzy classification using probability based rule weighting. In: Fogel DB (ed.) *Proc. 11th IEEE Intl. Conf. Fuzzy Systems* 12–17 May, Honolulu, HI. IEEE Press, Piscataway, NJ: 991–996.
90. Wang H, Kwong S, Jin Y, Wei W, Man KF (2005) Agent-based evolutionary approach for interpretable rule-based knowledge extraction. *IEEE Trans. Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 35(2): 143–155.
91. Wang H, Kwong S, Jin Y, Wei W, Man KF (2005) Multi-objective hierarchical genetic algorithm for interpretable fuzzy rule-based knowledge extraction. *Fuzzy Sets and Systems*, 149(1): 149–186.
92. Wang LX, Mendel JM (1992) Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. *IEEE Trans. Neural Networks*, 3(5): 807–814.
93. Wang LX, Mendel JM (1992) Generating fuzzy rules by learning from examples. *IEEE Trans. Systems, Man, and Cybernetics* 22(6): 1414–1427.
94. Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Trans. Evolutionary Computation*, 3(4): 257–271.

Resources

1 Key Books

1.1 Fuzzy Rule-Based Classification Systems

Abe S (2001) *Pattern Classification: Neuro-Fuzzy Methods and Their Comparison*. Springer-Verlag, Berlin.

Bezdek JC, Pal SK (eds.) (1992) *Fuzzy Models for Pattern Recognition: Methods That Search for Structures in Data*. IEEE Press, Piscataway, NJ.

Ishibuchi H, Nakashima T, Nii M (2004) *Classification and Modeling with Linguistic Information Granules: Advanced Approaches to Linguistic Data Mining*. Springer-Verlag, Berlin.

Kuncheva LI (2000) *Fuzzy Classifier Design*. Physica-Verlag, Heidelberg.

1.2 Genetic Algorithms

Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA.

Goldberg DE (2002) *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Springer-Verlag, Berlin.

Holland JH (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

1.3 Genetic Fuzzy Systems

Casillas J, Cordon O, Herrera F, Magdalena L (eds.) (2003) *Accuracy Improvements in Linguistic Fuzzy Modeling*. Springer-Verlag, Berlin.

Cordon O, Herrera F, Hoffman F, Magdalena L (2001) *Genetic Fuzzy Systems*. World Scientific, Singapore.

1.4 Evolutionary Multiobjective Optimization

Abraham A, Jain LC, Goldberg R (eds.) (2005) *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer-Verlag, Berlin.

Coello CAC, Lamont GB (2004) *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, Singapore.

Coello CAC, van Veldhuizen DA, Lamont GB (2002) *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, Boston, MA.

Deb K (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK.

1.5 Evolutionary Multiobjective Machine Learning and Knowledge Extraction

Ghosh A, Dehuri S, Ghosh S (eds.) *Multi-Objective Evolutionary Algorithms for Knowledge Discovery from Data Bases*. Springer-Verlag, Berlin (in press).

Jin Y (ed.) (2001) *Multi-Objective Machine Learning*. Springer-Verlag, Berlin.

2 Conferences

2.1 Fuzzy Systems

Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT)

IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)

International Fuzzy Systems Association World Congress (IFSA World Congress)

Intl. Conf. North American Fuzzy Information Processing Society (NAFIPS)

2.2 Genetic Algorithms

Foundations of Genetic Algorithms (FOGA)

Genetic and Evolutionary Computation Conference (GECCO)

IEEE Congress on Evolutionary Computation (CEC)

International Conference on Parallel Problem Solving from Nature (PPSN)

2.3 Genetic Fuzzy Systems

International Workshop on Genetic and Evolving Fuzzy Systems (GEFS)

2.4 Evolutionary Multiobjective Optimization

IEEE Symposium on Computational Intelligence in Multicriteria Decision Making (MCDM)

International Conference on Evolutionary Multi-Criterion Optimization (EMO)

2.5 Hybrid Systems

International Conference on Hybrid Intelligent Systems (HIS)

2.6 Broader Areas, Including Fuzzy Systems and Genetic Algorithms

IEEE International Conference on Systems, Man, and Cybernetics (SMC)

3 Journals

3.1 Fuzzy Systems

Fuzzy Sets and Systems (Elsevier)

IEEE Transactions on Fuzzy Systems

International Journal of Approximate Reasoning (Elsevier)

Intl. J. Uncertainty Fuzziness and Knowledge-Based Systems (World Scientific)

3.2 Genetic Algorithms

Evolutionary Computation (MIT Press)

IEEE Transactions on Evolutionary Computation

3.3 Broader Areas, Including Fuzzy Systems and Genetic Algorithms

IEEE Computational Intelligence Magazine

IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics

Intl. J. Computational Intelligence Research (Research India Publications)

Soft Computing (Springer)

4 Websites

ACM Special Interest Group on Genetic and Evolutionary Computation (SIGEVO) <http://www.sigevo.org/>

BICS: The Berkeley Initiative in Soft Computing
<http://www.cs.berkeley.edu/~zadeh/>

European Society for Fuzzy Logic and Technology (EUSFLAT)
<http://www.eusflat.org/>

Evolutionary Multiobjective Optimization Web Page
<http://www.lania.mx/~ccoello/EMOO/>

IEEE Computational Intelligence Society
<http://www.ieee-cis.org/>

IEEE Systems, Man, and Cybernetics Society
<http://www.ieee-smc.org/>

International Fuzzy Systems Association (IFSA)
<http://www.cmplx.cse.nagoya-u.ac.jp/~ifsa/>

KEEL: Knowledge Extraction based on Evolutionary Learning
<http://sci2s.ugr.es/keel/>

5 (Open Source) Software

Fuzzy Sets and Systems Software Repository

<http://www.fuzzysoftware.org/>

PISA: A Platform and Programming Language Independent Interface for Search Algorithms

<http://www.tik.ee.ethz.ch/pisa/>

6 Data Bases

UCI Machine Learning Repository

<http://www.ics.uci.edu/~mlern/MLRepository.html>

Artificial Neural Networks

Data Mining in QoS-Aware Media Grids

Xiuju Fu¹, Xiaorong Li¹, Lipo Wang², David Ong³,
and Stephen John Turner³

¹ Institute of High Performance Computing, Singapore 117528,
fuxj@ihpc.a-star.edu.sg, lixr@ihpc.a-star.edu.sg

² School of Electrical and Electronic Engineering, Nanyang Technological
University, Singapore 639798, elpwang@ntu.edu.sg

³ School of Computing, Nanyang Technological University, Singapore 639798,
david@ntu.edu.sg, ASSJTurner@ntu.edu.sg

1 Introduction

With the advent of high-speed networking technology and multimedia compression, various network-based multimedia services have become available [29]. Clients can download music, send/receive multimedia emails, browse multimedia material in eLibraries and enjoy high quality movies on-line. Media streaming [24] is one of the most popular real-time multimedia services. It enables clients to view multimedia content online without completely downloading it. In addition, it can support complete flexibility in presentation by controlling playback – in other words, clients can alter the presentation by using Video Cassette Recorder (VCR)-like control facilities, such as ‘rewind’, ‘fast-forward’, ‘pause’, and the like. However, rendering high quality media streaming via networks is still a challenging task, not only due to the large size of video files, but also because of the critical requirements to guarantee Quality-of-Service (QoS) – these being delay, loss rate, jitter, and so forth – for distributing real-time media streams over networks.

Grid computing [8] is a recently developed technology which aggregates heterogeneous network resources (CPU, storage, bandwidth, software, information, and similar) through the networks. It has the features of loosely coupled distributed architecture, service-oriented structure, as well as self-organization and decentralized methods of resource management. Grid middleware provides a firm foundation to support large-scale streaming services. Recent developments in media grids involve using industry standard streaming protocols (RTP, RTSP, and RTCP) and integrating various grid technologies (for instance, information service, data management service and resource management service) into media grid to support a large population of Internet streaming users.

A QoS-aware media grid [35] aims at presenting scalable, robust and secure media access over grid environments [13]. An ever-increasing demand in computational resources has prompted the growth of grid techniques, and has increased the heterogeneity of network services and connections. QoS-aware media streaming [24] is considered a critical part of a media grid, and involves the access of text, graphics, audio and video content. As the main components of multimedia services, audio and video streaming require qualified network resources, such as high bandwidth and low latency. Since media streaming servers operate in a shared network environment, an efficient traffic prediction mechanism on available bandwidth is important for choosing appropriate servers and resolutions of media content to provide Quality-of-Service (QoS) of media streaming in distributed heterogeneous grid environments. To support QoS-aware streaming services, multiple versions of media content with different resolutions are provided for adapting to variations in network conditions. If prediction of available bandwidth can be made *beforehand*, a suitable version of media content can be chosen based on this prediction, which could help decrease degradation, such as lost packets and jitter caused by insufficient in bandwidth.

Typical data mining (DM) tasks include prediction, classification, clustering, and discovery of association rules. With the objective of discovering unknown patterns from data, DM methodologies have been derived from the fields of machine learning (ML), artificial intelligence (AI), and statistics. Data mining techniques have begun to serve fields outside of computer science, scientific research and artificial intelligence, such as the financial area and factory assembly lines. DM has been shown to lead to improved efficiency in manufacturing, prompting marketing campaigns, detecting fraud, and predicting diseases based on medical records. With ever increasing demands on computing resources, DM applications have become desirable in grid computing environments.

Being an integrated environment, Grids are autonomous, share resources, heterogenous and distributed in nature. A Media Grid shares these characteristics, but as well emphasizes media streaming functions. Many techniques have been applied to media streaming for improving QoS through predicting the following:

1. *bandwidth* – in order to adapt to dynamic media content streaming;
2. *media streaming service request patterns* – in order to automatically duplicate media content;
3. *workload and resource availability of media servers* – to determine intelligent scheduling strategies;
4. *usage patterns of grid computing environments*;
5. *user request patterns in the media grid*.

Data are accumulated from heterogeneous resources in media grids, but there is currently a lack of an efficient data mining framework and appropriate

techniques for analyzing data generated in such grids. The quality of media streaming is mainly affected by network capacity, bandwidth, and throughput. Capacity is the maximum possible bandwidth over a network path. There are two types of bandwidth: (i) *available bandwidth* is the maximum allowable bandwidth, and (ii) *consumption bandwidth* is the amount of bandwidth consumed. With bandwidth prediction, we could avoid congestion caused by heavy network loads and reduce overestimation (underestimation) of the bandwidth requirements of clients. In this Chapter, we focus on applying neural networks for predicting bandwidth, which facilitates QoS in media streaming.

The Chapter is organized as follows. Section 2 describes related work. Section 3 presents a media grid framework. Section 4 describes the data mining strategy for bandwidth prediction in media grids. Experiments of bandwidth prediction are presented in Sect. 5. Sect. 6 concludes the Chapter.

2 Related Work

Grid computing is a relatively recent technology that aggregates large amounts of geographically distributed resources – such as PC clusters, supercomputers, network storages/caches – to make applications which are impossible for a local machine/cluster to be possible over networks. During the last decade, grid technologies have been applied to computationally intensive applications in the field of high energy physics, and projects like **GrIPhyN** [17], **PPDG** [27], **BIRN** [5], **EuroGrid** [12] have achieved much success in data processing and distribution. While most current research focuses on providing high capacity computational services on grids, there is increasing interest in exploring the deployment of grid technologies for providing multimedia services over networks. Recent developments of media grids involve using industry standard streaming protocols [3] (for instance, RTP, RTSP, and RTCP) and integrating various grid technologies (for example, information, data management and resource management services) into media grids [35] to support a large population of internet streaming users. In this Section, we review previous work on network bandwidth prediction over media grid environments, and briefly introduce neural networks.

2.1 Network Bandwidth Prediction

Prediction techniques have been widely applied in media streaming processes. One such application is to predict variable bit rate (VBR) video traffic to improve the network utilization efficiency while supporting QoS of VBR video. The prediction aims at forecasting real-time VBR video traffic for dynamic bandwidth allocation. A number of algorithms both in the time domain and wavelet domain for video traffic prediction have been proposed in the literature [25]. Another application of prediction models is to predict the network

traffic *directly*. In this Chapter, we focus on predicting network traffic in the time domain instead of looking into VBR video traffic.

In [33] and [34], the bandwidth of outgoing network node links is modelled according to wavelet transformation coefficients. Neural networks are applied as well to predict these coefficients. However, it is difficult to determine the coefficients in the wavelet modeling method. The network traffic is dynamic and might vary dramatically under different conditions or during different time periods. It is therefore inappropriate to fix the coefficients for predicting the dynamic bandwidth in wavelet modeling.

The Network Weather Service (NWS) [41] is a well-known network performance measurement and has been used to predict network traffic in grid computing. Besides probes which are used to record network performance, NWS uses prediction methods such as mean-based, median-based and autoregression to forecast the network traffic. Despite its strengths, this approach does have disadvantages similar to the wavelet modeling method – in other words, it is difficult to adapt to the dynamic network conditions of grid environments.

As a popular data mining tool, neural networks have been employed to predict network bandwidth. For example, [11] used neural networks to predict available network bandwidth. In their work, the recorded network traffic is divided into non-overlapped and continuous bins. The time stamp, minimum packet rate, maximum packet rate, average packet rate, minimum bit rate, maximum bit rate, and average bit rate are derived from each bin data and used as the inputs to neural networks. The outputs of the neural network predictor are the bandwidth of later K -step bins, with K being defined by the user. This method holds promise for predicting network bandwidth. Our network bandwidth prediction method is similar to this method. However, we propose a new performance metric to better evaluate the performance of neural network predictors.

2.2 Brief Overviews on Neural Networks

There are many different types of artificial neural networks (ANNs) in use today. ANNs can be categorized according to different aspects, such as learning algorithm, the number of network layers, the direction of signal flow, the activation function, and so on.

Based on the learning algorithm, neural networks can be classified into three major categories:

- In *supervised learning*, pairs of input and target vectors are required to train networks, so that appropriate outputs corresponding to input signals are generated accordingly. With an input vector applied, the error between the output of the neural network and its target output is calculated, which

is then used to tune weights in order to minimize this error. The delta or least mean square (LMS) learning rule is a well-known method for minimizing errors. Supervised learning includes error-correction learning, reinforcement learning and stochastic learning.

- *Unsupervised learning* does not require target vectors for the outputs. Without input-output training pairs as external teachers, unsupervised learning is self-organized to produce consistent output vectors by modifying weights. Paradigms of unsupervised learning include Hebbian learning and competitive learning. Kohonen's self-organizing map (SOM) is a typical neural network whose learning is unsupervised.
- Some neural networks employ *hybrid learning*. For example, counterpropagation networks and radial basis function (RBF) networks use both supervised (at the output layer) and unsupervised (at the hidden layer) learning. Counterpropagation neural networks combine network paradigms of SOM and Grossberg's outstar [40]. The counterpropagation neural network can be used to produce corresponding outputs when an input vector is incomplete, noisy or partially in error.

According to the direction of signal flow, neural networks can be categorized as *feedforward* – in which weight connections are fed forward from inputs through hidden neurons to output neurons – and *recurrent* – in which feedback connections are present. Compared with feedforward neural networks, recurrent neural networks can be unstable and dynamic. Hopfield neural networks [40] are well-known recurrent neural networks. Recurrent neural networks have been studied as examples of chaotic systems [37,38]. The bidirectional associative memory (BAM) network is another type of neural network which employs feedback. The BAM is heteroassociative. Both BAM and Hopfield neural networks are able to produce correct outputs when inputs are partially missing or incorrect.

Adaptive resonance theory (ART) networks [40] deal well with the stability-plasticity dilemma, namely “How can a learning system be designed to remain plastic, or adaptive, in response to significant events and yet remain stable in response to irrelevant events”? [40] ART classifies an input vector into its class according to the stored pattern with which it is most similar. The stored pattern is tuned to make it closer to (align with) the input vector. Without finding its matching pattern – it is within a predefined tolerance for matching purposes – a new category is generated by storing a pattern which is similar to the input vector.

Although there are neural networks *without* hidden layers [26], their applications are few due to their limited ability to approximate data in a global view. Focusing on neural networks with hidden layers, we can categorize them based on how the activation of a hidden unit is determined. In multi-layer perceptron (MLP) neural networks, the activation of a hidden unit is determined by the scalar product of the input vector and its corresponding weight vector.

In radial basis function (RBF) neural networks the activation is determined by the distance between the input vector and a prototype vector.

In this Chapter, we employ MLPs exclusively.

3 System Model of Data Analysis over Media Grid

In this Section, we describe our data analysis system model in which data mining techniques are employed.

3.1 Architecture

Our media grid multi-agent data analysis system adopts distributed analysis agents to provide on-line monitoring services for heterogeneous Internet users. Figure 1 shows the layout of the agent-based data analysis system where media servers, clients, and analysis agents are geographically distributed over the network. Each analysis agent can work independently or cooperatively to monitor and predict the resource usage of media servers over a certain area so as to improve the global resource utilization. Distributed analysis agents reduce the processing burden of streaming servers and are sufficiently flexible

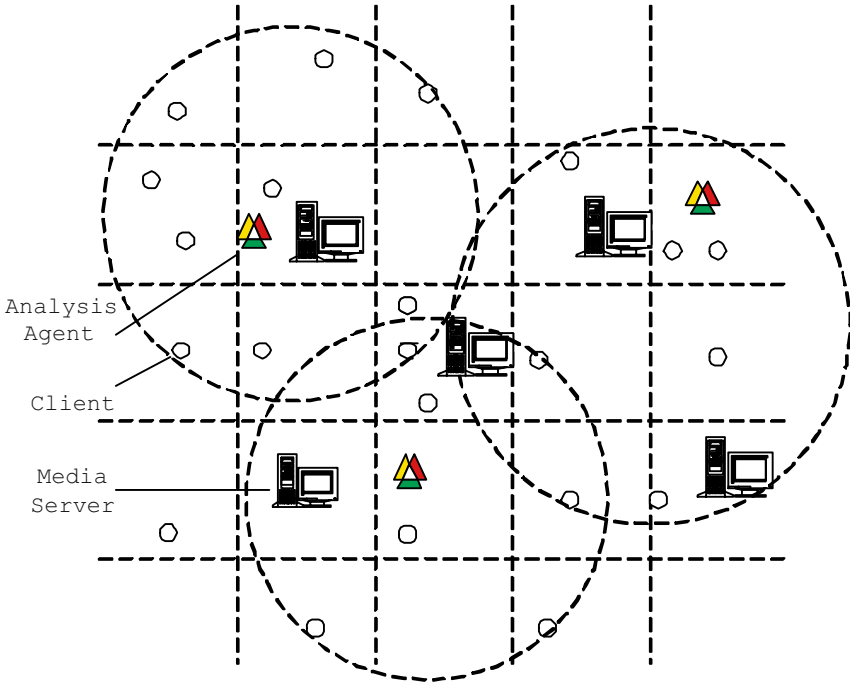


Fig. 1. Topology layout of a multi-agent based data analysis system

to deal with various streaming applications. Such a system is highly scalable and can ameliorate the effects of failure and overloading in a centralized monitoring system.

A Media Grid is a distributed infrastructure which brings together various distributed resources into a virtual environment and provides customized streaming services in a cost-effective way. It accumulates various types of abundant resources on the Internet (such as network storage, Personal Computers, or PC clusters) to improve the system computation and storage capacity. Figure 2 shows the hierarchical layered architecture of a media grid. The lowest layer is the resource layer which manipulates distributed physical resources – media servers, storage, camera, and the like – to support higher level grid services. Above the resource layer is the grid middleware layer which includes the grid component services – for example, information service [21], data management service [16], and so on. The grid middleware provides a firm foundation to support large-scale streaming services. It inherits such features of grids generally as the integration of loosely-coupled network resources, a service-oriented structure, self-organization and decentralized methods of resource management. Media grid components are built above the grid component layer to provide multimedia-related services for Internet users. The media grid portal serves as an abstraction layer for the

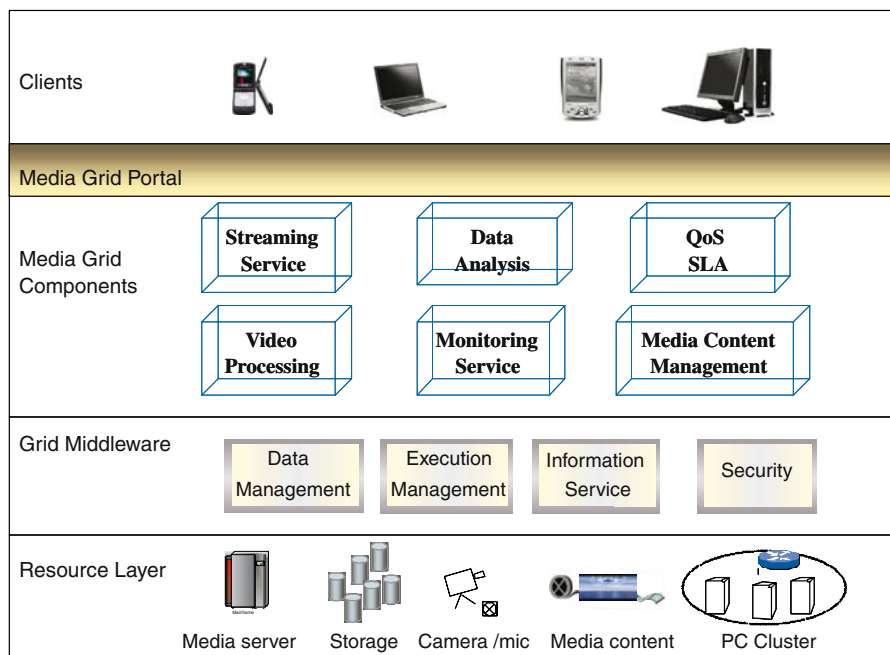


Fig. 2. Layered architecture of media grid

various media grid components. Through this portal, clients can subscribe to access media content with their QoS requirements such as playback time, bit rate, resolution, and so forth.

3.2 System Components

A media grid multi-agent data analysis system monitors and predicts network resource usage over distributed media grid environments. It consists of four main components: an analysis agent, a web portal, data storage, and a grid information service.

- *Analysis Agent*: The analysis Agent collects QoS information from a streaming server/client or (both), analyzes it, provide feedback to the user, and stores the data into a database.
- *Web Portal*: The web portal is a web service based interface which allows users to customize the quality monitoring by specifying their quality metrics or measurement conditions.
- *Data Storage*: QoS information and results are stored in a distributed database for archiving or long-term analysis.
- *Grid Information Service* [21]: When the user submits requests to the web portal, the system needs to find a suitable analysis agent. The Grid Information Service will help to provide information on the available resources.

Figure 3 demonstrates a working scenario of the multi-agent data analysis service, including the interaction between each component. Each analysis agent will register itself to the grid information service as a computational resource over the network. The data analysis time sequence includes:

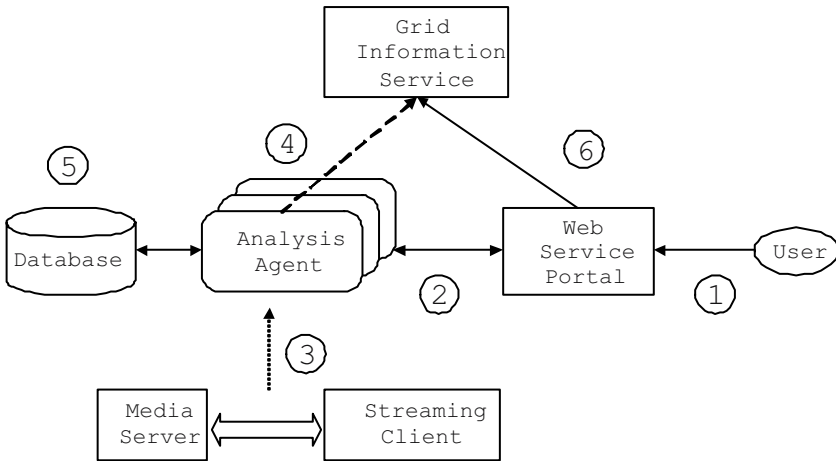


Fig. 3. Architecture and component interaction

1. clients submit requests via the web service portal;
2. data are collected from the streaming servers and are passed to the analysis agent for analysis;
3. the analysis agent performs data processing and predicts bandwidth consumption;
4. results are stored into the database for archiving or future analysis;
5. the results of the analysis or quality reports are sent back to the service providers to assist with resource management.

For a large-scale network with multiple media servers distributed over different domains, it is not practical to use only *one* centralized server to monitor and analyze the system information, due to the limited CPU and bandwidth resources. Such a system connects multiple agents and improves capacity by utilizing distributed computation power – such as CPUs and workstations – over the Internet.

4 Data Mining Strategy for Bandwidth Prediction

In this Section, we briefly introduce MLPs as neural network predictors for predicting network traffic in media grids. A data mining strategy for training these neural networks is presented later (Sect. 4.2).

4.1 Multi-Layer Perceptron Neural Network

A typical multi-layer perceptron (MLP) neural network classifier is shown in Fig. 4.

A hidden layer is required for MLPs to classify linearly inseparable data sets and make prediction. The input nodes do not carry out any processing. A hidden neuron in the hidden layer is shown in Fig. 5.

The j th output of a feedforward MLP neural network is [6]:

$$y_j = f \left(\sum_{i=1}^K W_{ij}^{(2)} * \phi_i(\mathbf{x}) + b_j^{(2)} \right) \quad (1)$$

where $W_{ij}^{(2)}$ is the weight connecting hidden neuron i with output neuron j , K is the number of hidden neurons, $b_j^{(2)}$ is the bias of output neuron j , $\phi_i(\mathbf{x})$ is the output of hidden neuron i , and \mathbf{x} is the input vector.

$$\phi_i(\mathbf{x}) = f(\mathbf{W}_i^{(1)} \cdot \mathbf{x} + b_i^{(1)}) \quad (2)$$

where $\mathbf{W}_i^{(1)}$ is the weight vector connecting the input vector with hidden neuron i , and $b_i^{(1)}$ is the bias of hidden neuron i .

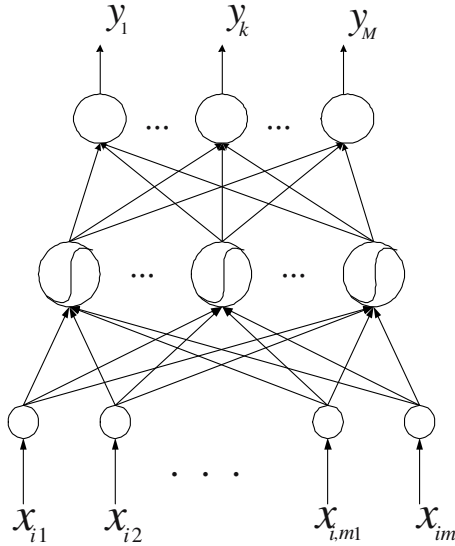


Fig. 4. A three-layer MLP neural network with one hidden layer

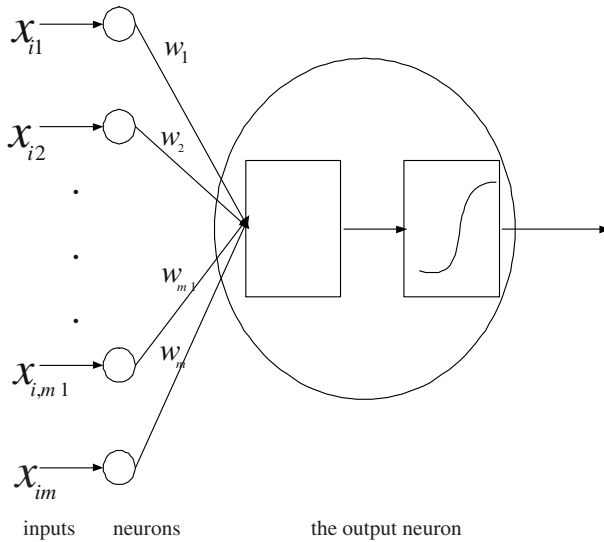


Fig. 5. An MLP hidden neuron

A common activation function f is the sigmoid (or logistic) function:

$$f(z) = \frac{1}{1 + e^{-\beta z}} \tag{3}$$

where β is the gain.

Another activation function often used in MLP neural networks is the hyperbolic tangent, which takes on values between -1 and $+1$ (instead of 0.1 , as with the sigmoid) [40]:

$$f(z) = \frac{e^{\beta z} - e^{-\beta z}}{e^{\beta z} + e^{-\beta z}} \quad (4)$$

There are many training algorithms for MLP neural networks reported in the literature, for example: gradient descent error backpropagation (BP), backpropagation with adaptive learning rate, backpropagation with momentum, Quasi-Newton backpropagation, Bayesian regularization backpropagation, conjugate gradient backpropagation, and the Levenberg-Marquardt algorithm [20].

The backpropagation technique [30] is commonly used to train MLPs. This technique could learn more than two layers of a network. The key idea of the back-propagation technique is that the error obtained from the output layer is propagated backwards to the hidden layer and is used to guide training of the weights between the hidden layer and the input layer.

[22] showed that the BP algorithm is very sensitive to initial weight selection. Prototype patterns [9] and the orthogonal least square algorithm [23] can be used to initialize the weights. The initialization of weights and biases has a great impact on both the network training (convergence) time and generalization performance. Usually, the weights and biases are initialized to small random values. If the random initial weights happen to be far from a good solution or they are near a poor local minimum, training may take a long time or become trapped there [15]. Proper weight initialization will place the weights close to a good solution, which reduces training time and increases the possibility of reaching a good solution.

[15] proposed initialization of weights using a clustering algorithm based on mean local density (MLD). This method easily leads to good performance, whereas random weight initialization leads to a wide variety of different results, many of which are poor. However, it is noted that the *best* result from random weight initialization was much better than the result obtained from the MLD initialization method.

4.2 Data Mining Strategy

Assume media content is available in multiple formats with different resolutions. Rather than streaming high resolution media content which suffers jitter or long delays due to insufficient bandwidth, smooth streaming of content using a relatively lower resolution might be preferable. Media content is available in multiple formats with different resolutions. By having knowledge of bandwidth in advance, our strategy could be to automatically switch to an appropriate resolution of media content in order to adapt to changing network

traffic conditions. Another reason for predicting network traffic is to better allocate the job load to different media servers distributed in different physical locations.

The Data mining model is as follows:

1. *Raw data collection* – Data are collected for training neural networks in order to predict future network traffic;
2. *Data cleaning and transformation* – The collected raw data are in the form of a time-series, and therefore need to be transformed into multi-dimensional format prior to inputting to the neural network. Moreover, when noise is present in the data, de-noising (filtering) is usually required;
3. *Neural networks are trained according to historical data* – The performance of neural network predictors is evaluated to meet the needs of media grid QoS;
4. The trained neural network predictors are used for *predicting network traffic, and for determining an adaptive streaming media strategy.*

Data Collection for Network Bandwidth

In order to predict network traffic, it is important to collect historical data. Future trends can then be discovered (predicted) using various techniques, based on this historical data. In general, if the network varies dramatically day-by-day, at least two weeks of data are preferred. In order to collect historical network traffic data for training and testing purposes, a C-based program – *InfoDaemon* – is used to capture the server incoming and outgoing bandwidth every 2 seconds. The captured data are then passed to the analysis agent for processing and analysis. In this Chapter, we focus on the analysis and prediction of the outgoing bandwidth consumption, since most traffic is caused by streams transmitted from media servers.

Data Preprocessing

Data preprocessing is critical due to its significant influence on the performance of data mining models. Data preprocessing usually includes noise elimination, feature selection, data partition, data transformation, data integration, and missing data processing.

The time-series data representing the network bandwidth can be written as:

$$Y = \{y(t)|t = 1, 2, \dots, N\} \quad (5)$$

where N is the number of data records. The data are transformed into the format which is used for inputs to the neural network predictors. The

one-dimensional (1D) data are transformed into multi-dimensional data, as follows:

$$Y_1 = \begin{pmatrix} y(1) & y(m+1) & y(2m+1) & \dots \\ y(2) & y(m+2) & y(2m+2) & \dots \\ \dots & \dots & \dots & \dots \\ y(m) & y(2m) & y(3m) & \dots \end{pmatrix} \tag{6}$$

where m is the size of the window which divides the 1D time series data into m -dimensional data, m being determined by the user according to media streaming strategy. If a longer period of bandwidth prediction is needed, m is set as a larger number.

Including the m bandwidth rates in one vector, we need to add additional variables which are significant for prediction, and these are transformed from the m variables. They include time stamp, minimum bandwidth rate, maximum bandwidth rate, and average bandwidth of the m bandwidth rates. Thus a new datum X with $m + 4$ dimensions is generated accordingly:

$$X = \begin{pmatrix} y(1) & y(m+1) & y(2m+1) & \dots \\ y(2) & y(m+2) & y(2m+2) & \dots \\ \dots & \dots & \dots & \dots \\ y(m) & y(2m) & y(3m) & \dots \\ t & t+1 & t+3 & \dots \\ a(1) & a(2) & a(3) & \dots \\ m_i(1) & m_i(2) & m_i(3) & \dots \\ m_a(1) & m_a(2) & m_a(3) & \dots \end{pmatrix} \tag{7}$$

where $a(i)$, $m_i(i)$ and $m_a(i)$ are the average, minimum and maximum bandwidth of the vector $Y_1(i) = \{y(i * m + 1), y(i * m + 2), \dots, y((i + 1) * m)\}$, respectively; t is the initial time stamp, which can be set as $t = 1$.

4.3 Performance Metrics

Several performance metrics for measuring the accuracy of a predictor exist in the literature. For instance, the coefficient of determination which is the mean squared error (MSE) normalized by the variance of the actual data is used as the performance metric in [19]. The disadvantage of the coefficient of determination lies in that the performance evaluation is inappropriate when the actual data only vary around the mean value. In [28], the maximum and mean error are used as the measurement of performance. In [11], the relative prediction error is used as the metric to evaluate predictor performance:

$$err = \frac{PredictedValue - ActualValue}{ActualValue} \tag{8}$$

The mean error and relative mean error suffer the same problem in which the performance values produced are affected significantly if there are isolated errors with a large magnitude in value.

In order to overcome the aforementioned disadvantages, we propose a new multi-level performance metric represented by the vector $P = \{p_1, p_2, \dots, p_l\}$, where l is determined empirically by the user, and reflects the level of performance metric needed. The relative prediction errors from Eqn. (8) are sorted in ascending order. Assume $l = 6$. p_1, p_2, \dots, p_6 are the mean relative errors of the first 20%, 40%, 60%, 80%, 90%, 100% of sorted relative errors, respectively. This multi-level performance metric is represented by a multi-level relative mean error vector. l can be set empirically; in this Chapter, we set $l = 6$.

The network traffic data is recorded every day. The neural network is trained according to one day's data, and then used to predict the next day's network traffic. The data are transformed into a $(m + 4)$ -dimensional data set according to Eqn. (6) and Eqn. (7). For example, if $m = 5$, the neural network is trained to predict the average bandwidth of the next 10 seconds according to the preceding 10-second traffic (recall that the original data are recorded every two seconds).

5 Experimental System and Performance Evaluation

5.1 System Hardware and Software

Figure 6 illustrates the prototype system which consists of analysis agents, a web service portal, media servers, and clients. The media servers are Linux PCs with *Fedora Core 4*, running *Darwin Streaming Server* (version 5.5.1); *Darwin Server* – an open source server developed by Apple Computer Inc. – streams media to clients over industry standard protocols (such as RTP, RTSP, RTCP). To monitor certain streaming sessions, users (either the service provider or streaming clients) can query the web server and customize their measurement metrics for quality measurement and assessment. The web portal runs on the web server, which is in charge of accepting/rejecting requests for quality measurement and assessment. Once a request has been accepted by the web portal, it will be allocated to an analysis agent. In our experimental system, each analysis agent is a Java program located on a Linux machine; however, it is suitable for both MS-Windows and Linux platforms. Table 1 summarizes the hardware and software used in the prototype system.

5.2 Request Arrival Pattern

We consider a request rate reported in [2], where a client submits requests to media servers with various probabilities at different time in a 24-hour period. Requests arrive in *Poisson* distribution according to the probability described in Fig. 7. There are three media servers, and 1500 users are randomly generated to stream videos from any server according to the request arrival pattern.

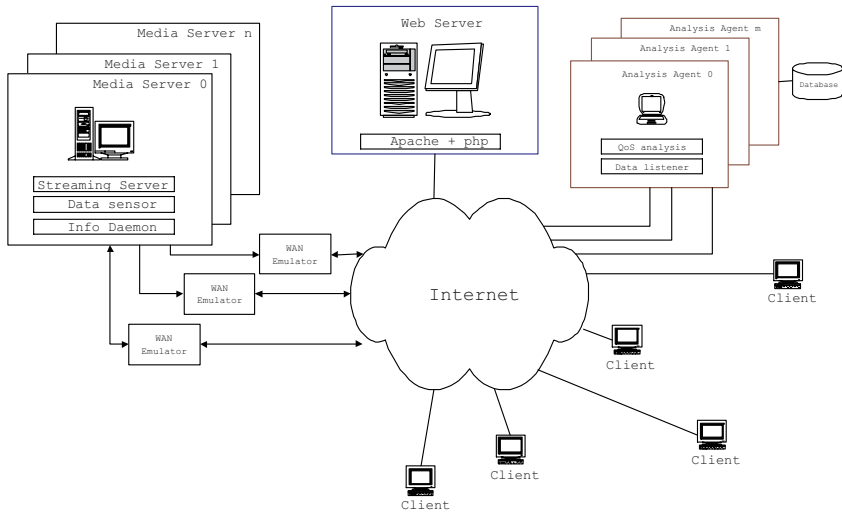


Fig. 6. Experimental system setup

Table 1. Hardware and Software

Function	OS	Memory	Number	Software
Media server	Linux	1GB	3	Darwin server, Data Sensor, Info Daemon
WAN emulator	Linux	1GB	3	Iproute2
Analysis agent	Linux	512	2	Data Collector, QoS Analyzer
Clients	WinXP	256	n	QuickTime, RealPlayer, IBMTToolkitForMPEG4

5.3 Results and Analysis

The data files are generated by **Info Daemon** located at each media server (Fig. 6), which records the bandwidth incoming and outgoing traffic every two seconds. Simple noise detection is carried out by detecting those samples with the extreme values and isolating them.

The neural networks are used to predict the network bandwidth of media streaming in a grid environment. In this work, 20 neurons are used in the first layer, and 10 neurons are used in the second layer of the neural network predictor. The activation function is a sigmoid. The weights of the neural network predictors are determined automatically during the training process.

Figures 8 through 13 show the target and predicted values with $m = 5, 10, 15, 20, 25, 30$, which correspond to bandwidth record sizes of 10s, 20s, 30s, 40s, 50s and 60s, respectively. For one-day data, there are originally

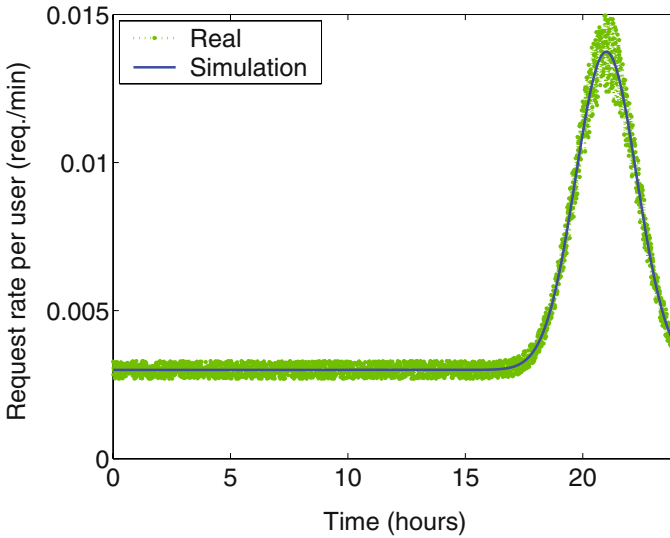


Fig. 7. Request rate per user over 24 hours

$24 \times 60 \times 60 \div 2 = 43,200$ samples. The horizontal axis shows the number of data samples following transformation. A different number of training samples is generated with m . For example, when $m = 5$, there are $43,200 \div 5 = 8,640$ samples with $m + 4 = 9$ dimensions.

From Figs. 8 through 13, it is clear that a bigger window size of bandwidth records leads to a lower prediction accuracy of the mean bandwidth value. The multi-level relative mean error vectors for different values of m are shown in Table 2 and Fig. 14. In the relative mean error vector, there is a sharp increase from 90% sorted relative errors to the whole relative mean error (100%). This means there are isolated relative errors of very large magnitude. This also shows that bigger window size corresponds to lower prediction accuracy.

6 Conclusions

In this Chapter, we presented a practical multi-agent based data analysis system which employs MLP neural networks to predict network traffic over media grids. Instead of using the mean square error and relative mean error as the performance metric for evaluating neural network predictor performance, we proposed a multi-level performance metric which represents the relative mean errors of predictors with a vector. Each item of the vector represents a certain percentage of the relative mean error of the relative prediction errors, sorted in ascending order. The metric can reflect overall performance of the predictors in a multi-level way, at the same time revealing isolated large errors. There is

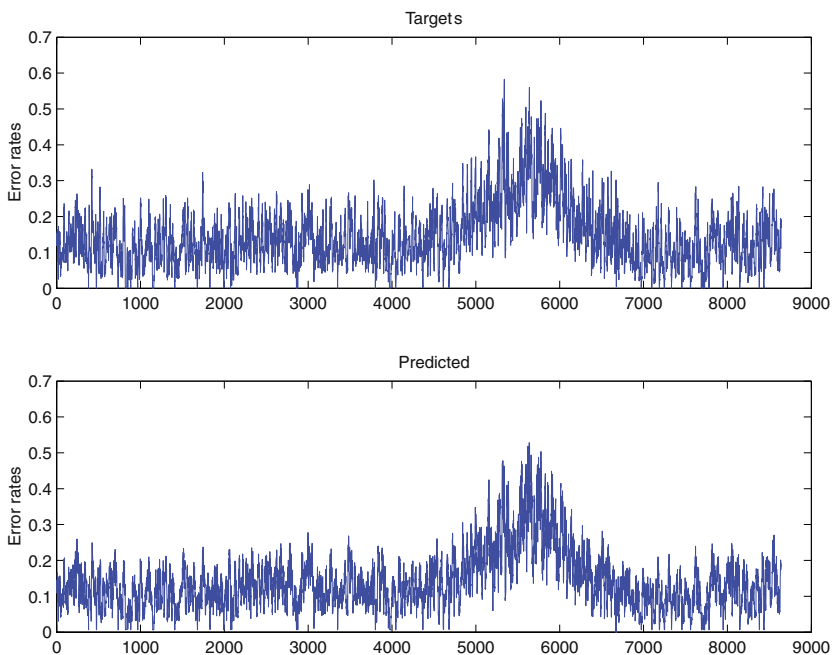


Fig. 8. Target and predicted values with $m = 5$

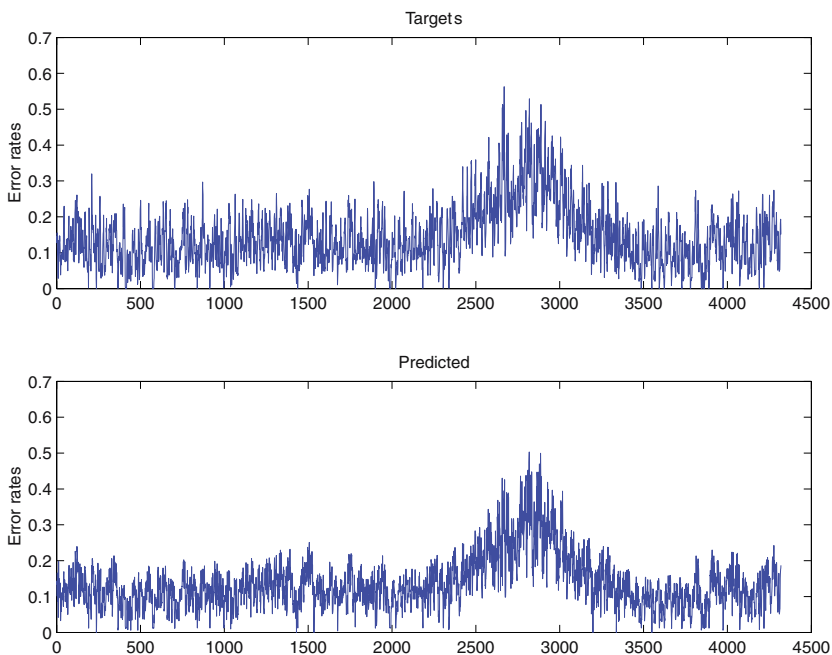


Fig. 9. Target and predicted values with $m = 10$

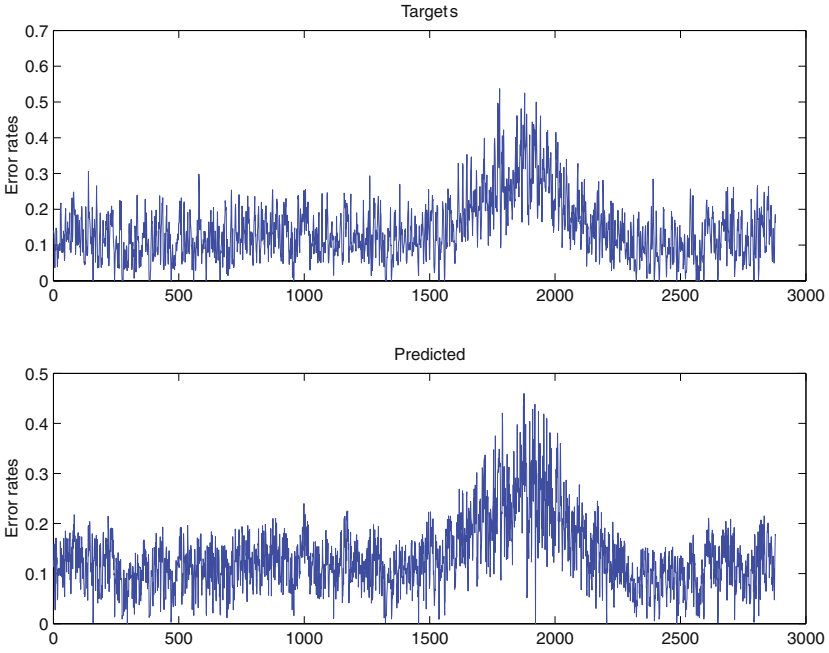


Fig. 10. Target and predicted values with $m = 15$

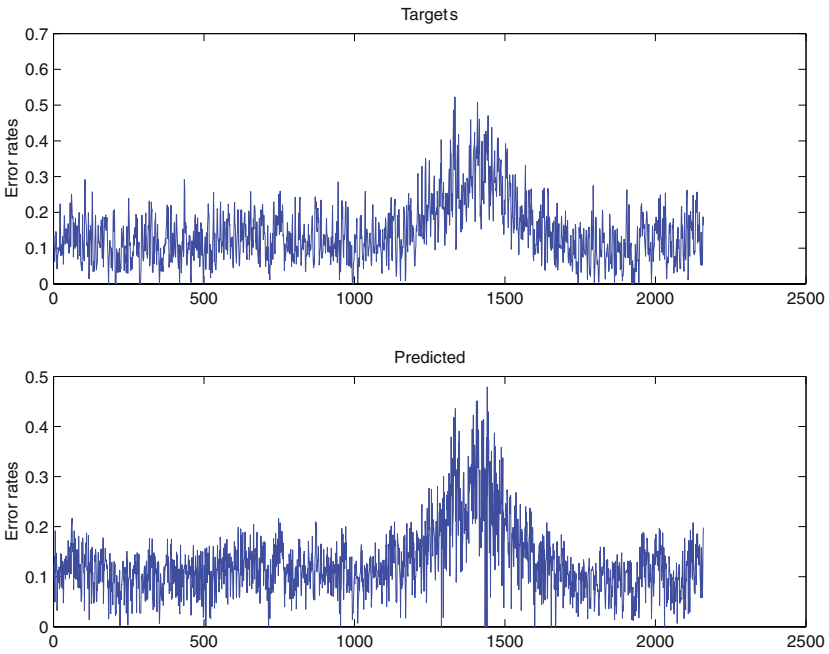


Fig. 11. Target and predicted values with $m = 20$

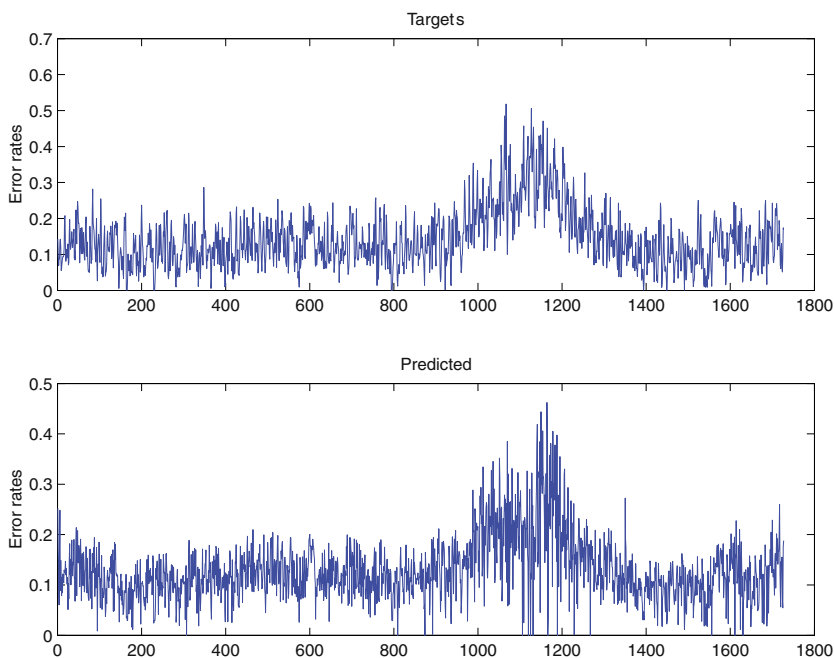


Fig. 12. Target and predicted values with $m = 25$

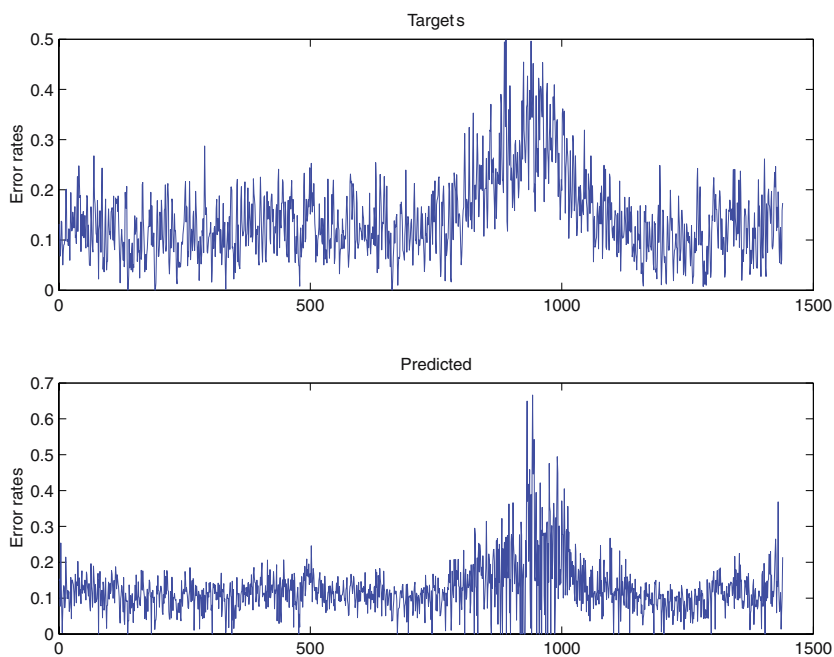


Fig. 13. Target and predicted values with $m = 30$

Table 2. Multi-level mean relative prediction errors

m	20%	40%	60%	80%	90%	100%
5	1.54	3.51	5.69	8.726	11.302	16.231
10	2.302	5.055	8.021	11.928	14.786	20.936
15	2.622	5.852	9.484	14.027	17.143	24.059
20	3.464	7.407	11.567	16.654	19.848	27.282
25	3.771	8.383	13.017	18.577	22.198	31.932
30	4.632	9.95	15.336	21.471	26.068	36.931

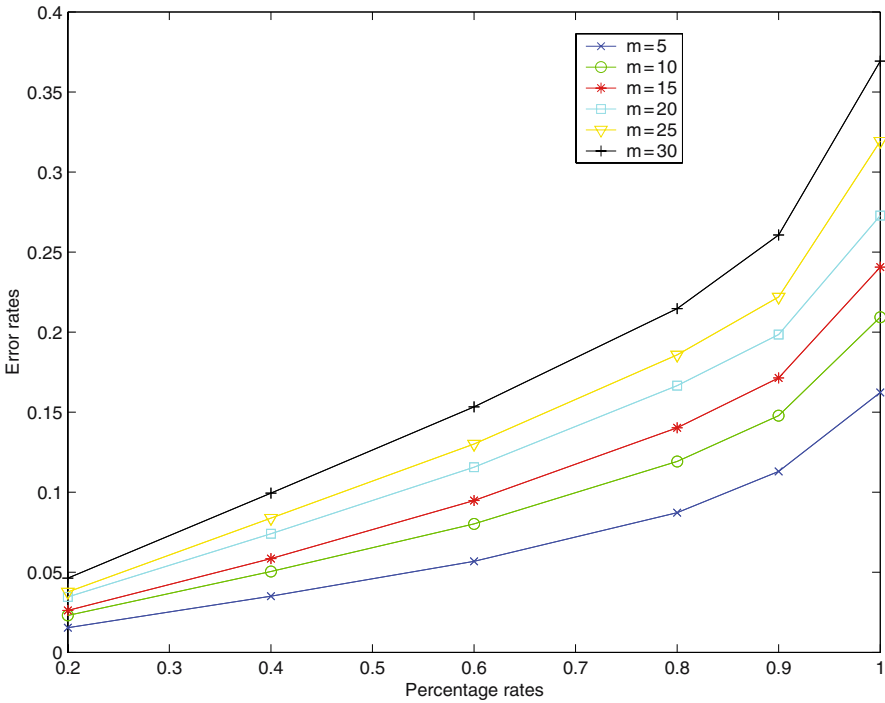


Fig. 14. Multi-level prediction errors with different m

a lot of scope for applying data mining techniques in media grid environments. In our future work, DM techniques will be applied in real time to other media streaming tasks, such as job scheduling and user request pattern analysis.

Acknowledgment

This research is supported by Project IHPC/AC/06-001, and funded by the Institute of High Performance Computing, Singapore.

References

1. Agrawal R, Imielinski T, Swami A (1993) Database mining: a performance perspective. *IEEE Trans. Knowledge and Data Engineering*, 5: 914–925.
2. An de Haar PG, Schoenmakers AF, Eilley ES, Tedd DN, Tickell SA, Lloyd PR, Badham M, O'brien S, Poole R, Sampson P, Harding J, Simula T, Varonen T, Sauvala S (1997) DIAMOND project: video-on-demand system and trials. *European Trans. Telecommunications*, 8(4): 337–244.
3. Apostolopoulos JG, Tan W-T, Wee SJ (2002) Video Streaming: Concepts, Algorithms, and Systems. *Hewlett-Packard White paper* (available online at <http://www.hpl.hp.com/techreports/2002/HPL-2002-260.html> – last accessed 6 September, 2006).
4. Berry MJA, Gordon SL (2000) *Mastering Data Mining: The Art and Science of Customer Relationship Management*. Wiley, New York, NY.
5. Biomedical Informatics Research Network (BIRN) (available online at <http://www.birn.net> – last accessed 6 September, 2006).
6. Bishop CM (1995) *Neural Networks for Pattern Recognition*. Oxford University Press, New York, NY.
7. Carpenter GA and Grossberg S (1988) The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3): 77–88.
8. Cunha JC and Rana OF (2006) *Grid Computing: Software Environments and Tools*. Springer-Verlag, London, UK.
9. Denoeux T, Lengelle R (1993) Initializing back propagation network with prototypes. *Neural Computation*, 6: 351–363.
10. Deutsch JM (2003) Evolutionary algorithms for finding optimal gene sets in microarray prediction. *Bioinformatics*, 19: 45–52.
11. Eswaradass A, Sun X-H, Wu M (2005) A neural network based predictive mechanism for available bandwidth. *Proc. 19th IEEE Intl. Parallel and Distributed Processing Symp.*, 4–8 April, Denver, CO. IEEE Computer Society Press, Piscataway, NJ.
12. EuroGrid (available online at <http://www.eurogrid.org/> – last accessed 6 September, 2006).
13. Foster I, Kesselman C (1998) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA.
14. Fu XJ, Wang LP (2001) Linguistic rule extraction from a simplified RBF neural network. *Computational Statistics* (special issue on Data Mining and Statistics), 16(3): 361–372.
15. Geva S, Wong MT, Orłowski M (1997) Rule extraction from trained artificial neural network with functional dependency preprocessing. *Proc. 1st Intl. Conf. Knowledge-Based Intelligent Engineering Systems*, 21–23 May, Adelaide, South Australia, 2: 559–564.
16. Globus Toolkit (available online at <http://www.globus.org/toolkit/docs/4.0/data/> – last accessed 6 September, 2006).
17. Grid Physics Network (GriPhyn) (available online at <http://www.griphyn.org> – last accessed 6 September, 2006).
18. Halgamuge S, Wang LP (eds.) (2005) *Computational Intelligence for Modeling and Prediction*. Springer-Verlag, Berlin.

19. Hall J, Mars P (1998) The limitations of artificial neural networks for traffic prediction. *Proc. 3rd IEEE Symp. Computers and Communications*, 30 June – 2 July, Athens, Greece. IEEE Computer Society Press, Piscataway, NJ: 147(2): 8–12.
20. Haykin S (1999) *Neural Networks: A Comprehensive Foundation (2nd ed)*. Prentice Hall, Englewood Cliffs, NJ.
21. Jie W, Hung T, Wentong C (2005) An information service for grid virtual organization: architecture, implementation and evaluation. *J. Supercomputing*, 34(3): 273–290.
22. Kolen JF, Pollack JB (1990) Back Propagation is sensitive to initial conditions. *Advances in Neural Information Processing Systems 3* Morgan Kaufmann, San Francisco, CA: 860–867.
23. Lehtokangas M, Saarinen J, Kaski K, Huuhtanen P (1995) Initialization weights of a multilayer perceptron by using the orthogonal least squares algorithm. *Neural Computation*, 7: 982–999.
24. Li X, Hung T, Veeravalli B (2006) Design and implementation of a multimedia personalized service over large scale networks. *Proc. IEEE Intl. Conf. Multimedia & Expo (ICME)*. Toronto, Canada.
25. Liu HB, Mao GQ (2005) Prediction algorithms for real-time variable-bit-rate video. *Proc. Asia-Pacific Conf. Communications*, 3–5 October, Perth, Western Australia: 664–668.
26. Moreau Y, Vandewalle J (1997) When Do Dynamical Neural Networks with and without Hidden Layer Have Identical Behavior? *Technical Report ESAT-SISTA TR97-51*, Dept. Electrical Engineering, Katholieke Universiteit Leuven, Belgium.
27. Partical Physics Data Grid (PPDG) (available online at <http://www.ppdg.net> – last accessed 6 September, 2006).
28. Ramaswamy S, Gburzynski P (1998) A neural network approach to effective bandwidth characterization in ATM networks. In: Kouvatsos D (ed.) *Modeling and Evaluation of ATM Networks* Kluwer Academic Publishers, Boston, MA: 431–450.
29. Rao KR, Bojkovic ZS, Milovanovic DA (2002) *Multimedia Communication Systems: Techniques, Standards, and Networks*. Prentice Hall, Upper Saddle River, NJ.
30. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. *Nature*, 323: 533–536.
31. Sun J, Li HB (2004) 3-D physical motion-based bandwidth prediction for video conferencing *IEEE Trans. Circuits and Systems for Video Technology*, 14(5): 584–594.
32. Teo KK, Wang LP, Lin ZP (2000) Wavelet multi-layer perceptron neural network for time-series prediction. *Proc. 10th Intl. Conf. Computing and Information*, 18–21 November, Kuwait.
33. Ting R (1998) A multiscale Analysis and Analysis Technique for Management of Resources in ATM Networks. *PhD Thesis* School of Engineering, City University of New York, NY.
34. Turner CF, Jeremiah RM (2002) The independent wavelet bandwidth allocation algorithm. *Proc. IEEE Global Telecommunications Conf.* 17–21 November, Taipei, Taiwan, IEEE Computer Society Press, Piscataway, NJ, 1: 107–111.
35. Walsh AE (2005) The media grid: a public utility for digital media. *Dr. Dobb's J. Distributed Computing* (Distributed Computing issue), November: 16–23.

36. Wang LP, Fu, XJ (2005) *Data Mining with Computational Intelligence*. Springer-Verlag, Berlin.
37. Wang LP, Li S, Tian FY, Fu XJ (2004) A noisy chaotic neural network for solving combinatorial optimization problems: Stochastic chaotic simulated annealing. *IEEE Trans. System, Man, Cybernetics, Part B – Cybernetics*, 34(5): 2119–2125.
38. Wang LP, Smith K (1998) On chaotic simulated annealing. *IEEE Trans. Neural Networks*, 9: 716–718.
39. Wang LP, Teo KK, Lin Z (2001) Predicting time series using wavelet packet neural networks. *Proc. Intl. Joint Conf. Neural Networks*, 15–19 July, Washington, DC. IEEE Computer Society Press, Piscataway, NJ: 1593–1597.
40. Wasserman PD (1989) *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, New York, NY.
41. Wolski R, Spring NT, Hayes J (1999) The network weather service: a distributed resource performance forecasting service for metacomputing. *J. Future Generation Computing Systems*, 15(5–6): 757–768.

Resources

1 Key Books

Bishop CM (1995) *Neural Networks for Pattern Recognition*. Oxford University Press, New York

Cunha JC, Rana OF (2006) *Grid Computing: Software Environments and Tools*. Springer, London

Haykin S (1999) *Neural Networks: a Comprehensive Foundation (2nd ed)*. Prentice Hall, Upper Saddle River, NJ

Rao KR, Bojkovic ZS, Milovanovic DA (2002) *Multimedia Communication Systems: Techniques, Standards, and Networks*. Prentice Hall, Upper Saddle River, NJ

Wang LP, Fu, XJ (2005) *Data Mining with Computational Intelligence*. Springer-Verlag, Berlin

Wasserman PD (1989) *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, New York, NY

2 Key Survey/Review Articles

Apostolopoulos JG, Tan W-T, Wee SJ (2002) Video Streaming: Concepts, Algorithms, and Systems. *Hewlett-Packard White Paper*, available online at <http://www.hpl.hp.com/techreports/2002/HPL-2002-260.html> (last accessed 6 September, 2006)

3 Organisations, Societies, Special Interest Groups

IEEE Neural Network Society (publisher of *IEEE Trans. Neural Networks*)

International Neural Network Society (publisher of *Neural Networks* Elsevier)

MIT Press (publisher of *Neural Computation*)

IEEE computer society (publisher of *IEEE Trans. Computers*)

<http://mediagrid.org> (Boston College)

4 Key International Conferences/Workshops

Neural Information Processing Symposium – NIPS (published as *Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Francisco, CA)

International Joint Conference on Neural Networks (IEEE)

International Conference on Multimedia and Expo (IEEE)

IEEE Consumer Communications & Networking Conference

5 (Open Source) Software

Globus Toolkit, available online at <http://www.globus.org/toolkit/docs/4.0/data/> (last accessed 6 September, 2006)

The Self-Organizing Maps: Background, Theories, Extensions and Applications

Hujun Yin

School of Electrical and Electronic Engineering, The University of Manchester,
M60 1QD, UK, hujun.yin@manchester.ac.uk

1 Introduction

For many years, artificial neural networks (ANNs) have been studied and used to model information processing systems based on or inspired by biological neural structures. They not only can provide solutions with improved performance when compared with traditional problem-solving methods, but also give a deeper understanding of human cognitive abilities. Among various existing neural network architectures and learning algorithms, Kohonen's self-organizing map (SOM) [46] is one of the most popular neural network models. Developed for an associative memory model, it is an unsupervised learning algorithm with a simple structure and computational form, and is motivated by the retina-cortex mapping. Self-organization in general is a fundamental pattern recognition process, in which intrinsic inter- and intra-pattern relationships among the stimuli and responses are learnt without the presence of a potentially biased or subjective external influence. The SOM can provide topologically preserved mapping from input to output spaces. Although the computational form of the SOM is very simple, numerous researchers have already examined the algorithm and many of its problems, nevertheless research in this area goes deeper and deeper – there are still many aspects to be exploited.

In this Chapter, we review the background, theories and statistical properties of this important learning model and present recent advances from various pattern recognition aspects through a number of case studies and applications. The SOM is optimal for vector quantization. Its topographical ordering provides the mapping with enhanced fault- and noise-tolerant abilities. It is also applicable to many other applications, such as dimensionality reduction, data visualization, clustering and classification. Various extensions of the SOM have been devised since its introduction to extend the mapping as effective solutions for a wide range of applications. Its connections with other learning paradigms and application aspects are also exploited. The Chapter is intended

to serve as an updated, extended tutorial, a review, as well as a reference for advanced topics in the subject.

2 Background

Kohonen's self-organizing map (SOM) is an abstract mathematical model of topographic mapping from the (visual) sensors to the cerebral cortex. Modeling and analyzing the mapping are important to understanding how the brain perceives, encodes, recognizes and processes the patterns it receives and thus, if somewhat indirectly, are beneficial to machine-based pattern recognition. This Section looks into the relevant biological models, from two fundamental phenomena involved – lateral inhibition and Hebbian learning – to Willshaw and von der Malsburg's self-organization retinotopic model, and then subsequently to Kohonen's simplified and abstracted SOM model. Basic operations and the SOM algorithm, as well as methods for choosing model parameters, are also given.

2.1 Biological Background: Lateral Inhibition and Hebbian Learning

Human visual perception and the brain make up the most complex cognition system and the most complex of all biological organs. Visual inputs contribute to over 90% of the total information (from all sensors) entering the brain. Nature and our living environment are constantly shaping our perception and cognition systems. Physiologically, human and indeed other animal visual (and other perception) systems have been evolved to so many different types of eyes and mammalian visual pathways for different purposes and conditions. For example, many insects have compound eyes, which have good temporal resolution and are more directionally sensitive and at the same time make them smaller and group them into a single structure – giving insects a better ability to detect spatial patterns and movement in order to escape from predators. Compound eyes have good time resolving power. Human eyes need 0.05 second to identify objects, while compound eyes need only 0.01 second. That is, they are good at recognizing (fast) moving objects. Eyes of large animals including humans have evolved to single-chambered 'camera lens' eyes, which have excellent angle resolution and are capable of seeing distant objects. Camera eyes have great space resolving power: high spatial resolution for good details of objects and patterns, and long depth resolution for both very near and very far objects. They also have brilliant sensitivities for light intensity – over 20 billion times (that is, 206 dB) range (the brightest to the darkest) – compared with most digital cameras, which are below 16-bit resolution (30 dB).

What information do eyes extract from the retina or sensory cells? Visual information is processed in both the retina and brain, but it is widely believed

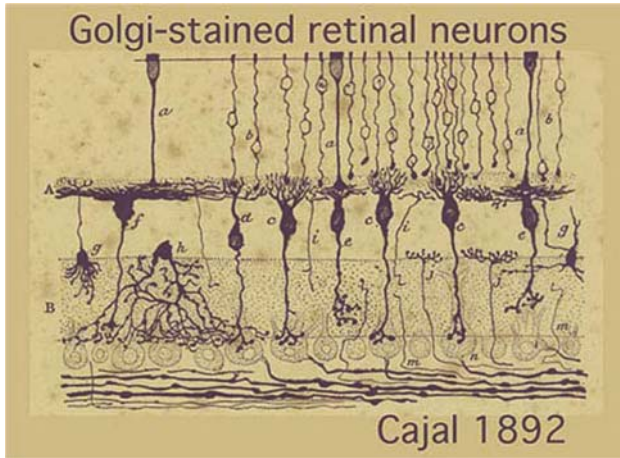


Fig. 1. A cross-sectional diagram of the retina, drawn by Santiago Ramón y Cajal (1853–1934)

and verified that most processing is done in the retina, such as extracting lines, angles, curves, contrasts, colours, and motion. The retina then encodes the information and sends through optic nerves and optic chiasma, where some left and right nerves are crossed, to the brain cortex in the left and/or right hemispheres. The retina is a complex neural network. Figure 1 shows a drawing of the cross section of the retina. The human retina has over 100 million photosensitive cells (combining rods and cones), processing in parallel the raw images, codes and renders to just over one million optic nerves, to be transmitted in turn to the brain cortex.

The Perceptron models some cells in the retina, especially the bipolar and ganglion cells. These cells take inputs from the outputs of cells in the previous layer. To put many units together and connect them into layers, one may hope the resulting network – the Multi-Layer Perceptron – will have some functionality similar to the retina (despite neglecting some horizontal interconnections). Indeed, such a structure has been demonstrated as being capable of certain cognitive and information processing tasks.

Cells in neural networks (either in the retina or brain) also connect and interact horizontally. The experiment on limulus, or the horseshoe crab, by Haldan K. Hartline (1967 Nobel Prize Laureate) and his colleagues in the 1960s, has confirmed such processing on the limulus retina (the surface of the compound eye is shown in Fig. 2(a)). They revealed the so-called ‘lateral inhibition’ activity among the retina cells. In other words, there exist both short-range excitatory interaction between nearby cells, as well as long-range inhibitory interaction between distant neighbours. This consequently explains the so-called ‘Mach band’ phenomenon on the edges or sharp changes of light intensity [87] – see Fig. 2(b).

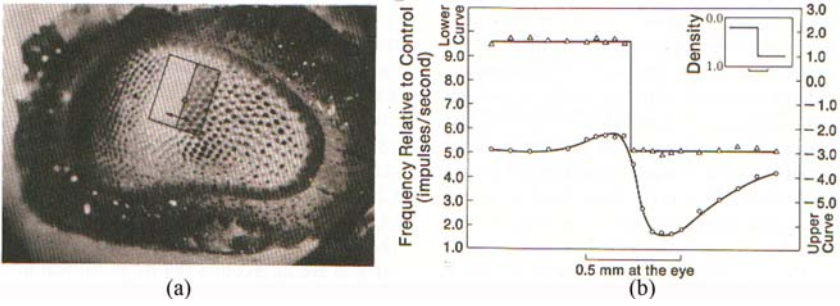


Fig. 2. (a) Surface of the Limulus eye and simuli: small spot light and rectangular lighter/darker pattern; (b) recordings of spike rate in the ommatidium axon (the upper curve is the response to the small spot light at high and low intensities corresponding to those of the test pattern in the insert; the lower curve is the response to the rectangular lighter/darker test pattern (from [87]; also see [99])



Fig. 3. Artistic drawing of a woman (*left*), and applying Pearson and Ronbinson's edge detector – an improved Marr and Hildreth mask – on the photo of the same woman (*right*) ([86], reprinted by permission of Cambridge University Press; also cited in [9])

Lateral inhibition tells us that neurons in the retina do not just feed the information to upper levels, but also perform an important visual processing task: edge detection and enhancement. Figure 3 demonstrates a psychological experiment that also confirms such fundamental processing in visual perception.

Neural networks present completely different approaches to computing and machine intelligence from traditional symbolic AI. The goal is to emulate the way that natural systems, especially brains, perform on various cognitive or recognition tasks. When a network of simple processing units interconnect with each other, there are potentially a massive number of synaptic weights available to be configured and modified such that the network will suit a

particular task. This configuration and modification process is carried out by a learning procedure, that is, learning or training algorithm. The way these simple units connect together is called the neural architecture. There are two basic types: feed-forward, in which layers of neurons are concatenated, and recurrent, in which neurons have feedback from themselves and others. Examples of these two architectures are the Multi-Layer Perceptron (MLP), and the Hopfield network, respectively.

The Oxford English Dictionary defines learning as “the process which leads to the modification of behavior”.¹ Learning in general intelligent systems is often referred to as a process of the systems’ adaptation to their experience or environment – a key feature of intelligence. According to Hebb, learning occurs when “some growth process or metabolic change takes place” [30]. Learning in the context of neural networks can be defined as [29]:

“Learning is a process by which the free parameters of neural networks are adapted through a process of simulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.”

Neural networks also differ from traditional pattern recognition approaches, which usually require solving some well-defined functions or models, such as feature extraction, transformation, and discriminant analysis by a series of processing steps. Neural networks can simply learn from examples. Presented repeatedly with known examples of raw patterns and with an appropriate learning or training algorithm, they are able to extract the most intrinsic nature of the patterns and perform recognition tasks. They will also have the ability to carry out similar recognition tasks, not only on trained examples but also on unseen patterns. Learning methods and algorithms undoubtedly play an important role in building successful neural networks.

Although many learning methods have been proposed, there are two fundamental kinds of learning paradigms: supervised learning and unsupervised learning. The former is commonly used in most feed-forward neural networks, in which the input-output (or input-target) functions or relationships are built from a set of examples, while the latter resembles a self-organization process in the cortex and seeks inter-relationships and associations among the input.

The most representative supervised learning rule is error-correction learning. When presented with an input-output pair, learning takes place when an error exists between a desired response or target output and the actual output of the network. This learning rule applies an adjustment, proportional to this error, to the weights of the neuron concerned. Derivation of such a rule can be often traced backed to minimizing the mean-square error function – more details can be found in [29]. A derivative of supervised learning is so-called

¹ Simpson JA, Weiner ESC (eds.) (1988) *Oxford English Dictionary (2nd ed.)*. Clarendon Press, Oxford, UK.

reinforcement learning, based on trail-and-error (and reward). The following definition has been given by [101]:

“If an action taken by a learning system is followed by a satisfactory state of affairs, then the tendency of the system to produce that particular action is strengthened or reinforced. Otherwise, the tendency of the system to produce that action is weakened”.

In contrast to supervised learning, there is no direct teacher to provide how much output error a particular action has produced. Instead, the output has been quantified into either ‘positive’ or ‘negative’ corresponding to closer to or further from the goal. Reinforcement learning has popular backing from psychology.

Self-organization often involves both competition and correlative learning. When presented with a stimulus, neurons compete among themselves for possession or ownership of this input. The winners then strengthen their weights or their relationships with this input. Hebbian learning is the most common rule for unsupervised or self-organized learning. As stated in [30]:

“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A s efficiency as one of the cells firing B , is increased.”

Mathematically, the Hebbian learning rule can be directly interpreted as,

$$\frac{\partial w_{ij}(t)}{\partial t} = \alpha x_i(t)y_i(t) \quad (1)$$

where α is a positive learning rate ($0 < \alpha < 1$), and x and y are the input and output of the neural system, respectively (or can also be regarded as the outputs of the two neurons). That is, the change of the synaptic weight is proportional to the correlation between an input and its associated output. If the input and output are coherent, the weight connecting them is strengthened (xy is positive), otherwise, weakened (xy is either negative or zero).

Hebbian learning requires some modification before it can be used in practice, otherwise the weight will easily become saturated or unlimited (positive or negative). One solution is to add a ‘forgetting term’ to prevent weights from increasing/decreasing monotonically as in the SOM (see the next Section). Alternatively, we can normalize the weights. For instance, Oja proposed a weight normalization scheme on all weights. This naturally introduces a forgetting term to the Hebbian rule [81],

$$\begin{aligned} w_i(t+1) &= \frac{w_i(t) + \alpha x_i(t)y(t)}{\{\sum_{j=1}^n [w_j(t) + \alpha x_j(t)y(t)]^2\}^{1/2}} \\ &\approx w_i(t) + \alpha(t)[x_i(t) - y(t)w_i(t)] + O(\alpha^2) \end{aligned} \quad (2)$$

where $O(\alpha^2)$ represents second- and high-order terms in α , and can be ignored when a small learning rate is used.

The resulting Oja learning algorithm is a so-called principal component network, which learns to extract the most variant directions among the data set. Other variants of Hebbian learning include many algorithms used for Independent Component Analysis ([36, 82]).

2.2 From Von Malsburg and Willshaw's Self-Organization Model to Kohonen's SOM

External stimuli are received by various sensors or receptive fields (for example, visual-, auditory-, motor-, or somato-sensory), coded or abstracted by the living neural networks, and projected through axons onto the cerebral cortex, often to distinct parts of the cortex. In other words, the different areas of the cortex (cortical maps) often correspond to different sensory inputs, though some brain functions require collective responses. Topographically ordered maps are widely observed in the cortex. The main structures (primary sensory areas) of the cortical maps are established before birth ([47], [115], and similar) in a predetermined topographically ordered fashion. Other more detailed areas (associative areas), however, are developed through self-organization gradually during life and in a topographically meaningful order. Therefore studying such topographically ordered projections, which had been ignored during the early period of neural information processing research [48], is undoubtedly important for understanding and constructing dimension-reduction mapping and for the effective representation of sensory information and feature extraction.

The self-organized learning behavior of brains has been studied for a long time by many people. Pioneering works include for example, Hebb's learning law (1949) [30], Marr's theory of the cerebellar cortex (1969) [72], Willshaw, Buneman and Longnet-Higgins's non-holographic associative memory (1969) [114], Gaze's studies on nerve connections (1970), von der Malsburg and Willshaw's self-organizing model of retina-cortex mapping ([111], [115]), Amari's mathematical analysis of self-organization in the cortex (1980), Kohonen's self-organizing map (1982), and Cottrell and Fort's self-organizing model of retinotopy (1986). Many still have immense influence on today's research. Von der Malsburg (1973) and Willshaw (1976) first developed, in mathematical form, the self-organizing topographical mapping, mainly from two-dimensional presynaptic sheets to two-dimensional postsynaptic sheets, based on retinatopic mapping: the ordered projection of visual retina to the visual cortex (see Fig. 4). Their basic idea was:

“.....the geometrical proximity of presynaptic cells is coded in the form of correlations in their electrical activity. These correlations can be used in the postsynaptic sheet to recognize axons of neighbouring presynaptic cells and to connect them to neighbouring postsynaptic cells, hence producing a continuous mapping.....”

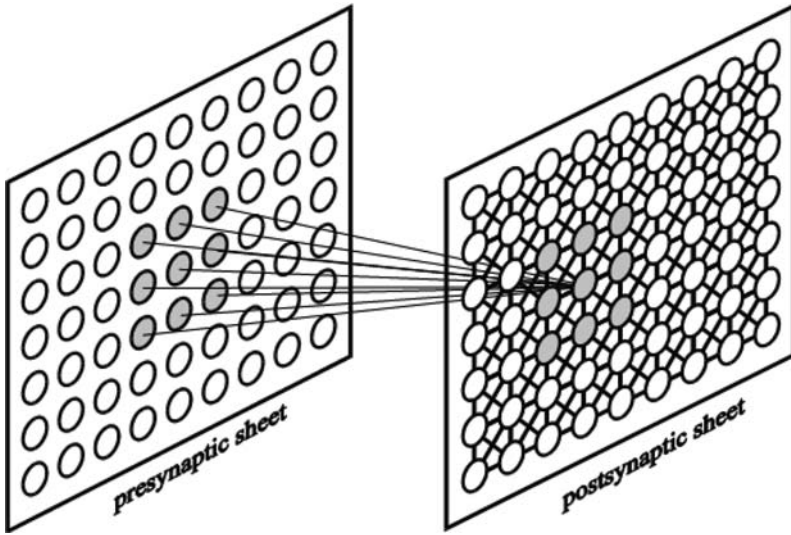


Fig. 4. von der Malsburg’s self-organizing map model: local clusters in a presynaptic sheet are connected to local clusters in a postsynaptic sheet; there are lateral interconnections within the postsynaptic sheet (solid lines are used to indicate such connections)

The model uses short-range excitatory connections between cells so that activity in neighbouring cells becomes mutually reinforced, and uses long-range inhibitory interconnections to prevent activity from spreading too far. The postsynaptic activities $\{y_j(t), j = 1, 2, \dots, N_y\}$, at time t , are expressed by

$$\frac{\partial y_i(t)}{\partial t} + cy_i(t) = \sum_j w_{ij}(t)x_i(t) + \sum_k e_{ik}y_k^*(t) - \sum_{k'} b_{ik'}y_{k'}^*(t) \quad (3)$$

where c is the membrane constant, $w_{ij}(t)$ is the synaptic strength between cell i and cell j in pre- and post-synaptic sheets respectively, $\{x_i(t), i = 1, 2, \dots, N_x\}$, the state of the presynaptic cells, equal to 1 if cell i is active or 0 otherwise, e_{kj} and b_{kj} are short-range excitation and long-range inhibition constants respectively, and $y_j^*(t)$ is an active cell in postsynaptic sheet at time t . The postsynaptic cells fire if their activity is above a threshold, say,

$$y_i^*(t) = \begin{cases} y_j^*(t) - \theta, & \text{if } y_j^*(t) > \theta \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

The modifiable synaptic weights between pre- and post-synaptic sheets are then facilitated in proportion to the product of activities in the appropriate pre- and postsynaptic cells (Hebbian learning):

$$\frac{\partial w_{ij}(t)}{\partial t} = \alpha x_i(t)y_j^*(t), \quad \text{subject to } \frac{1}{N_x} \sum_i w_{ij} = \text{constant} \quad (5)$$

where α is a small constant representing the learning rate. To prevent the synaptic strengths becoming unstable, the total strength associated with each postsynaptic cell is limited by normalization to a constant value after each iteration.

Kohonen (1982) abstracted the above self-organizing learning principle and function and proposed a much simplified learning mechanism which cleverly incorporates the Hebbian learning rule and lateral interconnection rules and can emulate the self-organizing learning effect. Although the resulting SOM algorithm was more or less proposed in a heuristic manner ([54]), it is a simplified and generalized model of the above self-organization process. As commented in [91]:

“Kohonen’s model of self-organizing maps represented an important abstraction of the earlier model of von der Malsburg and Willshaw; the model combines biological plausibility with proven applicability in a broad range of difficult data processing and optimization problems...”

In Kohonen’s model, the postsynaptic activities are similar to Eqn. (3). To find the solutions of this equation and ensure they are non-negative properties, a sigmoid type of nonlinear function is applied to each postsynaptic activity:

$$y_j(t+1) = \varphi \left[\mathbf{w}_j^T \mathbf{x}(t) + \sum_i h_{ij} y_i(t) \right] \quad (6)$$

where h_{kj} is similar to e_{kj} and b_{kj} , and the input is described as a vector as the map can be extended to any dimensional input. A typical structure is shown in Fig. 5.

A spatially-bounded cluster or bubble will then be formed among the postsynaptic activities and will stabilize at a maximum (without loss of generality which is assumed to be unity) when within the bubble, or a minimum (that is, zero) otherwise,

$$y_j(t+1) = \begin{cases} 1, & \text{if neuron } j \text{ is inside the bubble} \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

The bubble is centred on a postsynaptic cell whose synaptic connection with the presynaptic cells is mostly matched with the input or presynaptic state, that is the first term in the function in Eqn. (6) is the highest. The range or size, denoted by $\eta(t)$, of the bubble depends on the ratio of the lateral excitation and inhibition. To modify the Hebbian learning rule, in other words Eqn. (5), instead of using normalization, a forgetting term, $\beta y_j(t) w_{ij}(t)$, is added. Let $\alpha = \beta$, and apply Eqn. (7), the synaptic learning rule can then be formulated as,

$$\begin{aligned} \frac{\partial w_{ij}(t)}{\partial t} &= \alpha y_j(t) x_i(t) - \beta y_j(t) w_{ij}(t) = \alpha [x_i(t) - w_{ij}(t)] y_j(t) \\ &= \begin{cases} \alpha [x_i(t) - w_{ij}(t)], & \text{if } j \in \eta(t) \\ 0 & \text{if } j \notin \eta(t) \end{cases} \end{aligned} \quad (8)$$

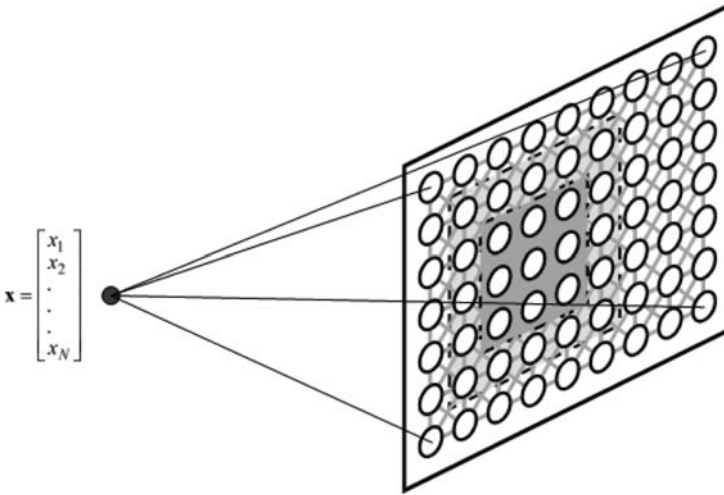


Fig. 5. Kohonen's self-organizing map model. The input is connected to every cell in the postsynaptic sheet (the map). The learning makes the map localized, in other words different local fields will respond to different ranges of inputs. The lateral excitation and inhibition connections are emulated by a mathematical modification, namely local sharing, to the learning mechanism (so there are no actual connections between cells – grey lines are used to indicate these virtual connections)

At each time step the best matching postsynaptic cell is chosen according to the first term of the function in Eqn. (6), which is the inner product, or correlation, of the presynaptic input and synaptic weight vectors. When normalization is applied to the postsynaptic vectors, as it usually is, this matching criterion is similar to the Euclidean distance measure between the weight and input vectors. Therefore the model provides a very simple computational form. The lateral interconnection between neighbouring neurons and the 'Mexican-hat' excitatory or inhibitory rules are simulated (mathematically) by a simple local neighbourhood excitation centred on the winner. Thus the neuron's lateral interconnections (both excitatory and inhibitory) have been replaced by neighbourhood function adjustment. The neighbourhood function's width can emulate the control of the exciting and inhibiting scalars. The constrained (with a decaying or forgetting term) Hebbian learning rule has been simplified and becomes a competitive learning model.

Most of Kohonen's work has been in associative memories ([43]–[48], and so on). In his studies, he has found that the spatially ordered representation of sensory information in the brain is highly related to the memory mechanism, and that the inter-representation and information storage can be implemented simultaneously by an adaptive, massively parallel, and self-organizing network [48]. This simulated cortex map, on the one hand can perform a self-organized search for important features among the inputs, and on the other hand can

arrange these features in a topographically meaningful order. This is why the map is also sometimes termed the ‘self-organizing feature map’, or SOFM. In this Chapter, however, it will be referred to by its commonly used term, the ‘self-organizing map’ (SOM), which comes from Kohonen’s original definition and purpose, namely associative memory.

2.3 The SOM Algorithm

The SOM uses a set of neurons, often arranged in a 2-D rectangular or hexagonal grid, to form a discrete topological mapping of an input space, $\mathbf{X} \in \mathcal{R}^n$. At the start of the learning, all the weights $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_M\}$ are initialized to small random numbers. \mathbf{w}_i is the weight vector associated to neuron i and is a vector of the same dimension – n – of the input, M is the total number of neurons, and let \mathbf{r}_i be the location vector of neuron i on the grid. Then the algorithm repeats the steps shown in Algorithm 1, where $\eta(\nu, k, t)$ is the neighbourhood function, and Ω is the set of neuron indexes. Although one can use the original stepped or top-hat type of neighbourhood function (one when the neuron is within the neighbourhood; zero otherwise), a Gaussian form is often used in practice – more specifically $\eta(\nu, k, t) = \exp[-\frac{\|\mathbf{r}_\nu - \mathbf{r}_k\|^2}{2\sigma(t)^2}]$, with σ representing the effective range of the neighbourhood, and is often decreasing with time.

Algorithm 1 Self-Organizing Map algorithm

repeat

1. At each time t , present an input $\mathbf{x}(t)$, and select the winner,

$$\nu(t) = \arg \min_{k \in \Omega} \|\mathbf{x}(t) - \mathbf{w}_k(t)\| \quad (9)$$

2. Update the weights of the winner and its neighbours,

$$\Delta \mathbf{w}_k(t) = \alpha(t) \eta(\nu, k, t) [\mathbf{x}(t) - \mathbf{w}_\nu(t)] \quad (10)$$

until the map converges

The coefficients $\{\alpha(t), t \geq 0\}$, termed the ‘adaptation gain’, or ‘learning rate’, are scalar-valued, decrease monotonically, and satisfy [47]:

$$(i) 0 < \alpha(t) < 1; \quad (ii) \lim_{t \rightarrow \infty} \sum \alpha(t) \rightarrow \infty; \quad (iii) \lim_{t \rightarrow \infty} \sum \alpha^2(t) < \infty; \quad (11)$$

They are the same as to those used in stochastic approximation ([92, 94]). The third condition in Eqn. (11) has been relaxed by Ritter and Schulden to a less restrictive one, namely, $\lim_{t \rightarrow \infty} \alpha(t) \rightarrow 0$ [90].

If the inner product similarity measure is adopted as the best matching rule,

$$\nu(t) = \arg \min_{k \in \Omega} [\mathbf{w}_k^T \mathbf{x}(t)] \quad (12)$$

then the corresponding weight updating will become [53]:

$$\mathbf{w}_k(t+1) = \begin{cases} \frac{\mathbf{w}_k(t) + \alpha(t)\mathbf{x}(t)}{\|\mathbf{w}_k(t) + \alpha(t)\mathbf{x}(t)\|} & k \in \eta_\nu \\ \mathbf{w}_k(t) & k \notin \eta_\nu \end{cases} \quad (13)$$

Such a form is often used in text/document mining applications (for example, [23]).

The SOM algorithm vector-quantizes or clusters the input space and produces a map which preserves topology. It can also be and has been used for classification. In this case, the map is trained on examples of known categories. The nodes are then classified or labelled so that the map can be used to classify unseen samples. Various methods for labelling nodes can be found in [59]. The classification performance can be further improved by the Learning Vector Quantization (LVQ) algorithm [53].

3 Theories

3.1 Convergence and Cost Functions

Although the SOM algorithm has a simple computational form, a formal analysis of it and the associated learning processes and mathematical properties is not easily realized. Some important issues still remain unanswered. Self-organization, or more specifically the ordering process, has been studied in some depth; however a universal conclusion has been difficult, if not impossible, to obtain. This Section reviews and clarifies the statistical and convergence properties of the SOM and associated cost functions, the issue that still causes confusions to many even today. Various topology preservation measures will be analyzed and explained.

The SOM was proposed to model the sensory-to-cortex mapping thus the unsupervised associative memory mechanism. Such a mechanism is also related to vector quantization (VQ) [63] in coding terms. The SOM has been shown to be an asymptotically optimal VQ [117, 126]. More importantly, with its neighbourhood learning, the SOM is both an error tolerant VQ and Bayesian VQ [66, 68].

Convergence and ordering has only been formally proven in the trivial one-dimensional case. A full proof of both convergence and ordering in multidimensional systems is still outstanding, although there have been several attempts (for instance, [19, 20, 62, 64, 90, 126]). [19] and [20] especially showed that there was no cost function that the SOM will follow exactly. Such an issue

is also linked to the claimed lack of an exact cost function that the algorithm follows. Recent work by various researchers has already shed light on this intriguing issue surrounding the SOM. Yin and Allison extended the Central Limit Theorem and used it to show that when the neighbourhood reduces to just the winner, the weight vectors (code references) are asymptotically Gaussian distributed and will converge in a mean squares sense to the means of the Voronoi cells – in other words, an optimal VQ (with the SOM's nearest distance winning rule) [126],

$$\mathbf{w}_k \rightarrow \frac{1}{P(X_k)} \int_{V_k} \mathbf{x} p(\mathbf{x}) d\mathbf{x} \quad (14)$$

where V_k is the Voronoi cell (the data region) for which the weight vector w_k is responsible, and $p(\mathbf{x})$ is the probability density function of the data. In general cases with the effect of the neighbourhood function, the weight vector is a kernel smoothed mean [119],

$$\mathbf{w}_k \rightarrow \frac{\sum_{t=1}^T \eta(\nu, k, t) \mathbf{x}(t)}{\sum_{t=1}^T \eta(\nu, k, t)} \quad (15)$$

Yin and Allison have also proved that the initial state has a diminishing effect on the final weights when the learning parameters follow the convergence conditions [126]. Such an effect has been recently verified by [15] using Monte-Carlo bootstrap cross validation; the ordering was not considered. In practice, as only limited data samples are used and training is performed in finite time, good initialization can help guide to a faster or even better convergence. For example, initializing the map to a principal linear sub-manifold can reduce the ordering time, if the ordering process is not a key requirement.

Luttrell first related hierarchical noise tolerant coding theory to the SOM. When the transmission channel noise is considered, a two-stage optimization has to be done, not only to minimize the representation distortion (as in VQ) but also to minimize the distortion caused by the channel noise. He revealed that the SOM can be interpreted as such a coding algorithm. The neighbourhood function acts as the model for the channel noise distribution and should not go to zero as in the original SOM. Such a noise tolerant VQ has the following objective function ([66, 67]),

$$D_2 = \int d\mathbf{x} p(\mathbf{x}) \int d\mathbf{n} \pi \|\mathbf{x} - \mathbf{w}_k\|^2 \quad (16)$$

where \mathbf{n} is the noise variable and $\pi(\mathbf{n})$ is the noise distribution. [18] and [78] have also linked the SOM and this noise tolerant VQ with minimal wiring of cortex-like maps.

When the code book (the map) is finite, the noise can be considered as discrete, then the cost function can be re-expressed as,

$$D_2 = \sum_i \int_{V_i} \sum_k \pi(i, k) \| \mathbf{x} - \mathbf{w}_k \|^2 p(\mathbf{x}) d\mathbf{x} \quad (17)$$

where V_i is the Voronoi region of cell i . When the channel noise distribution is replaced by a neighbourhood function (analogous to inter-symbol dispersion), it becomes the cost function of the SOM algorithm. The neighbourhood function can be interpreted as a channel noise model. Such a cost function has been discussed in the SOM community (for example, [32], [50], [58], [117]). The cost function is therefore,

$$E(\mathbf{w}_1, \dots, \mathbf{w}_N) = \sum_i \int_{V_i} \sum_k \eta(i, k) \| \mathbf{x} - \mathbf{w}_k \|^2 p(\mathbf{x}) d\mathbf{x} \quad (18)$$

This leads naturally to the SOM update algorithm using the sample or stochastic gradient descent method [92] – that is, for each Voronoi region, the sample cost function is,

$$\hat{E}_i(\mathbf{w}_1, \dots, \mathbf{w}_N) = \int_{V_i} \sum_k \eta(i, k) \| \mathbf{x} - \mathbf{w}_k \|^2 p(\mathbf{x}) d\mathbf{x} \quad (19)$$

The optimization for all weights $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_N\}$ can be sought using the sample gradients. The sample gradient for \mathbf{w}_j is,

$$\frac{\partial \hat{E}_i(\mathbf{w}_1, \dots, \mathbf{w}_N)}{\partial \mathbf{w}_j} = \frac{\partial \sum_k \eta(i, k) \| \mathbf{x} - \mathbf{w}_k \|^2}{\partial \mathbf{w}_j} = 2\eta(i, k) \| \mathbf{x} - \mathbf{w}_j \| \quad (20)$$

which leads to the SOM update rule – Eqn.(10). Note that although the neighbourhood function $\eta_{i,k}$ is only implicitly related to \mathbf{w}_j , it does not contribute to the weight optimization, nor does the weight optimization lead to its adaptation (neighbourhood adaptation is often controlled by a pre-specified scheme, unrelated to the weight adaptation); thus the neighbourhood can be omitted from the partial differentiation. This point has caused problems in interpreting the SOM cost function in the past.

It has however been argued that this energy function is violated at boundaries of Voronoi cells where input has exactly the same smallest distance to two neighbouring neurons. Thus this energy function holds mainly for the discrete case where the probability of such boundary input points is close to zero or the local (sample) cost function \hat{E}_i should be used in deciding the winner [32]. When a spatial-invariant neighbourhood function is used (as is often the case), assigning the boundary input to either cell will lead to the same local sample cost (or error), therefore any input data on the boundary can be assigned to either Voronoi cells that have the same smallest distance to it, just as in the ordinary manner (on a first-come-first-served fashion, for example). Only when the neurons lie on the map borders does such violation occur, due to unbalanced neighbourhood neurons. The result is slightly more contraction towards to the centre (inside) of the map for the border neurons,

compared to the common SOM algorithm, as shown in [50]. Using either the simple distance or local distortion measure as the winning rule will result in border neurons being contracted towards the centre of the map, especially when the map is not fully converged or when the effective range of the neighbourhood function is large. With the local distortion rule, this boundary effect is heavier as greater local error is incurred at the border neurons due to their few neighbouring neurons compared with any inside neurons.

To follow the cost function exactly, the winning rule should be modified to follow the local sample cost function \hat{E}_i (or the local distortion measure) instead of the simplest nearest distance, that is,

$$\nu = \arg \min_i \sum_k \eta(i, k) \| \mathbf{x} - \mathbf{w}_k \|^2 \tag{21}$$

When the neighbourhood function is symmetric (as is often the case), and when the data density function is smooth, this local distortion winning rule is the same as the simplest nearest distance rule for most non-boundary nodes, especially if the number of nodes is large. On the map borders, however, differences exist due to the imbalance of nodes present in the neighbourhoods. Such differences become negligible to the majority of the neurons, especially when a large map is used, and when the neighbourhood function shrinks to its minimum scale.

3.2 Topological Ordering

The ordering to a large extent is still an outstanding and subtle issue, largely due to the fact that there is no clear (or agreed) definition of ‘order’ [25]. This is the very reason why a full self-organization convergence theorem including both statistical convergence, ordering, and the exact cost function, is still subject to debate – which has prompted many alternatives, such as [8], [26], and [100]. The ordering and ordered map are clearly defined only in the 1-D trivial case. Extension to higher dimensions proves difficult, if not impossible. [7] have proposed a measure called topology product to measure the topological ordering of the map,

$$P = \frac{1}{N^2 - N} \sum_i \sum_j \log \left(\prod_{l=1}^j \frac{d^D(\mathbf{w}_i, \mathbf{w}_{\eta^O(l,i)}) d^O(i, \eta^O(l,i))}{d^D(\mathbf{w}_i, \mathbf{w}_{\eta^D(l,i)}) d^O(i, \eta^D(l,i))} \right)^{\frac{1}{2k}} \tag{22}$$

where d^D and d^O represent the distance measures in the input (or data) space, and output (or map) space, respectively; $\eta(l, i)$ represents the l th neighbour of node i in either data (D) or map (O) space.

The first ratio in the product measures the ratio or match of weight distance sequences of a neighbourhood (up to j) on the map and in the data

space. The second ratio is the index distance sequences of the neighbourhood on the map and in the data space. The topographic product measures the product of the two ratios of all possible neighbourhoods.

[109] proposed a topographic function to measure the ‘neighbourhood-ness’ of weight vectors in data space as well as on the lattice. The neighbourhood-ness of the weight vectors is defined by the adjacent Voronoi cells of the weights. The function measures the degree to which the weight vectors are ordered in the data space as to their indexes on the lattice, as well as how well the indexes are preserved when their weight vectors are neighbours.

Defining a fully ordered map can be straightforward using the distance relations [117]. For example, if all the nearest neighbour nodes (on the lattice) have their nearest neighbour nodes’ weights in their nearest neighbourhood in the data space, we can call the map a 1st-order (ordered) map [117], that is,

$$d(\mathbf{w}_i, \mathbf{w}_j) \leq d(\mathbf{w}_i, \mathbf{w}_k), \quad \forall i \in \Omega; j \in \eta_i^1; k \notin \eta_i^1 \tag{23}$$

where Ω is the map, and η_i^1 denotes the 1st-order neighbourhood of node i .

Similarly if the map is a 1st-order ordered map, and all the 2nd nearest neighbouring nodes (on the lattice) have their 2nd nearest neighbouring nodes’ weights in their 2nd nearest neighbourhood, we can call the map is a 2nd-order (ordered) map. For the 2nd ordered map, the distance relations to be satisfied are,

$$d(\mathbf{w}_i, \mathbf{w}_j) \leq d(\mathbf{w}_i, \mathbf{w}_k), \quad \forall i \in \Omega; j \in \eta_i^1; k \notin \eta_i^1 \& k \in \eta_i^2; l \notin \eta_i^2 \tag{24}$$

and so forth to define higher ordered maps with interneuron distance hierarchies [117].

An m th order map is optimal for tolerating channel noise spreading up to the m th neighbouring node. Such fully ordered maps however may not be always achievable, especially when the mapping is a dimensional reduction one. Then the degree (percentage) of nodes with their weights being ordered can be measured, together with the probabilities of the nodes being utilized, can be used to determine the topology preservation and that to what degree and to what order the map can tolerate the channel noise.

[25] proposed the C measure – a correlation between the similarity of stimuli in the data space and the similarity of their prototypes in the map space – to quantify the topological preservation,

$$C = \sum_i \sum_j F(i, j) G[M(i), M(j)] \tag{25}$$

where F and G are symmetric similarity measures in the input and map spaces respectively, and can be problem specific, and $M(i)$ and $M(j)$ are the mapped points or weight vectors of node i and j , respectively.

The C measure directly evaluates the correlation between distance relations between two spaces. Various other topographic mapping objectives can be unified under the C measure, such as multidimensional scaling, minimal wiring, the travelling salesperson problem (TSP), and noise tolerant VQ. It has also been shown that if a mapping that preserves ordering exists then maximizing C will find it. Thus the C measure is also the objective function of the mapping, an important property different from other topology preservation measures and definitions.

One can always use the underlying cost function Eqn. (18) to measure the goodness of the resulting map including the topology preservation, at least one can use a temporal window to take a sample of it as suggested in [50]. The (final) neighbourhood function specifies the level of topology (ordering) the mapping is likely to achieve or is required. To draw an analogy to the above C measure, the neighbourhood function can be interpreted as the G measure used in Eqn. (25) and the $\|\mathbf{x} - \mathbf{w}_k\|^2$ term represents the F measure. Indeed, the input x and weight w_j are mapped on the map as node index i and j , and their G measure is the neighbourhood function (for example, exponentials). Such an analogy also sheds light on the scaling effect of the SOM. Multidimensional scaling also aims to preserve local similarities on a mapped space (see the next Section for more details).

4 Extensions and Links with Other Learning Paradigms

The SOM has been a popular model for unsupervised learning as well as pattern organization and association. Since its introduction, various extensions have been reported to enhance its performance and scope. For instance, ‘Neural Gas’ was developed to map data onto arbitrary or hierarchical map structures rather than confined to rectangular grids for improved VQ performance [74, 75]. The adaptive subspace SOM (ASSOM) has been proposed to combine principal component learning and the SOM to map data with reduced feature space, in order to form translation-, rotation- and scale-invariant filters [51, 52]. The parameterized SOM (PSOM) has been proposed to extend SOM for continuous mapping using basis functions on the grid to interpolate the map [112]. The stochastic SOM [26] defines a topographic mapping from a Bayesian framework and a Markov chain encoder, and further explains the stochastic annealing effect in SOM. The Dislex [76, 77] applies hierarchical topographical maps to associate cross-modal patterns such as images with audio or symbols. The U-matrix was proposed to imprint the distance information on the map for visualization [105]. The visualization induce SOM (ViSOM) has been proposed to directly preserve distance information on the map so as to visualize data structures and distributions [118, 119].

The Temporal Kohonen map [11] and its improved version, the Recurrent SOM [108], as well as the Recursive SOM [110] have extended the SOM for

mapping temporal data such as time series, or sequential data such as protein sequences. Extensions along this direction continue to be a focus of research. Extension on probabilistic approaches which enhances the scope and capability of SOM include the Self-Organizing Mixture Network (SOMN) [128], Kernel-based topographic maps [106, 107]; and the generic topographic mapping (GTM) [8]. There are many extensions developed in recent years – too many to completely list here. Recent extensions are also proposed for handling non-vectorial [55] and qualitative data [35]. For more comprehensive lists and recent developments, please refer to [1, 13, 38, 40, 53, 83].

The remainder of this Section covers extensions of SOM and their associations with data visualization, manifold mapping, density modeling and kernel methods.

4.1 SOM, Multidimensional Scaling and Principal Manifolds

The SOM is often associated with VQ and clustering. However it is also associated with data visualization, dimensionality reduction, nonlinear data projection, and manifold mapping. A brief review on various data projection methods and their relationships has been given before [121].

Multidimensional Scaling

Multidimensional scaling (MDS) is a traditional study related to dimensionality reduction and data projection. MDS tries to project data points onto an (often two-dimensional) sheet by preserving as closely as possible the inter-point metrics [14]. The projection is generally nonlinear and can reveal the overall structure of the data. A general fitness function or the so-called stress function is defined as,

$$S = \frac{\sum_{i,j} (d_{ij} - D_{ij})^2}{\sum_{i,j} D_{ij}^2} \quad (26)$$

where d_{ij} represents the proximity of data points i and j in the original data space, and D_{ij} represents the dissimilarity or distance (usually Euclidean) between mapped points i and j in the projected space. Note, that global Euclidean distance is usually used to calculate the inter-point distances. Recently, Isomap was proposed to use geodesic (curvature) distance instead for better nonlinear scaling [102].

MDS relies on an optimization algorithm to search for a configuration that gives as low a stress as possible. A gradient method is commonly used for this purpose. Inevitably, various computational problems – such as local minima and divergence – may occur due to the optimization process itself. The methods are also often computationally intensive. The final solution depends on the starting configuration and parameters used in the algorithm.

Sammon mapping is a widely-known example of MDS [95]. The objective is to minimize the differences between inter-point (Euclidean) distances in the original space and those in the projected plane. In Sammon mapping intermediate normalization (of the original space) is used to preserve good local distributions and at the same time maintain a global structure. The Sammon stress is expressed as,

$$S_{Sammon} = \frac{1}{\sum_{i<j} d_{ij}} \sum_{i<j} \frac{(d_{ij} - D_{ij})^2}{d_{ij}} \quad (27)$$

A second order Newton optimization method is used to recursively solve the optimal configuration. It converges faster than the simple gradient method, but the computational complexity is even higher. It still has local minima and inconsistency problems. Sammon mapping has been shown to be useful for data structure analysis. However, like other MDS methods, the Sammon algorithm is a point-to-point mapping, which does not provide an explicit mapping function and cannot naturally accommodate new data points. It also requires the computation and storage of all the inter-point distances. This proves difficult or even impossible for many practical applications where data arrives sequentially, the quantity of data is large, and/or memory space for the data is limited.

In addition to being computationally expensive, especially for large data sets, and not being adaptive, another major drawback of MDS is lack of an explicit projection function. Thus for any new input data, the mapping has to be recalculated based on all available data. Although some methods have been proposed to accommodate the new arrivals using triangulation [16, 61], the methods are generally not adaptive. However, such drawbacks can be overcome by implementing or parameterizing MDS using neural networks – for example, [65, 71].

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a classic linear projection method aiming at finding orthogonal principal directions from a set of data, along which the data exhibiting the largest variances. By discarding the minor components, PCA can effectively reduce data variables and display the dominant ones in a linear, low dimensional subspace. It is an optimal linear projection in the sense of the mean-square error between original points and projected ones, in other words,

$$\min_{\mathbf{x}} \left[\mathbf{x} - \sum_{j=1}^m (\mathbf{q}_j^T \mathbf{x}) \mathbf{q}_j \right]^2 \quad (28)$$

where $\{\mathbf{q}, j = 1, 2, \dots, m, m \leq n\}$ are orthogonal eigenvectors representing principal directions. They are the first m principal eigenvectors of the

covariance matrix of the input. The second term in the above bracket is the reconstruction or projection of x onto these eigenvectors. The term $\mathbf{q}_j^T \mathbf{x}$ represents the projection of x onto the j th principal dimension.

Traditional methods for solving the eigenvector problem involve numerical methods. Though fairly efficient and robust, they are not usually adaptive and often require presentation of the entire data set. Several Hebbian-based learning algorithms and neural networks have been proposed for performing PCA, such as the subspace network [81] and the generalized Hebbian algorithm [96]. The limitation of linear PCA is obvious, as it cannot capture nonlinear relationships defined by higher than second-order statistics. If the input dimension is much higher than two, the projection onto the linear principal plane will provide limited visualization power.

Nonlinear PCA and Principal Manifolds

Extension to nonlinear PCA (NLPCA) is not unique, due to the lack of a unified mathematical structure and an efficient and reliable algorithm, and in some cases due to excessive freedom in selection of representative basis functions [39, 70]. Several methods have been proposed for nonlinear PCA, such as the five-layer feedforward associative network [56] and the kernel PCA [97]. The first three layers of the associative network project the original data onto a curve or surface, providing an activation value for the bottleneck node. The last three layers define the curve and surface. The weights of the associative NLPCA network are determined by minimizing the following objective function,

$$\min_{\mathbf{x}} \sum_{\mathbf{x}} \|\mathbf{x} - \mathbf{f}\{s_f(\mathbf{x})\}\|^2 \quad (29)$$

where $\mathbf{f} : R^1 \rightarrow R^n$ (or $R^2 \rightarrow R^n$). The function modelled by the last three layers defines a curve (or a surface), $s_f : R^n \rightarrow R^1$ (or $R^n \rightarrow R^2$); the function modelled by the first three layers defines the projection index.

The kernel-based PCA uses nonlinear mapping and kernel functions to generalize PCA to NLPCA and has been used for various pattern recognition tasks. The nonlinear function $\Phi(x)$ maps data onto high-dimensional feature space, where the standard linear PCA can be performed via kernel functions: $k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$. The projected covariance matrix is then,

$$Cov = \frac{1}{N} \sum_{i=1}^N \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^T \quad (30)$$

The standard linear eigenvalue problem can now be written as $\lambda \mathbf{V} = \mathbf{K} \mathbf{V}$, where the columns of \mathbf{V} are the eigenvectors, and \mathbf{K} is a $N \times N$ matrix with elements as kernels $k_{ij} := k(\mathbf{x}_i, \mathbf{x}_j) = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$.

The principal curves and principal surfaces [28, 60] are the principal nonlinear extensions of PCA. The principal curve is defined as a smooth and

self-consistency curve, which does not intersect itself. Denote \mathbf{x} as a random vector in R^n with density p and finite second moment. Let $f(\cdot)$ be a smooth unit-speed curve in R^n , parameterized by the arc length (from one end of the curve) over $A \in R$, a closed interval.

For a data point \mathbf{x} its projection index on f is defined as

$$\rho_f(\mathbf{x}) = \sup_{\rho \in A} \{ \rho : \|\mathbf{x} - f(\rho)\| = \inf_{\theta} \|\mathbf{x} - f(\theta)\| \} \quad (31)$$

The curve is called self-consistent or a principal curve of ρ if

$$f(\rho) = E[\mathbf{X} \mid \rho_f(\mathbf{X}) = \rho] \quad (32)$$

The principal component is a special case of the principal curves if the distribution is ellipsoidal. Although principal curves have been mainly studied, extension to higher dimensions – for example principal surfaces or manifolds – is feasible in principle. However, in practice, a good implementation of principal curves/surfaces relies on an effective and efficient algorithm. The principal curves/surfaces are more of a concept that invites practical implementations. The HS algorithm is a nonparametric method [28], which directly iterates the two steps of the above definition. It is similar to the standard LGB VQ algorithm [63], combined with some smoothing techniques.

Algorithm 2 The Hastie and Stuetzle (HS) algorithm

Initialization: choose the first linear principal component as the initial curve, $f^{(0)}(\mathbf{x})$.

repeat

Projection: project the data points onto the current curve and calculate the projections index – that is $\rho^{(t)}(x) = \rho_{f^{(t)}}(\mathbf{x})$.

Expectation: for each index, take the mean of the data points projected onto it as the new curve point – in other words, $f^{t+1}(\rho) = E[\mathbf{X} \mid \rho_{f^{(t)}}\mathbf{X} = \rho]$.

until a convergence criterion is met (for example, when the change of the curve between iterations falls below a threshold).

For a finite data set, the density p is often unknown, and the above expectation is replaced by a smoothing method such as the locally weighted running-line smoother or smoothing splines. For kernel regression, the smoother is,

$$f(\rho) = \frac{\sum_{i=1}^N \mathbf{x}_i \mathcal{K}(\rho, \rho_i)}{\sum_{i=1}^N \mathcal{K}(\rho, \rho_i)} \quad (33)$$

The arc length is simply computed from the line segments. There are no proofs of convergence for the algorithm, but no convergence problems have been reported, although the algorithm is biased in some cases [28]. Banfield and Raftery have modified the HS algorithm by taking the expectation of the

residual of the projections in order to reduce the bias [5]. [42] have proposed an incremental – for example, segment-by-segment – and arc length constrained method for practical construction of principal curves.

Tibshirani has introduced a semi-parametric model for the principal curve [103]. A mixture model was used to estimate the noise along the curve; and the expectation-maximization (EM) method was employed to estimate the parameters. Other options for finding the nonlinear manifold include the Generic Topographic Map [8] and probabilistic principal surfaces [10]. These methods model the data by a means of a latent space. They belong to the semi-parameterized mixture model, although types and orientations of the local distributions vary from method to method.

Visualization induced SOM (ViSOM)

For scaling and data visualization, a direct and faithful display of data structure and distribution is highly desirable. ViSOM has been proposed to extend the SOM for direct distance preservation on the map [118, 119], instead of using a colouring scheme such as U-matrix [105], which imprints qualitatively the inter-neuron distances as colours or grey levels on the map. For the map to capture the data structure naturally and directly, (local) distance quantities must be preserved on the map, along with the topology. The map can be seen as a smooth and graded mesh, or manifold embedded into the data space onto which the data points are mapped and the inter-point distances are approximately preserved.

In order to achieve that, the updating force, $\mathbf{x}(t) - \mathbf{w}_k(t)$, of the SOM algorithm is decomposed into two elements $[\mathbf{x}(t) - \mathbf{w}_\nu(t)] + [\mathbf{w}_\nu(t) - \mathbf{w}_k(t)]$. The first term represents the updating force from the winner ν to the input $\mathbf{x}(t)$, and is the same to the updating force used by the winner. The second force is a lateral contraction force bringing neighbouring neuron k to the winner ν . In the ViSOM, this lateral contraction force is constrained or regulated in order to help maintain unified local inter-neuron distances $\|\mathbf{w}_\nu(t) - \mathbf{w}_k(t)\|$ on the map.

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha(t)\eta(v, k, t)[\mathbf{x}(t) - \mathbf{w}_\nu(t)] + \beta[\mathbf{w}_\nu(t) - \mathbf{w}_k(t)] \quad (34)$$

where the simplest constraint can be $\beta = \frac{d_{\nu k}}{(D_{\nu k}\lambda)^{-1}}$, with $d_{\nu k}$ being the distance of neuron weights in the input space, $D_{\nu k}$ the distance of neuron indexes on the map, and λ a (required) resolution constant.

ViSOM regularizes the contraction force so that the distances between nodes on the map are analogous to the distances of their weights in the data space. The aim is to adjust inter-neuron distances on the map in proportion to those in the data space, in other words $D_{\nu k} \propto d_{\nu k}$. When the data points are eventually projected onto a trained map, the distance between point i and j on the map is proportional to that of the original space, subject to the

quantization error (the distance between a data point and its neural representative). This has a similar effect to MDS, which also aims at achieving this proportionality, $D_{ij} \propto d_{ij}$.

The SOM is shown to be a qualitative scaling, while the ViSOM is a metric scaling [124]. The key feature of ViSOM is that the distances between the neurons (which data are mapped to) on the map (in a neighbourhood) reflect the corresponding distances in the data space. When the map is trained and data points mapped, the distances between mapped data points will resemble approximately those in the original space (subject to the resolution of the map). This makes visualization more direct, quantitatively measurable, and visually appealing. The map resolution can be enhanced (and the computational cost reduced) by interpolating a trained map or incorporating local linear projections [122]. The size or covering range of the neighbourhood function can also be decreased from an initially large value to a final smaller one. The final neighbourhood, however, should not contain just the winner. The rigidity or curvature of the map is controlled by the ultimate size of the neighbourhood. The larger this size, the flatter the final map is in the data space. Guidelines for setting these parameters have been given in [120]. An example on data visualization will be shown in the next Section.

Several authors have since introduced improvements and extensions to ViSOM. For example, in [116], a probabilistic data assignment [26] is used in both the input assignment and the neighbourhood function; also an improved second order constraint is adopted. The resulting SOM has a clearer connection to an MDS cost function. Estévez and Figueora extend the ViSOM to an arbitrary, neural gas type of map structure [21]. Various other variants of SOM, such as hierarchical, growing, and hierarchical and growing structures are readily extendable to the ViSOM for various application needs.

The SOM has been related to the discrete principal curve/surface algorithm [91]. However differences remain in both the projection and smoothing processes. In the SOM the data are projected onto the nodes rather than onto the curve. The principal curves perform the smoothing entirely in the data space – see Eqn. (33). The smoothing process in SOM and ViSOM, as a convergence criterion, is [120],

$$\mathbf{w}_k = \frac{\sum_{i=1}^L \mathbf{x}_i \eta(\nu, k, i)}{\sum_{i=1}^L \eta(\nu, k, i)} \quad (35)$$

Smoothing is governed by the indexes of the neurons in the map space. The kernel regression uses the arc length parameters (ρ, ρ_i) or $\|\rho - \rho_i\|$ exactly, while the neighbourhood function uses the node indexes (ν, k) or $\|\mathbf{r}_\nu - \mathbf{r}_k\|$. Arc lengths reflect the curve distances between the data points. However, node indexes are integer numbers denoting the nodes or positions on the map grid, not the positions in the input space. So $\|\mathbf{r}_\nu - \mathbf{r}_k\|$ does

not resemble $\| \mathbf{w}_\nu - \mathbf{w}_k \|$ in the common SOM. In the ViSOM, however, as the local inter-neuron distances on the map represent those in the data space (subject to the map resolution), the distances of nodes on the map are in proportion to the difference of their positions in the data space, that is $\| \mathbf{r}_\nu - \mathbf{r}_k \| \sim \| \mathbf{w}_\nu - \mathbf{w}_k \|$. The smoothing process in the ViSOM resembles that of the principal curves as shown below,

$$\mathbf{w}_k = \frac{\sum_{i=1}^L \mathbf{x}_i \eta(\nu, k, i)}{\sum_{i=1}^L \eta(\nu, k, i)} \approx \frac{\sum_{i=1}^L \mathbf{x}_i \eta(\mathbf{w}_\nu, \mathbf{w}_k, i)}{\sum_{i=1}^L \eta(\mathbf{w}_\nu, \mathbf{w}_k, i)} \tag{36}$$

This shows that ViSOM is a better approximation to the principal curves/surfaces than the SOM. SOM and ViSOM are similar only when the data are uniformly distributed, or when the number of nodes becomes very large, in which case both SOM and ViSOM will closely approximate the principal curves/surfaces.

4.2 SOM and Mixture Models

The SOM has been linked with density matching models and the point density that the SOM produces is related to the density of the data. However the SOM does not exactly follow the data density. Such properties have been studied and treated under the VQ framework [17, 54, 67, 89, 117].

The self-organizing mixture network (SOMN) [128] extends and adapts the SOM to a mixture density model, in which each node characterizes a conditional probability distribution. The joint probability density of the data or the network output is described by a mixture distribution,

$$p(\mathbf{x} | \Theta) = \sum_{i=1}^K p_i(\mathbf{x} | \theta_i) P_i \tag{37}$$

where $p_i(\mathbf{x} | \theta_i)$ is the i th component-conditional density, and θ_i is the parameter for the i th conditional density, $i = 1, 2, \dots, K$; $\Theta = (\theta_1, \theta_2, \dots, \theta_K)^T$, and P_i is the prior probability of the i th component or node and is also called the mixing weights. For example, a Gaussian mixture has the following the conditional densities respectively,

$$p_i(\mathbf{x} | \theta_i) = \frac{1}{(2\pi)^{d/2} |\sum_i|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \mathbf{m}_i)^T \sum_i^{-1} (\mathbf{x} - \mathbf{m}_i) \right] \tag{38}$$

where $\theta_i = \{ \mathbf{m}_i, \sum_i \}$ are the mean vector and covariance matrix, respectively.

Suppose that the true environmental data density function and the estimated one are $p(\mathbf{x})$ and $\hat{p}(\mathbf{x})$, respectively. The Kullback-Leibler information distance measures the divergence between these two, and is defined as,

$$\mathbf{I} = - \int \log \frac{\hat{p}(\mathbf{x})}{p(\mathbf{x})} p(\mathbf{x}) d\mathbf{x} \quad (39)$$

It is always positive and is equal to zero only when two densities are identical.

When the estimated density is modelled as a mixture distribution, one can seek the optimal estimate of the parameters by minimizing the Kullback-Leibler divergence via its partial differentials in respect to model parameters, more specifically,

$$\frac{\partial \mathbf{I}}{\partial \theta_i} = - \int \left[\frac{1}{\hat{p}(\mathbf{x} | \hat{\Theta})} \frac{\partial \hat{p}(\mathbf{x} | \hat{\Theta})}{\partial \theta_i} \right] p(\mathbf{x}) d\mathbf{x}, \quad i = 1, 2, \dots, K \quad (40)$$

As the true data density is not known, the stochastic gradient is used for solving these non-directly solvable equations. This results in the following adaptive update rules for the parameters and priors [129],²

$$\begin{aligned} \hat{\theta}_i(t+1) &= \hat{\theta}_i(t) + \alpha(t) \eta(\nu(\mathbf{x}), i) \left[\frac{1}{\hat{p}(\mathbf{x} | \hat{\Theta})} \frac{\partial \hat{p}(\mathbf{x} | \hat{\Theta})}{\partial \theta_i} \right] \\ &= \hat{\theta}_i(t) + \alpha(t) \eta(\nu(\mathbf{x}), i) \left[\frac{\hat{P}_i(t)}{\sum_j \hat{P}_i(t) \hat{p}_j(\mathbf{x} | \hat{\theta}_j)} \frac{\partial \hat{p}_i(\mathbf{x} | \hat{\theta}_i)}{\partial \theta_i} \right] \end{aligned} \quad (41)$$

and

$$\begin{aligned} \hat{P}_i(t+1) &= \hat{P}_i(t) + \alpha(t) \left[\frac{\hat{p}_i(\mathbf{x} | \hat{\theta}_i) \hat{P}_i(t)}{\hat{p}(\mathbf{x} | \hat{\Theta})} - \hat{P}_i(t) \right] \\ &= \hat{P}_i(t) - \alpha(t) \eta(\nu(\mathbf{x}), i) \left[\hat{P}(i | \mathbf{x}) - \hat{P}_i(t) \right] \end{aligned} \quad (42)$$

where $\alpha(t)$ is the learning coefficient or rate at time step t ($0 < \alpha(t) < 1$), and decreases monotonically. The winner is found via the maximum *posterior* probability of the node,

$$\hat{P}(i | \mathbf{x}) = \frac{\hat{P}_i \hat{p}_i(\mathbf{x} | \hat{\theta}_i)}{\sum_j \hat{p}_j(\mathbf{x} | \hat{\Theta})} \quad (43)$$

When the SOMN is limited to the homoscedastic case – namely equal variances and equal priors (non-informative priors) for all components – only the means are the learning variables. The above winner rule becomes,

$$v = \arg \max_i \frac{\hat{p}_i(\mathbf{x} | \hat{\theta}_i)}{\sum_j \hat{p}_j(\mathbf{x} | \hat{\theta}_j)} \quad (44)$$

² A common neighbourhood function, $\eta(\nu(\mathbf{x}), i)$, can be added as in the SOM, but is optional.

When the conditional density function is isotropic or symmetric or is a function of $\|\mathbf{x} - \mathbf{m}\|$, the above winning rule is a function of commonly used Euclidean norm $\|\mathbf{x} - \mathbf{m}\|$. The corresponding weight updating rule is,

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \alpha(t)\eta(\nu(\mathbf{x}), i) \frac{1}{\sum_j p_j(\mathbf{x} | \theta_j)} \frac{\partial p_i(\mathbf{x} | \theta_i)}{\partial \mathbf{m}_i} \quad (45)$$

For example, for a Gaussian mixture with equal variance and prior for all nodes, it is easy to show that the winning and mean update rules become,

$$v = \arg \max_i \left[\exp \left(-\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2\sigma^2} \right) \right] \quad (46)$$

and

$$\begin{aligned} & \mathbf{m}_i(t+1) \\ &= \mathbf{m}_i(t) + \alpha(t)\eta(\nu(\mathbf{x}), i) \frac{1}{2\sigma^2} \frac{1}{\sum_j p_j(\mathbf{x} | \theta_j)} \exp \left(-\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2\sigma^2} \right) (\mathbf{x} - \mathbf{m}_i) \end{aligned} \quad (47)$$

The winning rule becomes equivalent to the simple distance measure. The update formula bear a similarity to that of the original SOM. The term $\exp \left(-\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2\sigma^2} \right)$ is playing a similar role as the neighbourhood function, defined by the distances between weights and input instead of node indexes. The SOM approximates it by quantizing it using node indexes. The SOMN is also termed a ‘Bayesian SOM’, as it applies the Bayesian learning principle to the SOM learning rule [129].

4.3 SOM and Kernel Method

A kernel is a function $\mathcal{K} : X \times X \rightarrow \mathcal{R}$, where X is the input space. This function is a dot product of the mapping function $\phi(\mathbf{x})$ – in other words $\mathcal{K}(\mathbf{x}; \mathbf{y}) = [\phi(\mathbf{x}), \phi(\mathbf{y})]$, where $\phi : X \rightarrow F, F$ being a high dimensional inner product feature space. The mapping function $\phi(\mathbf{x})$ is often nonlinear and not known. All the operations are defined in terms of the kernel function instead of the mapping function. The kernel methodology has become increasingly popular within supervised learning paradigms, with the Support Vector Machine (SVM) being a widely known example. When nonlinearly mapping data or patterns to high dimensional space, the patterns often become linearly separable.

The kernel method has also been applied to the SOM. Following the kernel PCA [97], a k -means based kernel SOM has been proposed [69]. Each data point x is mapped to the feature space via a (unknown or imaginary) nonlinear function $\phi(\mathbf{x})$. In principle each mean can be described as a weighted sum of the observations in the feature space $\mathbf{m}_i = \sum_n \gamma_{i,n} \phi(\mathbf{x}_n)$, where $\{\gamma_{i,n}\}$ are

the constructing coefficients. The algorithm then selects a mean or assigns a data point with the minimum distance between the mapped point and the mean,

$$\begin{aligned} \|\phi(\mathbf{x} - \mathbf{m}_i)\|^2 &= \|\phi(\mathbf{x} - \sum_n \gamma_{i,n} \phi(\mathbf{x}_n))\|^2 \\ &= \mathcal{K}(\mathbf{x}, \mathbf{x}) - 2 \sum_n \gamma_{i,n} \mathcal{K}(\mathbf{x}, \mathbf{x}_n) + \sum_{n,m} \gamma_{m,n} \mathcal{K}(\mathbf{x}_n, \mathbf{x}_m) \end{aligned} \quad (48)$$

The update of the mean is based on a soft learning algorithm,

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \Lambda[\phi(\mathbf{x} - \mathbf{m}_i(t))] \quad (49)$$

where Λ is the normalized winning frequency of the i th mean and is defined as,

$$\Lambda = \frac{\zeta_{i(\mathbf{x}),j}}{\sum_{n=1}^{t+1} \zeta_{i,n}} \quad (50)$$

where ζ is the winning counter and is often defined as a Gaussian function between the indexes of the two neurons.

As the mapping function $\phi(\mathbf{x})$ is not known, the update rule (Eqn.(49)) is further elaborated and leads to the following updating rules for the constructing coefficients of the means [69],

$$\gamma_{i,n}(t+1) = \begin{cases} \gamma_{i,n}(t)(1 - \zeta), & \text{for } n \neq t+1 \\ \zeta, & \text{for } n = t+1 \end{cases} \quad (51)$$

Note that these constructing coefficients, $\{\gamma_{i,n}\}$, together with the kernel function, effectively define the kernel SOM in feature space. The winner selection – that is, Eqn. (48) – operates on these coefficients and the kernel function. No *explicit* mapping function $\phi(\mathbf{x})$ is required. The exact means or neurons' weights – $\{\mathbf{m}_i\}$ – are not required.

There is another, direct way to kernelize the SOM by mapping the data points and neuron weights, both defined in the input space, to a feature space, then applying the SOM in the mapped dot product space. The winning rules of this second type of kernel SOM have been proposed as follows, either in the input space [85],

$$v = \arg \min_i \|\mathbf{x} - \mathbf{m}_i\| \quad (52)$$

or in the feature space [4],

$$v = \arg \min_i \|\phi(\mathbf{x}) - \phi(\mathbf{m}_i)\| \quad (53)$$

It will soon become clear that these two rules are equivalent for certain kernels, such as the Gaussian. The weight update rule proposed by [4] is,

$$\mathbf{m}_i(t + 1) = \mathbf{m}_i(t) + \alpha(t)\eta(v(\mathbf{x}), i)\Delta J(\mathbf{x}, \mathbf{m}_i) \tag{54}$$

where $J(\mathbf{x}, \mathbf{m}_i) = \|\phi(\mathbf{x}) - \phi(\mathbf{m}_i)\|^2$ is the distance function in the feature space or the proposed instantaneous or sample objective function. and are the learning rate and neighbourhood function, respectively.

Note that,

$$J(\mathbf{x}, \mathbf{m}_i) = \|\phi(\mathbf{x}) - \phi(\mathbf{m}_i)\|^2 = \mathcal{K}(\mathbf{x}, \mathbf{x}) + \mathcal{K}(\mathbf{m}_i, \mathbf{m}_i) - 2\mathcal{K}(\mathbf{x}, \mathbf{m}_i) \tag{55}$$

and,

$$\nabla J(\mathbf{x}, \mathbf{m}_i) = \frac{\partial \mathcal{K}(\mathbf{m}_i, \mathbf{m}_i)}{\partial \mathbf{m}_i} - 2\frac{\partial \mathcal{K}(\mathbf{x}, \mathbf{m}_i)}{\partial \mathbf{m}_i} \tag{56}$$

Therefore this kernel SOM can also be operated entirely in the feature space with the kernel function. As the weights of the neurons are defined in the input space, they can be explicitly resolved.

These two kernel SOMs have been proved equivalent ([59], [123]); they can all be derived from applying the energy function (Eqn. (18)) on the mapped feature space,

$$E_F = \sum_i \int_{v_i} \sum_j \eta(i, j) \|\phi(\mathbf{x}) - \phi(\mathbf{m}_j)\|^2 p(\mathbf{x}) d\mathbf{x} \tag{57}$$

The kernel SOM can be seen as a result of directly minimizing this transformed energy stochastically, in other words, by using the sample gradient on $\sum_j \eta(v(\mathbf{x}), j) \|\phi(\mathbf{x}) - \phi(\mathbf{m}_j)\|^2$,

$$\frac{\partial \hat{E}_F}{\partial \mathbf{m}_i} = \frac{\partial}{\partial \mathbf{m}_i} \sum_j \eta(v(\mathbf{x}), j) \|\phi(\mathbf{x}) - \phi(\mathbf{m}_j)\|^2 = -2\eta(v(\mathbf{x}), i)\nabla J(\mathbf{x}, \mathbf{m}_i) \tag{58}$$

This leads to the same weight update rule of the kernel SOM as Eqn. (54).

Various kernel functions such as Gaussian (or radial basis function), Cauchy and polynomial, are readily applicable to the kernel SOM [59]. For example, for Gaussian kernel, the winning and weight update rules are,

$$\begin{aligned} v &= \arg \min_i J(\mathbf{x}, \mathbf{m}_i) = \arg \min_i [-2\mathcal{K}(\mathbf{x} - \mathbf{m}_i)] \\ &= \arg \min_i \left[-\exp\left(-\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2\sigma^2}\right) \right] \end{aligned} \tag{59}$$

and,

$$\mathbf{m}_i(t + 1) = \mathbf{m}_i(t) + \alpha(t)\eta(v(\mathbf{x}), i)\frac{1}{2\sigma^2}\exp\left(-\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2\sigma^2}\right)(\mathbf{x} - \mathbf{m}_i) \tag{60}$$

respectively. Please note for Gaussian kernel functions, although the winning rule (Eqn. (59)) is derived from the feature space, it is equivalent to that of the original SOM and is conducted in the input space.

Comparing the kernel SOM algorithm (Eqns. (59) and (60)) with those of the SOMN (Eqns. (46) and (47)), it can be easily seen that the two methods are the same [123]. That is, the kernel SOM (with Gaussian kernels) is implicitly applying a Gaussian mixture to model the data. In other words, the SOMN is a kind of kernel method. As the SOM is seen as a special case of the SOMN, the original SOM has a certain effect of the kernel method.

5 Applications and Case Studies

Thousands of applications of the SOM and its variants have been reported since its introduction [40, 53, 83] – too many to list here. There is a dedicated international Workshop on SOMs (WSOM), as well as focused sessions in many neural network conferences. There have also been several special journal issues dedicated to advances in SOM and related topics [1, 13, 38]. Moreover, many new applications are being reported in many relevant journals today. SOMs will remain an active topic in their continued extension, combination and applications in the years to come.

In this Section, several typical applications are provided as case studies. They include image and video processing; density or spectrum profile modeling; text/document mining and management systems; gene expression data analysis and discovery; and high dimensional data visualizations. Other typical applications not discussed here include image/video retrieval systems – for instance, PicSOM [57]; nonlinear ICA (Nonlinear PCA and ICA) [27, 31, 37, 82, 84]; classification (LVQ) [53]; cross-modal information processing and associations [76, 77]; novelty detection [73]; robotics [6]; hardware implementation [98]; and computer animation [113].

5.1 Vector Quantization and Image Compression

The SOM is an optimal VQ when the neighbourhood eventually shrinks to just the winner, as it will satisfy the two necessary conditions for VQ (Voronoi partition and centroid condition). The use of the neighbourhood function makes the SOM superior to common VQs in two main respects. Firstly, the SOM is better at overcoming the under- or over-utilization and local minima problem. The second is that the SOM will produce a map (codebook) with some ordering (even when the neighbourhood eventually vanishes) among the code vectors, and this gives the map an ability to tolerate noise in the input or retrieval patterns. An example is provided in Fig. 6, in which (b) shows the 16×16 codebook trained on the Lena test image of 512×512 pixels by SOM with distinctive ordering found among the code vectors; and (a) shows the quantized Lena image by the trained codebook. The code vectors are of 4×4 pixel blocks.

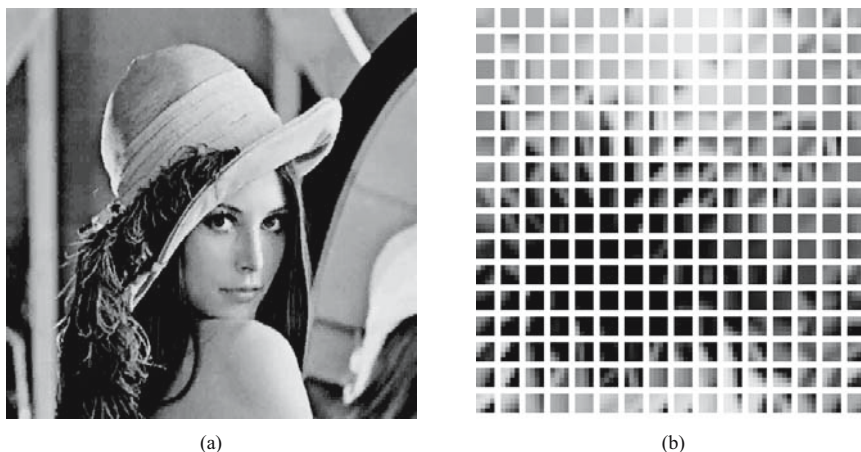


Fig. 6. (a) Quantized Lena image; (b) the SOM codebook (map)

It has been found that SOMs generally perform better than other VQs especially in situations where local optima are present [117]. The robustness of SOM has been further improved by introducing a constraint on the learning extent of a neuron based on the input space variance it covers. The algorithm is aiming to achieve global optimal VQ by limiting and unifying the distortions from all nodes to approximately equal amounts – the asymptotic property of the global optimal VQ (in other words, for a smooth underlying probability density and large number of code vectors as all regions in an optimal Voronoi partition have the same within region variance). The constraint is applied to the scope of the neighbourhood function so that the node covering a large region (thus having a large variance) has a large neighbourhood. The results show that the resulting quantization error is smaller. Such a SOM-based VQ has also been applied to video compression [2, 22] for improved performance at low bit rates.

5.2 Image Segmentation

The SOM has been used in a hierarchical structure, together with the Markov random field (MRF) model, for the unsupervised segmentation of textured images [125]. The MRF is used as a measure of homogeneous texture features from a randomly placed local region window on the image. Such features are noisy and poorly known. They are input to a first SOM layer, which learns to classify and filter them. The second local-voting layer – a simplified SOM – produces an estimate of the texture type or label for the region. The hierarchical network learns to progressively estimate the texture model, and classify the various textured regions of similar type. Randomly positioning the local window at each iteration ensures that the consecutive inputs to the SOMs are uncorrelated. The size of the window is large at the beginning to capture patch-like texture homogeneities and shrinks with time to reduce the

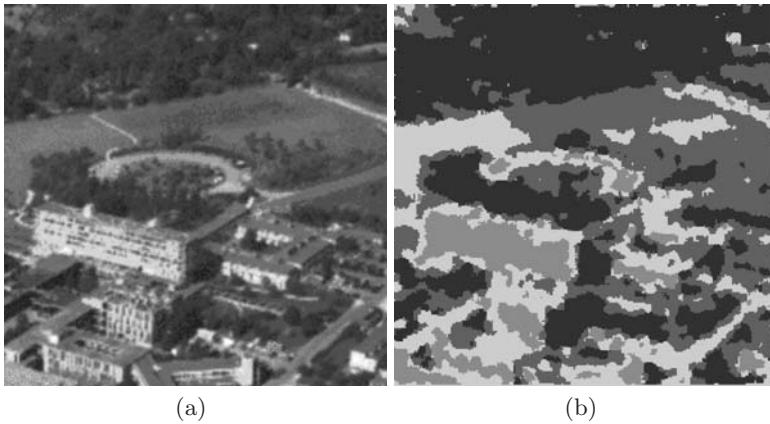


Fig. 7. (a) An aerial image; (b) segmented image using the SOM and Markov random field

estimation parameter noise at texture boundaries. The weights of the neurons in the first layer will converge to the MRF model parameters of various texture types, whilst the weights of the second layer will be the prototypes of these types – that is, the segmented image. The computational form of the entire algorithm is simple and efficient. The theoretical analysis of the algorithm shows that it will converge to the maximum likelihood segmentation. Figure 7 shows a typical example of such applications. The number of texture types was subjectively assumed as four. Interestingly, the algorithm has segmented the image into four meaningful categories: ‘trees’, ‘grass’, ‘buildings’, and ‘roads’.

5.3 Density Modeling

Some function profiles (such as spectra) can be considered as density histograms. If a spectrum consists of many components, then the SOMN described in Sect. 4.2 can be used to estimate the component profiles of the spectrum [128, 129]. Re-sampling the observed spectrum will provide distribution data for training. The x-ray diffraction patterns of crystalline complex organic molecules (such as proteins) consist of a large number of Bragg diffraction spots. These patterns represent the intensity Fourier transform of the molecular structure (actually the electron density maps); the crystallographers need to determine the precise position of each spot together with its magnitude (namely, integrated spot intensity). The patterns exhibit relatively high background noise together with spot spreading (due to shortcomings in the experiments or limitations in the detection processes), which results in overlapping spots. Automatic analysis of these patterns is a non-trivial task. An example of such a pattern image is shown in Fig. 8(a), which is an 8-bit greyscale and of size 88×71 pixels. In order for the SOMN to learn the profiles of these diffraction spots, the image (diffraction intensity function) has to be

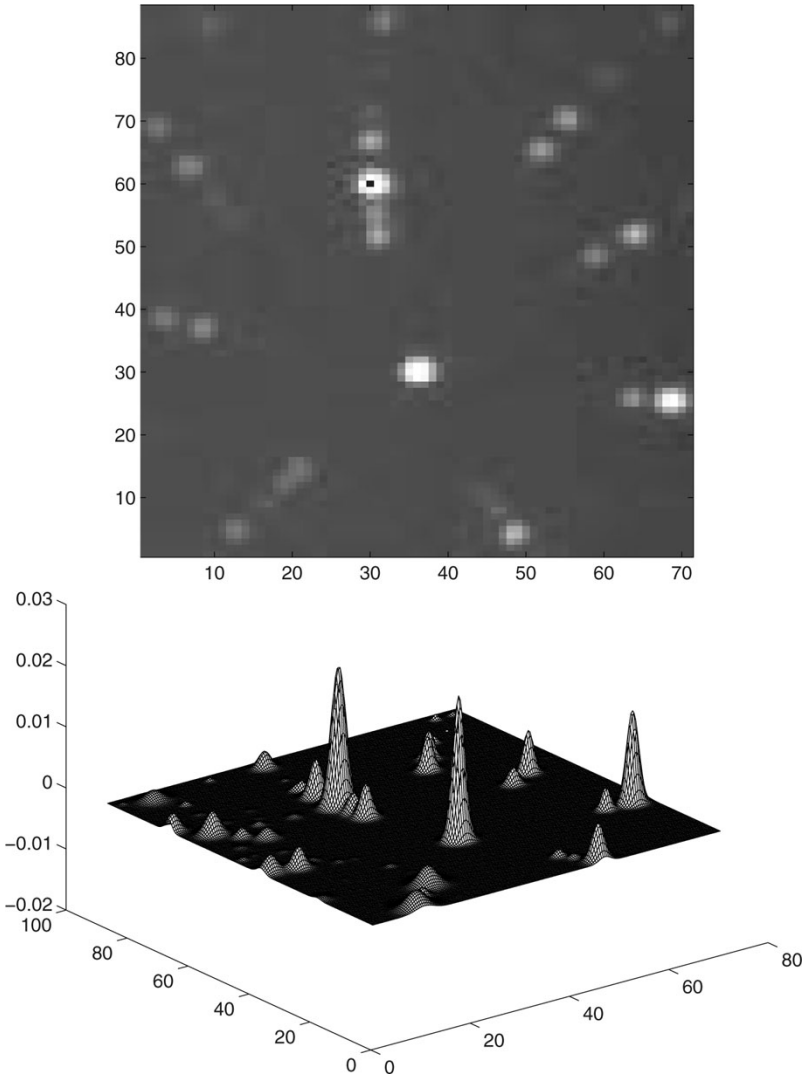


Fig. 8. (a) X-ray diffraction pattern (part); (b) modelled profiles by a 20×20 SOMN (from [129])

re-sampled to provide distribution data. A set of training data (10,647 points in total) was obtained by double sampling this image. A 400-neuron SOMN, arranged in a 20×20 grid, was used to learn this density. In this case, the number of spots (peaks or components) in a pattern (a mixture) will not generally be known *a priori*. The initial positions of the neurons were regularly placed inside the data space – in other words, a $[1, 88] \times [1, 71]$ rectangular grid. The initial variances were assigned equally to a diagonal matrix with the

diagonal values equal to a fraction of the grid size, and the initial mixing priors were assigned equally to $1/400$. The grid was pre-ordered to save unnecessary computational cost as this is a mapping with the same dimension.

After a few learning cycles, the SOMN allocated the spots to the Gaussian kernels and decomposed the overlapping ones. Individual neurons and their parameters provide centre (mean vectors) and width (covariance matrices) information for relevant spots. The total intensity of each peak is readily obtainable and is simply related to its mixing weight. The result of the estimation after five epochs is shown in Fig. 8(b). The number of active nodes (that is, surviving ones) is much less than the initial guess of 400. The SOMN has dynamically fitted to the correct mixture number and suppressed others. As the updating at each input was limited to a small area (3×3 – the winner and its first order neighbourhood, in this example), the SOMN required a much lighter computational effort than updating the entire network at each input (as the EM algorithm would). This becomes particularly advantageous when the number of the nodes is large. In this example, The EM algorithm of the same size would require approximately $400/(3 \times 3) \approx 44$ times more computing effort than the SOMN.

5.4 Gene Expression Analysis

The SOM has been applied as a valid and useful tool for clustering gene expressions [80, 104]. Several attempts have been made to deal with ordered sequence or temporal sequences using SOM. A common approach is to use the trajectories of consecutive winning nodes on the SOM. Other methods are based on modification of the learning topology by introducing recurrent connections, for example the Temporal Kohonen Maps (TKM) or Recurrent SOM (RSOM) mentioned in Sect. 4. In TKM the participation of earlier input vectors in each unit is represented by using a recursive difference equation which defines the current unit activity as a function of the previous activations and the current input vector. In the RSOM, which is a modification of the TKM, the scalar node activities of the TKM are replaced by difference vectors defined as a recursive difference equation of the new input, the previous difference vectors, and the weight vectors of the units. One potential problem with recurrent models is stability. In the case of temporal gene expression clustering, the data items presented to the map are not a spatial vector, but a sequence with time order in itself. They are time-series corresponding to the expression levels over time of a particular gene. Therefore, if a common 2-D SOM is used, the trained map can then be used to mark the trajectories of the expressions of the genes for comparison purposes.

We approach the temporal extension of SOM from another perspective, this being the similarity metric. If the similarity metric takes into account temporal properties, then the neurons in the resultant map will exhibit temporal relationships. As time is one dimensional, a 1-D SOM is more appropriate.

In addition, a circular (that is, a closed 1-D SOM), can further detect cyclic temporal characteristics [80]. A novel temporal metric termed the co-expression coefficient has been defined as [79],

$$ce(x, y) = \frac{\int x' y' dt}{\sqrt{\int x'^2 dt \int y'^2 dt}} \quad (61)$$

where x and y are two (often modelled, thus smoothed) gene expression profiles; x' and y' are their derivatives. It can be seen that the co-expression coefficient is the correlation coefficient of the derivatives of the profiles. Comparing the derivatives of the two better profiles rather than directly on the two profiles captures their temporal properties.

Two yeast cell cycle datasets (208 and 511 genes) were modelled using RBFs and then the modelled profiles were differentiated [80]. The Bayesian information criterion was used to validate the number of clusters obtained by the circular SOM. Each node presents the smaller distance only to its two neighbouring nodes in a chain-ordered fashion, this implies that characteristic traits are split or merged with larger or fewer number of clusters without changing the order or relation between them. Figure 9 presents the resulting SOMs and prototype profiles of the clusters. It can be easily seen that topology exists among the profiles. The topological order here refers to the time shift. This demonstrates that the proposed method is able to group profiles based on their temporal characteristics and can automatically order the groups based on their periodical properties.

Genes identified as cell-cycle-regulated by traditional biological methods have been used to evaluate the performance of the proposed technique. The result shows that the obtained clusters have high relevance to the distribution of these genes among the cell cycle phases identified by biological methods, compared with other clustering methods [80].

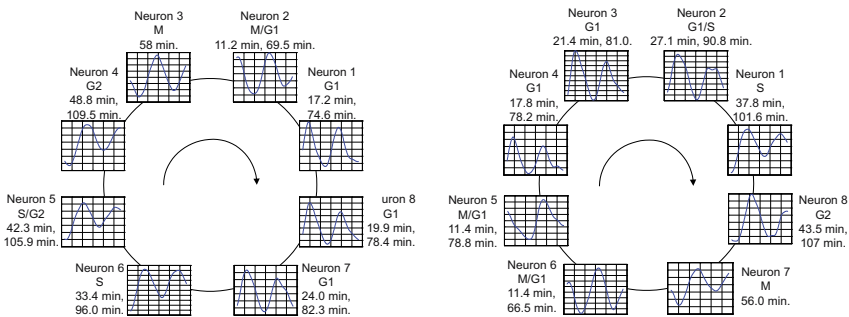


Fig. 9. Circular SOMs for clustering temporal gene expressions (yeast cell cycle dataset): (a) 208-gene dataset; (b) 511-gene dataset (from [80])

5.5 Data Visualization

Data projection and visualization has become a major application area for neural networks, in particular for the SOMs [53], as its topology preserving property is unique among other neural models. Good projection and visualization methods help to identify clustering tendency, to reveal the underlying functions and patterns, and to facilitate decision support. A great deal of research has been devoted to this subject, and a number of methods have been proposed. A recent review on this subject can be found in [121].

The SOM has been widely used as a visualization tool for dimensionality reduction (for instance, [34, 41, 53, 105]). The SOM's unique topology preserving property can be used to visualize the relative mutual relationships among the data. However, the SOM does not directly apply to scaling, which aims to reproduce proximity in (Euclidean) distance on a low visualization space, as it has to rely on a colouring scheme (for example, the U-matrix method [105]) to imprint the distances crudely on the map. Often the distributions of the data points are distorted on the map. The recently proposed ViSOM [118–120], described in Sect. 4.1, constrains the lateral contraction force between the neurons in the SOM and hence regularizes the inter-neuron distances with respect to a scalable parameter that defines and controls the resolution of the map. It preserves the data structure as well as the topology as faithfully as possible. ViSOM provides a direct visualization of both the structure and distribution of the data. An example is shown in Fig. 10, where a 100×100

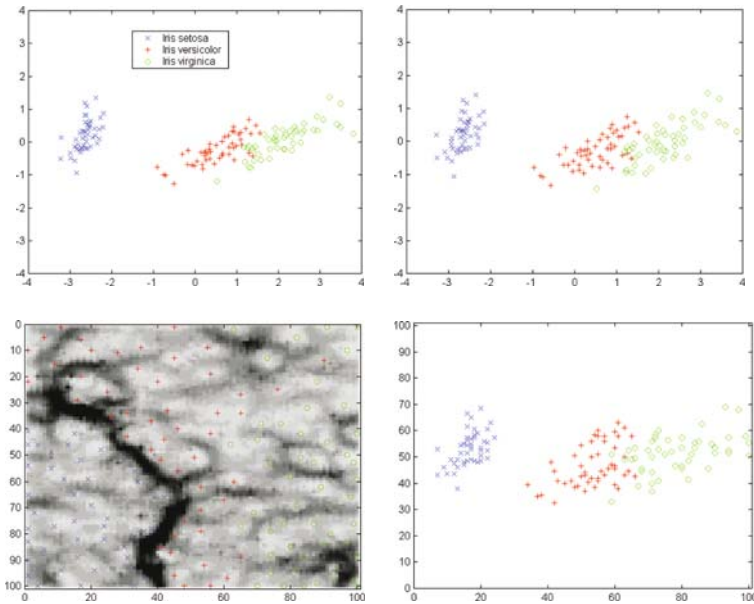


Fig. 10. Mapping and visualization of the iris data set: (*top left*) PCA, (*top right*) Sammon mapping; (*bottom left*) SOM with U matrix Colouring, (*bottom right*) ViSOM

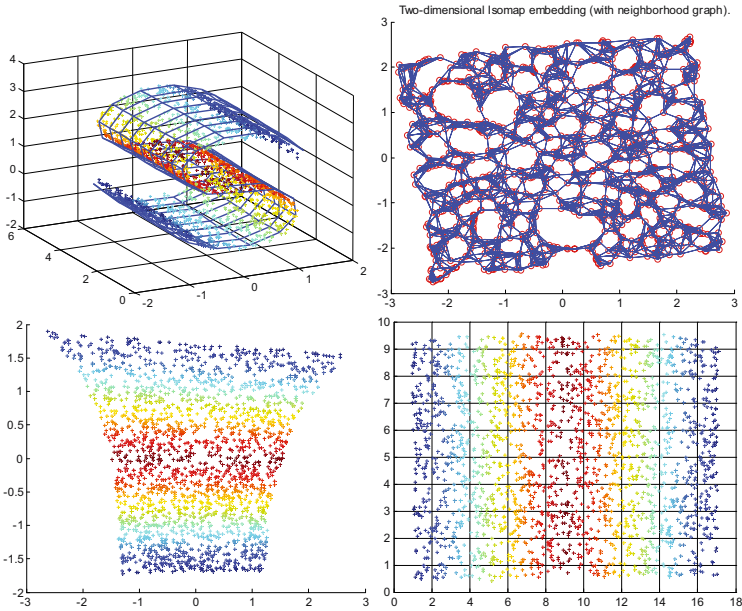


Fig. 11. Manifold mapping by various methods: (*top left*) original S-shape data and ViSOM embedding, (*top right*) Isomap projection; (*bottom left*) LLE projection, (*bottom right*) ViSOM projection

(hexagonal) ViSOM was used to map the 4-D Iris data set; it gives direct visualization of data distribution, similar to Sammon mapping. Although, the SOM with colouring can show the gap between iris setosa and the rest, it is impossible to capture the data structure and represent the data proximity on the map.

Usually for a fine mapping, the resolution parameter needs to be set to a small value. Moreover, a large number of nodes, that is a large map, is required, as for all discrete mappings. However such a computational burden can be greatly reduced by interpolating a trained map [127], or by incorporating a local linear projection on the trained low resolution map [122].

A comparison with other mapping methods, such as PCA, Sammon mapping, Isomap and Local Linear Embedding (LLE) [93] on a highly nonlinear ‘S’ shape manifold is also shown in Fig. 11. In this example, the resolution of the ViSOM is enhanced [122].

5.6 Text Mining and Information Management

With drastically increasing amounts of unstructured content available electronically within an enterprise or on the web, it is becoming inefficient if not impossible to rely on human operators to manually annotate electronic

documents. (Web) content management systems have become an important area of research for many applications, such as e-libraries, enterprise portals, e-commerce, software content management, document management, and knowledge discovery. The documents, generated in an enterprise either centrally or locally by employees, are often unstructured or arranged in *ad hoc* manner (for example, emails, reports, web pages, presentations). Document management addresses many issues, such as storage, indexing, security, revision control, retrieval and organization of documents. Many existing full-text search engines return a large ranked list of documents, many of which are irrelevant. This is especially true when queries are short and very general words are used. Hence document organization has become important in information retrieval and content management.

The SOM has been applied to organize and visualize vast amounts of textual information. Typical examples include the **Welfaremap** [41] and **WEBSOM** [34]. Many SOM variants have been proposed and applied to document organization, for instance, **TreeGCS** [33] and the growing hierarchical-SOM (GH-SOM) [88]. The main advantage of SOM is the topology preservation of input space, which makes similar topics appear closely on the map. Most of these applications however are based on 2-D maps and grids, which are intuitive for the concept of a digital library. However such a presentation of information (mainly document files) is counter to all existing computer file organizers and explorers, such as **MS Windows Explorer**.

We present a new way of utilizing the SOM as a topology-preserving manifold tree-structure for content management and knowledge discovery [23]. The method can generate a taxonomy of topics from a set of unannotated, unstructured documents. It consists of a hierarchy of self-organizing growing chains, each of which can develop independently in terms of size and topics. The dynamic development process is validated continuously using a proposed entropy-based Bayesian information criterion. Each chain meeting the criterion spawns child chains, with reduced vocabularies and increased specializations. This results in a topological tree hierarchy, which can be browsed like a table of contents directory or web portal. A typical tree is shown in Fig. 12. The approach has been tested and compared with several existing methods on real world web page datasets. The results have clearly demonstrated the advantages and efficiency in content organization of the proposed method in terms of computational cost and representation. The preserved topology provides a unique, additional feature for retrieving related topics and confining the search space.

An application prototype developed based this method is shown in Fig. 13. The left panel displays the generated content tree with various levels and preserved topology on these levels. The right panel shows the details of a selected level or branch or a particular document. The method bears a similar interface to many computer file managers, especially the most popular **MS Windows Explorer** style.

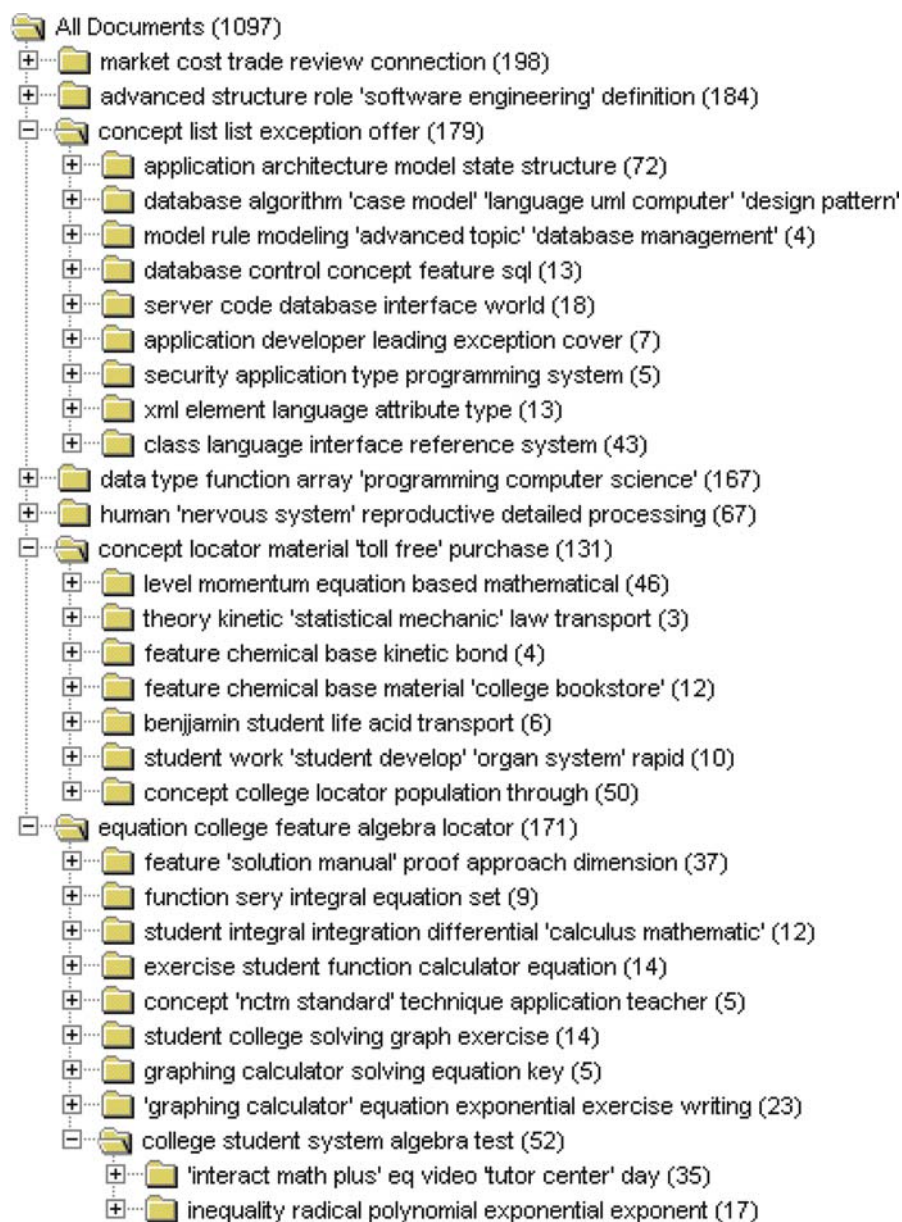


Fig. 12. A typical result of using a topological tree structure for organizing documents

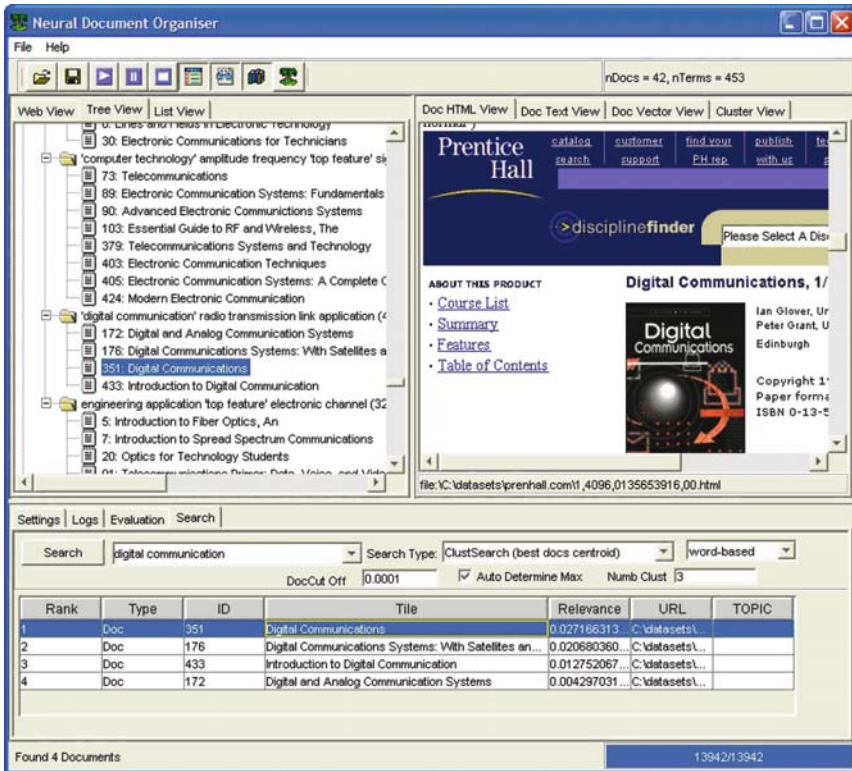


Fig. 13. Screen shot of a document management system developed using a topological tree structure

6 Summary and Future Directions

This Chapter provides an overview and review on the self-organizing map (SOM). First, it reviewed the biological background of SOM and showed that it is a simplified and abstract mathematical model of the retina-cortex mapping based on Hebbian learning and the lateral inhibition phenomena. Then from the mathematics of the algorithm, we discussed and explained its underlying cost function and various measures for mapping quality. Then its variant, the visualization induced SOM (ViSOM), was proposed for preserving local metrics on the map, and reviewed for use in data visualization and nonlinear manifold mapping. The relationships between SOM, ViSOM, multidimensional scaling, principal curve/surface, kernel PCA and several other nonlinear projection methods were analyzed and discussed. Both the SOM and ViSOM are multidimensional scaling methods and produce nonlinear dimension-reduction mapping or manifold of the input space. The SOM was shown to be a qualitative scaling method, while the ViSOM is a metric scaling method

and approximates a discrete principal curve/surface. The SOM has also been extended to a probabilistic model and the resulting self-organizing mixture model also reveals that self-organization is an entropy-related optimization process. Furthermore, such a self-organizing model naturally approximates the kernel method. Examples and typical applications of SOM and ViSOM in the context of pattern recognition, clustering, classification, data visualization and mining, and nonlinear manifolds were presented.

Future challenges lie in several areas. First, although the SOM-related methods are finding wide application in more and more fields, to make the methods more efficient, robust and consistent is a key challenge, especially for large-scale, real-world applications. To adapt the methods for various input formats and conditions, such as temporal sequences and qualitative inputs, is also an on-going research focus. For general pattern recognition, the SOM may have more potential than implied by current practice, which often limits the SOM to a 2-D map and empirically chosen model parameters. Ways of applying and extending SOM for optimal clustering and classification also need to be investigated further. Last but not the least, to make this biologically inspired model more biologically relevant is also a key challenge. The model may have to be further extended in order to deal with complex biological signals and networks, for example in handling spikes and more importantly multiple, perhaps inhomogeneous and population spike trains. A synergy with other biologically relevant models seems necessary for modeling large-scale complex biological systems, especially the brain. Neural coding is widely studied under information theory. Probabilistic extensions of the SOM may provide useful tools in deciphering and interpreting the information content and relationships conveyed among stimuli and responses.

References

1. Allinson NM, Obermayer K, Yin H (2002) *Neural Networks* (Special Issue on New Developments in Self-Organising Maps), 15: 937–1155.
2. Allinson NM, Yin H (1999) Self-organising maps for pattern recognition. In: Oja E, Kaski S (eds.) *Kohonen Maps*, Elsevier, Amsterdam, The Netherlands: 111–120.
3. Ameri S-I (1980) Topographic organisation of nerve fields. *Bulletin Mathematical Biology*, 42: 339–364.
4. Andras P (2002) Kernel-Kohonen networks. *Intl. J. Neural Systems*, 12: 117–135.
5. Banfield JD, Raftery AE (1992) Ice floe identification in satellite images using mathematical morphology and clustering about principal curves. *J. American Statistical Association*, 87: 7–16.
6. Barreto GA, Araujo AFR, Ducker C, Ritter H (2002) A distributed robotic control system based on a temporal self-organizing neural network. *IEEE Trans. Systems, Man and Cybernetics – C*, 32: 347–357.

7. Bauer H-U, Pawelzik KR (1992) Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Trans. Neural Networks*, 3: 570–579.
8. Bishop CM, Svensén M, Williams CKI (1998) GTM: the generative topographic mapping. *Neural Computation*, 10: 215–235.
9. Bruce V, Green PR (1990) *Visual Perception: Physiology, Psychology and Ecology (2nd ed.)*, Lawrence Erlbaum Associates, East Essex, UK.
10. Chang K-Y, Ghosh J (2001) A unified model for probabilistic principal surfaces. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23: 22–41.
11. Chappell GJ, Taylor JG (1993) The temporal Kohonen map. *Neural Networks*, 6: 441–445.
12. Cottrell M, Fort JC (1986) A stochastic model of retinotopy: a self-organising process. *Biological Cybernetics*, 53: 405–411.
13. Cottrell M, Verleysen M (2006) *Neural Networks* (Special Issue on Advances in Self-Organizing Maps), 19: 721–976.
14. Cox TF, Cox MAA (1994) *Multidimensional Scaling*, Chapman & Hall, London, UK.
15. de Bolt E, Cottrell M, Verleysen M (2002) Statistical tools to assess the reliability of self-organising maps. *Neural Networks*, 15: 967–978.
16. De Ridder D, Duin RPW (1997) Sammon mapping using neural networks: a comparison. *Pattern Recognition Letters*, 18: 1307–1316.
17. Dersch DR, Tavan P (1995) Asymptotic level density in topological feature maps. *IEEE Trans. Neural Networks*, 6: 230–236.
18. Durbin R, Mitchison G (1990) A dimension reduction framework for understanding cortical maps. *Nature*, 343: 644–647.
19. Erwin E, Obermayer K, Schulten K (1992) Self-organising maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67: 47–55.
20. Erwin E, Obermayer K, Schulten K (1992) Self-organising maps: stationary states, metastability and convergence rate. *Biological Cybernetics*, 67: 35–45.
21. Estévez PA, Figueroa CJ (2006) Online data visualization using the neural gas network. *Neural Networks*, 19: 923–934.
22. Ferguson KL, Allinson NM (2004) Efficient video compression codebooks using SOM-based vector quantisation. *Proc. IEE – Vision, Image and Signal Processing*, 151: 102–108.
23. Freeman R, Yin H (2004) Adaptive topological tree structure (ATTS) for document organisation and visualisation. *Neural Networks*, 17: 1255–1271.
24. Gaze RM (1970) *The Information of Nerve Connections*, Academic Press, London, UK.
25. Goodhill GJ, Sejnowski T (1997) A unifying objective function for topographic mappings. *Neural Computation*, 9: 1291–1303.
26. Graepel T, Burger M, Obermayer K (1997) Phase transitions in stochastic self-organizing maps. *Physics Reviews E*, 56: 3876–3890.
27. Haritopoulos M, Yin H, Allinson NM (2002) Image denoising using self-organising map-based nonlinear independent component analysis. *Neural Networks*, 15: 1085–1098.
28. Hastie T, Stuetzle W (1989) Principal curves. *J. American Statistical Association*, 84: 502–516.
29. Haykin S (1998) *Neural Networks: A Comprehensive Foundation (2nd ed.)*, Prentice Hall, Englewood Cliffs, NJ.
30. Hebb D (1949) *Organisation of behavior*, Wiley, New York, NY.

31. Herrmann M, Yang H (1996) Perspectives and limitations of self-organising maps in blind separation of source signals. In: Amari S-I, Xu L, Chan L-W, KIng I, Leung K-S (eds.) *Proc. Intl. Conf. Neural Information Processing (ICONIP'96)*, 24–27 September, Hong Kong. Springer-Verlag, Singapore: 1211–1216.
32. Heskes T (1999) Energy functions for self-organizing maps, In: Oja E, Kaski S (eds.) *Kohonen Maps*, Elsevier, Amsterdam: 303–315.
33. Hodge VJ, Austin J (2001) Hierarchical growing cell structures: TreeGCS. *IEEE Trans. Knowledge and Data Engineering*, 13: 207–218.
34. Honkela T, Kaski S, Lagus K, Kohonen T (1997) WEBSOM-self-organizing maps of document collections. In: *Proc. Workshop on Self-Organizing Maps (WSOM'97)*, 4–6 June, Helsinki, Finland. Helsinki University of Technology: 310–315.
35. Hsu C-C (2006) Generalising self-organising map for categorical data. *IEEE Trans. Neural Networks*, 17: 294–304.
36. Hyvärinen A, Karhunen J, Oja E (2001) *Independent Component Analysis*. Wiley, New York, NY.
37. Hyvärinen A, Pajunen P (1999) Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 12: 429–439.
38. Ishikawa M, Miikkulainen R, Ritter H (2004) *Neural Networks* (Special Issue on New Developments in Self-Organizing Systems), 17: 1037–1389.
39. Karhunen J, Joutsensalo J (1995) Generalisation of principal component analysis, optimisation problems, and neural networks. *Neural Networks*, 8: 549–562.
40. Kaski S, Kangas J, Kohonen T (1998) Bibliography of self-organizing map (SOM) papers: 1981–1997. *Neural Computing Surveys*, 1: 1–176.
41. Kaski S, Kohonen T (1996) Exploratory data analysis by the self-organizing map: Structures of welfare and poverty in the world. In: Refenes A-PN, Abu-Mostafa Y, Moody J, Weigend A (eds.) *Neural Networks in Financial Engineering*, World Scientific, Singapore: 498–507.
42. Kegl B, Krzyzak A, Linder T, Zeger K (1998) A polygonal line algorithm for constructing principal curves. *Neural Information Processing Systems (NIPS'98)*, 11: 501–507.
43. Kohonen T (1972) Correlation matrix memory. *IEEE Trans. Computers*, 21: 353–359.
44. Kohonen T (1973) A new model for randomly organised associative memory. *Intl. J. Neuroscience*, 5: 27–29.
45. Kohonen T (1974) An adaptive associative memory principle. *IEEE Trans. Computers*, 23: 444–445.
46. Kohonen T (1982) Self-organised formation of topologically correct feature map. *Biological Cybernetics*, 43: 56–69.
47. Kohonen T (1984) *Self-organization and Associative Memory*, Springer-Verlag, Berlin.
48. Kohonen T (1986) Representation of sensory information in self-organising feature maps, and relation of these maps to distributed memory networks. *Proc. SPIE*, 634: 248–259.
49. Kohonen T (1987) Adaptive, associative, and self-organizing functions in neural computing, *Applied Optics*, 26: 4910–4918.

50. Kohonen T (1991) Self-organizing maps: optimization approaches. In: Kohonen T, Makisara K, Simula O, Kangas J (eds.) *Artificial Neural Networks 2*, North-Holland, Amsterdam, The Netherlands: 981–990.
51. Kohonen T (1995) The adaptive-subspace SOM (ASSOM) and its use for the implementation of invariant feature detection. In: Fogelman-Soulié, Gallinari P (eds.) *Proc. Intl. Conf. Artificial Neural Systems (ICANN'95)*, 9–13 October, Paris, France. EC2 Nanterre, France, 1: 3–10.
52. Kohonen T (1996) Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map. *Biological Cybernetics*, 75: 281–291.
53. Kohonen T (1997) *Self-Organising Maps (2nd ed.)*. Springer-Verlag, Berlin.
54. Kohonen T (1999) Comparison of SOM point densities based on different criteria. *Neural Computation*, 11: 2081–2095.
55. Kohonen T, Somervuo P (2002) How to make a large self-organising maps for nonvectorial data. *Neural Networks*, 15: 945–952.
56. Kramer MA (1991) Nonlinear principal component analysis using autoassociative neural networks. *American Institute Chemical Engineers J.*, 37: 233–243.
57. Laaksonen J, Koskela M, Laakso S, Oja E (2000) PicSOM - content-based image retrieval with self-organizing maps. *Pattern Recognition Letters*, 21: 1199–1207.
58. Lampinen J, Oja E (1992) Clustering properties of hierarchical self-organizing maps. *J. Mathematical Imaging and Vision*, 2: 261–272.
59. Lau KW, Yin H, Hubbard S (2006) Kernel self-organising maps for classification. *Neurocomputing*, 69: 2033–2040.
60. LeBlanc M, Tibshirani RJ (1994) Adaptive principal surfaces. *J. American Statistical Association*, 89: 53–64.
61. Lee RCT, Slagle JR, Blum H (1977) A triangulation method for the sequential mapping of points from n-space to two-space. *IEEE Trans. Computers*, 27: 288–292.
62. Lin S, Si J (1998) Weight-value convergence of the SOM algorithm for discrete input. *Neural Computation*, 10: 807–814.
63. Linde Y, Buzo A, Gray RM (1980) An algorithm for vector quantizer design. *IEEE Trans. Communications*, 28: 84–95.
64. Lo ZP, Bavarian B (1991) On the rate of convergence in topology preserving neural networks. *Biological Cybernetics*, 65: 55–63.
65. Lowe D, Tipping ME (1996) Feed-forward neural networks and topographic mappings for exploratory data analysis. *Neural Computing and Applications*, 4: 83–95.
66. Luttrell SP (1990) Derivation of a class of training algorithms. *IEEE Trans. Neural Networks*, 1: 229–232.
67. Luttrell SP (1991) Code vector density in topographic mappings: Scalar case, *IEEE Trans. Neural Networks*, 2: 427–436.
68. Luttrell SP (1994) A Bayesian analysis of self-organising maps. *Neural Computation*, 6: 767–794.
69. MacDonald D, Fyfe C (2000) The kernel self organising map. In: *Proc. 4th Intl. Conf. Knowledge-based Intelligence Engineering Systems and Applied Technologies*, 30 August – 1 September, Brighton, UK, IEEE Press, Piscataway, NJ: 317–320.
70. Malthouse EC (1998) Limitations of nonlinear PCA as performed with generic neural networks. *IEEE Trans. Neural Networks*, 9: 165–173.

71. Mao J, Jain AK (1995) Artificial Neural Networks for Feature Extraction and Multivariate Data Projection. *IEEE Trans. Neural Networks*, 6: 296–317.
72. Marr D (1969) A theory of cerebellar cortex. *J. Physiology*, 202: 437–70.
73. Marsland S, Shapiron J, Nehmzow U (2002) A self-organising network that grows when required. *Neural Networks*, 15: 1041–1058.
74. Martinetz TM, Schulten KJ (1991) A “neuralgas” network learns topologies. In: Kohonen T, Mäkisara K, Simula O, Kangas J (eds.) *Artificial Neural Networks*, NorthHolland, Amsterdam, The Netherlands: 397–402.
75. Martinetz TM, Schulten KJ (1994) Topology representing networks. *Neural Networks*, 7: 507–522.
76. Miikkulainen R (1990) Script recognition with hierarchical feature maps. *Connection Science*, 2: 83–101.
77. Miikkulainen R (1997) Dyslexic and category-specific aphasic impairments in a self-organizing feature map model of the lexicon. *Brain and Language*, 59: 334–366.
78. Mitchison G (1995) A type of duality between self-organising maps and minimal wiring. *Neural Computation*, 7: 25–35.
79. Möller-Levet CS, Yin H (2005) Modeling and analysis of gene expression time-series based on co-expression. *Intl. J. Neural Systems*, (Special Issue on Bioinformatics), 15: 311–322.
80. Möller-Levet CS, Yin H (2005) Circular SOM for temporal characterisation of modelled gene expressions. In: Gallagher M, Hogan J, Maire F (eds.) *Proc. Intl. Conf. Intelligent Engineering Data Engineering and Automated Learning Conf. (IDEAL'05)*, 6–8 July, Brisbane, Australia. Lecture Notes in Computer Science 3578, Springer-Verlag, Berlin: 319–326.
81. Oja E (1989) Neural networks, principal components, and subspaces. *Intl. J. Neural Systems*, 1: 61–68.
82. Oja E (1995) PCA, ICA, and nonlinear Hebbian learning. In: Fogelman-Soulié F, Gallinari P (eds.) *Proc. Intl. Conf. Artificial Neural Networks (ICANN'95)*, 9–13 October, Paris, France. EC2, Nanterre, France: 89–94.
83. Oja M, Kaski S, Kohonen T (2003) Bibliography of self-organizing map (SOM) papers: 1998–2001 addendum. *Neural Computing Surveys*, 3: 1–156.
84. Pajunen P, Hyvärinen A, Karhunen J (1996) Nonlinear blind source separation by self-organising maps. In: Amari S-I, Xu L, Chan L-W, King I, Leung K-S (eds.) *Proc. Intl. Conf. Neural Information Processing (ICONIP'96)*, 24–27 September, Hong Kong. Springer-Verlag, Singapore: 1207–1210.
85. Pan ZS, Chen SC, Zhang DQ (2004) A kernel-base SOM classifier in input space. *Acta Electronica Sinica*, 32: 227–231 (in Chinese).
86. Pearson D, Hanna E, Martinez K (1990) Computer-generated cartoons. In: Barlow H, Blakemore C, Weston-Smith M (eds.) *Images and Understandings*, Cambridge University Press, Cambridge, UK.
87. Ratcliff F (1965) *Mach Bands: Quantitative Studies on Neural Networks in the Retina*. Holden-Day, Inc., San Francisco, CA.
88. Rauber A, Merkl D, Dittenbach M (2002) The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. *IEEE Trans. Neural Networks*, 13: 1331–1341.
89. Ritter H (1991) Asymptotical level density for class of vector quantisation processes. *IEEE Trans. Neural Networks*, 2: 173–175.

90. Ritter H, Schulten K (1988) Convergence properties of Kohonen's topology conserving maps: fluctuations, stability, and dimension selection. *Biological Cybernetics*, 60: 59–71.
91. Ritter H, Martinetz T, Schulten K (1992) *Neural Computation and Self-organising Maps: An Introduction*. Addison-Wesley, Reading, MA.
92. Robbins H, Monro S (1952) A stochastic approximation method. *Annals Mathematical Statistics*, 22: 400–407.
93. Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290: 2323–2326.
94. Sakrison DJ (1966) Stochastic approximation: A recursive method for solving regression problems. In: Balakrishnan V (ed.) *Advances in Communication Systems: Theory and Applications 2*, Academic Press, New York, NY: 51–100.
95. Sammon JW (1969) A nonlinear mapping for data structure analysis. *IEEE Trans. Computers*, 18: 401–409.
96. Sanger TD (1991) Optimal unsupervised learning in a single-layer linear feedforward network. *Neural Networks*, 2: 459–473.
97. Schölkopf B, Smola A, Müller KR (1998) Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10: 1299–1311.
98. Seiffert U, Michaelis B (2001) Multi-dimensional self-organizing maps on massively parallel hardware. In: Allinson N, Yin H, Allinson L, Slack J (eds.) *Advances in Self-Organising Maps*, Springer-Verlag, London: 160–166.
99. Shepherd GM (1988) *Neurobiology (2nd ed.)*, Oxford University Press, Oxford, UK.
100. Sum J, Leung C-S, Chan L-W, Xu L (1997) Yet another algorithm which can generate topography map. *IEEE Trans. Neural Networks*, 8: 1204–1207.
101. Sutton RS, Barto AG, Williams RJ (1991) Reinforcement learning is direct adaptive optimal control. In: *Proc. American Control Conf.*, 26–28 June, Boston, MA. IEEE Press, Piscataway, NJ: 2143–2146.
102. Tenenbaum JB, de Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. *Science*, 290: 2319–2323.
103. Tibshirani R (1992) Principal curves revisited. *Statistics and Computation*, 2: 183–190.
104. Törönen P, Kolehmainen K, Wong G, Castrén E (1999) Analysis of gene expression data using self-organising maps. *Federation European Biochemical Societies Letters*, 451: 142–146.
105. Ultsch A (1993) Self-organising neural networks for visualisation and classification. In: Opitz O, Lausen B, Klar R (eds.) *Information and Classification*, Springer-Verlag, Berlin: 864–867.
106. Van Hulle MM (1998) Kernel-based equiprobabilistic topographic map formation. *Neural Computation*, 10: 1847–1871.
107. Van Hulle MM (2002) Kernel-based equiprobabilistic topographic map formation achieved with an information-theoretic approach. *Neural Networks*, 15: 1029–1040.
108. Varsta M, del Ruiz Millán J, Heikkonen J (1997) A recurrent self-organizing map for temporal sequence processing. *Proc. ICANN'97*, Lausanne, Switzerland. Springer-Verlag, Berlin: 197–202.
109. Villmann T, Der R, Herrmann M, Martinetz TM (1997) Topology preservation in self-organizing feature maps: exact definition and measurement. *IEEE Trans. Neural Networks*, 8: 256–266.

110. Voegtlin T (2002) Recursive self-organizing maps. *Neural Networks*, 15: 979–992.
111. von der Malsburg C, Willshaw DJ (1973) Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 4: 85–100.
112. Walter J, Ritter H (1996) Rapid learning with parametrized self-organizing maps. *Neurocomputing*, 12: 131–153.
113. Wang Y, Yin H, Zhou L-Z, Liu Z-Q (2006) Real-time synthesis of 3D animations by learning parametric gaussians using self-organizing mixture networks. In: King I, Wang J, Chan L, Wang D (eds.) *Proc. Intl. Conf. Neural Information Processing (ICONIP'06)*, 2–6 October, Hong Kong. Lecture Notes in Computer Science 4233, Springer-Verlag, Berlin, II: 671–678.
114. Willshaw DJ, Buneman OP, Longnet-Higgins HC (1969) Non-holographic associative memory. *Nature*, 222: 960–962.
115. Willshaw DJ, von der Malsburg C (1976) How patterned neural connections can be set up by self-organization. *Proc. Royal Society of London – Series B*, 194: 431–445.
116. Wu S, Chow TWS (2005) PRSOM: A new visualization method by hybridizing multidimensional scaling and self-organizing map. *IEEE Trans. Neural Networks*, 16: 1362–1380.
117. Yin H (1996) Self-Organising Maps: Statistical Analysis, Treatment and Applications, *PhD Thesis*, Department of Electronics, University of York, UK.
118. Yin H (2001) Visualisation induced SOM (ViSOM). In: Allinson N, Yin H, Allinson L, Slack J (eds.) *Advances in Self-Organising Maps*, Springer-Verlag, London, UK: 81–88.
119. Yin H (2002) ViSOM-A novel method for multivariate data projection and structure visualisation. *IEEE Trans. Neural Networks*, 13: 237–243.
120. Yin H (2002) Data visualisation and manifold mapping using the ViSOM. *Neural Networks*, 15: 1005–1016.
121. Yin H (2003) Nonlinear multidimensional data projection and visualisation. In: Liu J, Cheung Y, Yin H (eds.) *Proc. Intl. Conf. Intelligent Data Engineering and Automated Learning (IDEAL'03)*, 21–23 March, Hong Kong. Lecture Notes in Computer Science 2690, Springer-Verlag, Berlin: 377–388.
122. Yin H (2003) Resolution enhancement for the ViSOM. In: *Proc. Workshop on Self-Organizing Maps*, 11–14 September, Kitakyushu, Japan. Kyushu Institute of Technology: 208–212.
123. Yin H (2006). On the equivalence between kernel self-organising maps and self-organising mixture density networks. *Neural Networks*, 19: 780–784.
124. Yin H (2007) Connection between self-organising maps and metric multidimensional scaling. In: *Proc. Intl. Joint Conf. Neural Networks (IJCNN2007)*, 12–17 August, Orlando, FL. IEEE Press, Piscataway, NJ: (in press).
125. Yin H, Allinson NM (1994) Unsupervised segmentation of textured images using a hierarchical neural structure. *Electronics Letters*, 30: 1842–1843.
126. Yin H, Allinson NM (1995) On the distribution and convergence of the feature space in self-organising maps. *Neural Computation*, 7: 1178–1187.
127. Yin H, Allinson NM (1999) Interpolating self-organising map (iSOM). *Electronics Letters*, 35: 1649–1650.
128. Yin H, Allinson NM (2001) Self-organising mixture networks for probability density estimation. *IEEE Trans. Neural Networks*, 12: 405–411.
129. Yin H, Allinson NM (2001) Bayesian self-organising map for Gaussian mixtures. *Proc. IEE – Vision, Image and Signal Processing*, 148: 234–240.

Resources

1 Key Books

Kohonen T (1995, 1997, 2001) *Self-Organizing Maps (1st, 2nd and 3rd editions)*. Springer-Verlag, Berlin.

Haykin S (1994) *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, NY.

Haykin S (1999) *Neural Networks: A Comprehensive Foundation (2nd ed.)*. Prentice Hall, Englewood Cliffs, NJ.

Oja E, Kaski S (eds.) (1999) *Kohonen Maps*. Elsevier, Amsterdam, The Netherlands.

Allinson N, Yin H, Allinson L, Slack J (eds.) (2001) *Advances in Self-Organising Maps*. Springer-Verlag, London, UK.

Ritter H, Martinetz T, Schulten K (1992) *Neural Computation and Self-organising Maps: An Introduction*. Addison Wesley, Reading, MA.

Van Hulle MM (2000) *Faithful Representations and Topographic Maps: From Distortion to Information Based Self-Organization*. Wiley, New York, NY.

2 Key Survey/Review Articles

Allinson NM, Obermayer K, Yin H (eds.) (2002) Special Issue on New Developments in Self-Organising Maps, *Neural Networks*, 15(8–9): 937–1155.

Ishikawa M, Miikkulainen R, Ritter H (eds.) (2004) Special Issue on New Developments in Self-Organizing Systems, *Neural Networks*, 17(8-9): 1037–1389.

Cottrell M, Verleysen M (eds.) (2006) Special Issue on Advances in Self-Organizing Maps, *Neural Networks*, 19(5–6): 721–976.

3 Key International Conferences/Workshops

WSOM – Intl. Workshop on Self-Organizing Maps (*biennial, since 1997*).

IJCNN – Intl. Joint Conference on Neural Networks (*annual, since 1987*).

ICANN – Intl. Conference on Artificial Neural Networks (*annual, since 1991*).

ESANN – European Symp. Artificial Neural Networks (*annual, since 1993*).

ICONIP – Intl. Conf. Neural Information Processing (*annual, since 1994*).

IDEAL – International Conference on Intelligent Data Engineering and Automated Learning (*annual, since 1998*).

ISNN – Intl. Symposium on Artificial Neural Networks (*annual, since 2004*).

4 (Open Source) Software

SOM Toolbox

<http://www.cis.hut.fi/projects/somtoolbox/>

Neural Systems Engineering

Steve Furber and Steve Temple

School of Computer Science, University of Manchester, Oxford Road,
Manchester M13 9PL, UK*, steve.furber@manchester.ac.uk,
steven.temple@manchester.ac.uk

1 Introduction

Biological brains and engineered electronic computers fall into different categories. Both are examples of complex information processing systems, but beyond this point their differences outweigh their similarities. Brains are flexible, imprecise, error-prone and slow; computers are inflexible, precise, deterministic and fast. The sets of functions at which each excels are largely non-intersecting. They simply seem to be different types of system. Yet throughout the (admittedly still rather short) history of computing, scientists and engineers have made attempts to cross-fertilize ideas from neurobiology into computing in order to build machines that operate in a manner more akin to the brain. Why is this?

Part of the answer is that brains display very high levels of concurrency and fault-tolerance in their operation, both of which are properties that we struggle to deliver in engineered systems. Understanding how the brain achieves these properties may help us discover ways to transfer them to our machines. In addition, despite their impressive ability to process numbers at ever-increasing rates, computers continue to be depressingly dumb, hard to use and totally lacking in empathy for their hapless users. If we could make interacting with a computer just a bit more like interacting with another person, life would be so much easier for so many people.

More fundamentally, understanding how the brain functions is one of the last great frontiers of science. ‘Wet’ neuroscience has revealed a great deal about the structure and operation of individual neurons, and medical instruments such as functional magnetic resonance imaging machines reveal a great deal about how neural activity in the various regions of the brain follows a

* This Chapter originally appeared as an article in *J. Royal Society Interface*, 2007, 4: 193–206; permission by the Royal Society to reprint it in the current Handbook is gratefully acknowledged.

sensory stimulus. But there is a wide gulf between these micro- and macro-level perspectives on brain function. Somewhere in this gulf are issues such as neural codes: how do populations of neurons jointly encode sensory and high-level information, modularity, temporal behavior, memory (short- and long-term), attention and, of course, consciousness?

The objective of understanding the architecture of brain and mind is recognized as one of the grand challenges in computing research [43] and is a long-term multi-disciplinary project pursued at many different levels of abstraction.

The computer engineer brings a constructionist approach to these issues. Given the various different models that have been offered to describe the information processing function of an individual neuron, how do we go about selecting an appropriate model and constructing useful computational functions using it (instead of logic gates) as the basic component part? Can we build a library of such functions as a kit of parts from which to construct higher-level systems? Will the results of this enterprise deliver new and better ways to build computers, and/or will it tell us anything at all about the biological systems that are the source of inspiration for this approach?

Therefore, we have two goals in this work: (i) to develop a ‘neural toolkit’ that can be used to build computers that share some of the properties of the brain, such as high levels of concurrency and fault-tolerance, and (ii) to build a machine that will allow realistic simulation and study of the brain itself. In this review, we present a framework for this field of inquiry, suggest promising lines of attack, and indicate when and where answers may emerge, so far as this can be foreseen.

1.1 The Neuron

The basic biological control component is the neuron. A full understanding of the ‘architecture of brain and mind’ [43] must, ultimately, involve finding an explanation of the phenomenological observations that can be expressed in terms of the interactions between neurons.

Neurons appear to be very flexible components whose utility scales over systems covering a vast range of complexities. Very simple creatures find a small number of neurons useful. Honeybees find it economic to support brains comprising around 850,000 neurons, which give them exceptional navigational capabilities while travelling several miles from their hive. Humans have evolved to carry brains comprising 10^{11} neurons or so and use these to support exceptional motor control and complex societal interactions. Figure 1 illustrates a very small part of the cortex and gives an indication of the extent of the interconnections between neurons.

The basic logic gate used in digital circuits can be considered to be ‘universal’, in the sense that any digital circuit can be built using the same basic

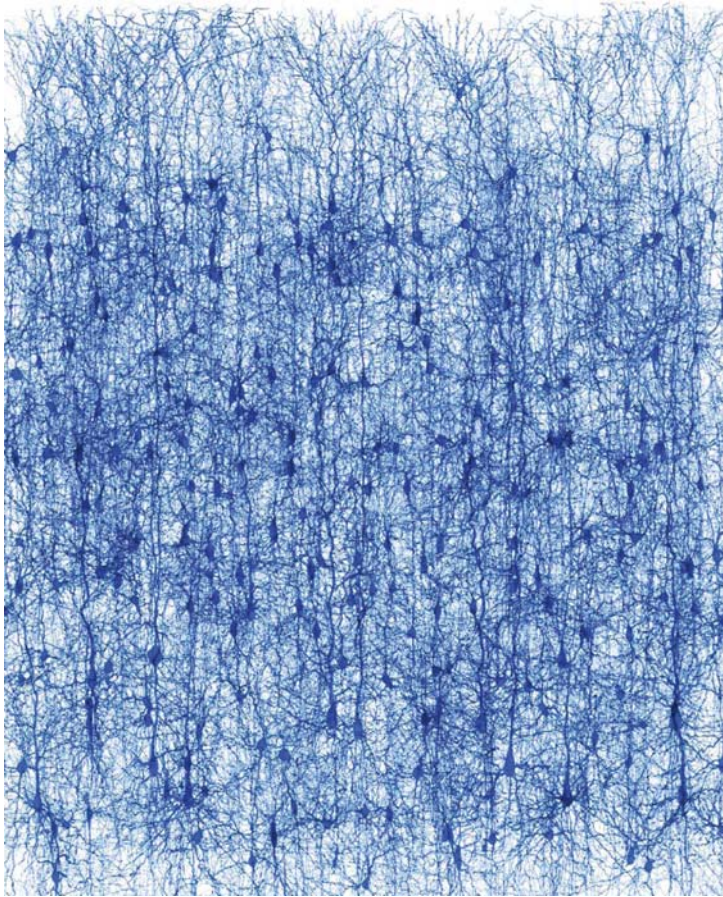


Fig. 1. A view of the neuron cells and connections from a very small area of the cortex (photo courtesy of the Brain Mind Institute, EPFL, Lausanne, Switzerland)

gate provided that a sufficient number of these gates are available - one structure can support all the functions within a class. The component neuron used across the range of biological brains is basically the same in its principles of operation, so in some sense it enjoys a universality similar to that of the logic gate in digital engineering, though the family of neurons employed in biological systems displays considerably more diversity in its members' physical characteristics.

There is a further similarity between neurons and logic gates: both are multiple-input single-output components. However, while the typical fan-in (the number of inputs to a component) and fan-out (the number of other components the output of a particular component connects to) of a logic gate are in the range 2–4, neurons typically have a fan-in and fan-out in the range 1,000–10,000.

A more subtle difference between a logic gate and a neuron is in their internal dynamics. Whereas a logic gate implements a process that is essentially static and defined by Boolean logic, so that at any time (from a short time after the last input change) the output is a well-defined stable function of the inputs, a neuron has complex dynamics that includes several time constants, maintains a more complex internal state, and its output is a time-series of action potentials or ‘spikes’. The information conveyed by the neuron’s output is encoded in the timing of the spikes in a way that is not yet fully understood, although rate codes, population codes and firing-order codes all seem to offer valid interpretations.

Accurate computer models of biological neurons exist, but they are very complex (for example, the Hodgkin-Huxley model [16]). Various simpler models have been proposed that capture some of the features of the biology but omit others. The difficulty lies in determining which of the features are essential to the information processing functions of the neuron and which are artefacts resulting from the way the cell developed, its need to sustain itself, and the complex evolutionary processes that led to its current form.

1.2 Neural Microarchitecture

The universality of the neuron as a component is also reflected in certain higher-level structures of the brain. For example, the cortex displays a six-layer structure and a regularity of interconnect between the neurons in the various layers [33] that suggest the use here of a neural ‘micro-architecture’. The same regular laminar cortical micro-architecture is in evidence across the cortex in regions implementing low-level vision processes such as edge detection and in regions involved in high-level functions such as speech and language processing. This apparent ‘universality’ (used here as defined earlier) of the cortical micro-architecture suggests that there are principles being applied, the understanding of which could offer a breakthrough in our understanding of brain function.

In contrast to the regularity and uniformity of the micro-architecture, the particular connectivity patterns that underpin these structures appear to be stochastic, guided by statistical principles rather than specific connectivity plans. The connectivity is also locally adaptive, so the system can be refined through tuning to improve its performance, a process termed ‘neuroplasticity’ [41].

1.3 Engineering with Neurons

As computer engineers we find the neuron’s universality across wide ranges of biological complexity to be intriguing, and there is a real challenge in understanding how this component can be used to build useful information

processing systems. There is an existence proof that this is indeed possible, but few pointers to how the resulting systems might work.

There are other ‘engineering’ aspects of biological neurons that are interesting, too. We have already mentioned the regularity of neural micro-architecture. Neurons are physically much larger than transistors, having cell bodies with dimensions typically less than $30\ \mu\text{m}$ whereas today’s transistors have dimensions below $0.1\ \mu\text{m}$. The power efficiency of neurons (measured as the energy required to perform a given computation) exceeds that of computer technology, possibly because the neuron itself is a relatively slow component. While computer engineers measure gate speeds in picoseconds, neurons have time constants measured in milliseconds. While computer engineers worry about speed-of-light limitations and the number of clock cycles it takes to get a signal across a chip, neurons communicate at a few metres per second. This very relaxed performance at the technology level is, of course, compensated by the very high levels of parallelism and connectivity of the biological system. Finally, neural systems display levels of fault-tolerance and adaptive learning that artificial systems have yet to approach.

1.4 Scoping the Problem

The scale of the problem of modeling the human brain has been scoped by, among others, Mead [30]. The hundred billion neurons have of the order of 10^{15} connections, each coupling an action potential at a mean rate of not more than a few hertz. This amounts to a total computational rate of around 10^{16} complex operations per second. No computer has yet been built that can deliver this performance in real-time, though this gap will be closed in the near future. Current supercomputer developments are aimed at delivering petaFLOP (10^{15} floating-point operations per second) performance levels, perhaps only one order of magnitude short of the performance required to model the human brain.

Perhaps more challenging is the issue of power efficiency. Moore observed that the number of transistors on a micro-chip doubled every year, and he predicted that this trend would continue for another 10 years [32]. The doubling period was subsequently modified to 18 months, but has continued at this rate to this day and is expected to continue for at least another decade. The observation is widely referred to as ‘Moore’s Law’. In fact, Moore’s Law ceased to be merely an observation a long time ago, becoming instead a semiconductor industry boardroom planning tool; therefore, it is now a self-fulfilling prophecy for as long as physics, engineering and industry economics allow. Although Moore’s paper referred only to the increasing number of transistors on a microchip, the process of component miniaturization that makes this possible has also led to the spectacular improvements in performance, power efficiency and cost-per-function that underpins the pervasive digital technology that we enjoy today.

Mead argued that current computer technology will still be 10 million times less power efficient than biology even when Moore's Law has run its full course, so a real-time digital model of the brain will consume tens of megawatts of power. He argued that analogue electronics can close much of that gap, and he has built several silicon implementations of analogue neural systems that support this argument [29]. The very high power efficiency of the biological system has been emphasized more recently by Laughlin and Sejnowski [23], and presents a real challenge to the computer engineer to find ways to build artificial systems that come anywhere close to matching it.

Delivering the necessary level of computational power is a pre-requisite to building a real-time model of the human brain, but it is far from being the only problem facing researchers in this area. One daunting challenge is the need to obtain the neural *netlist*, a term that we borrow here from electronic engineering where it is used to refer to a formalized description of an electronic circuit in terms of the type and parameters of each of the components (here neurons) and their connectivity patterns. A second challenge is to understand the developmental aspects of the brain's structure: the netlist is not static, but neurons grow and die and their interconnections extend and contract in response to activity and other biological factors. Thirdly, a brain needs a sensory system to provide its inputs and actuators to respond to its outputs; it needs to be embodied in some way in order to have a purpose for its activities. Fourthly, embodied brains do not live in isolation, they form societies and cultures that define and constrain their actions.

1.5 The Research Agenda

We use the term *neural systems engineering* to describe the constructionist approach to exploring the potential of the neuron as a component in an information processing system (in the widest sense of this term).

This approach could be pursued entirely through software modeling; though because the computational demands of large-scale neural modeling are high, there have been many projects where special-purpose hardware has been constructed to accelerate the computation. Building the system in hardware also ensures that real-world issues such as noise are addressed. The special-purpose hardware may be aimed at modeling the low-level details of the neuronal processes in analogue electronic circuitry (an approach known as *neuromorphic computing*), or at the other extreme, building massively parallel digital supercomputers with special features to support neural modeling.

The key issues at present are the following:

- To identify the simplest models that capture the information processing functions of a neuron. Neurons are very complex cells, but how much of this complexity is functionally relevant and how much is an artefact of the cell's evolutionary heritage, its need to grow, find energy, self-repair, and so on?

- To represent the heterogeneous nature of biological neural systems. There are many different types of neurons in the brain, and each instance of a particular type has unique parameters.
- To identify and implement the necessary connectivity patterns of the natural system.
- To identify the neural ‘codes’ whereby populations of neurons represent complex information and through which the continuous sensory inputs can influence discrete actions and decisions.
- To identify the mechanisms of neural adaptation that enable the system to self-organize and learn, continually tuning and optimizing its performance.

Ultimately, it would be extremely useful to be able to raise the level of abstraction at which neural networks are modelled. If, for example, the functionality of the cortical micro-column could be encapsulated in a set of mathematical equations, then the computational (and consequently power) demands of modeling the brain might come down by one or two orders of magnitude.

Among all these is the hope that some understanding will be gained of the emergent properties of complex dynamical systems, together with some insights into the fault-tolerant capabilities of biological systems and how these capabilities might be better emulated by engineered systems.

1.6 Chapter Structure

In Sect. 2, we look at the basic principles at work in neural computation. In Sect. 3, we look in detail at the problem we are addressing – what is known about the neuron’s function and connectivity in its role as a component in complex biological systems – and we look at some of the models that are used to capture its function. In Sect. 4, we discuss the issues that arise in constructing large-scale artificial neural systems, and in Sect. 5, we look at how these issues have been addressed by various teams around the world, including our own work in this area. Section 6 concludes the paper with some speculation about the prospects for progress and potential breakthroughs in our understanding of brain function and our capability for engineering more intelligent systems in the future.

2 Neural Computation

In this Section, we begin to look at the neuron as an information processing device. Any computational system must achieve a balance between its *processing*, *storage*, and *communication* functions. It is useful to consider how these three functions are achieved in neural systems.

In discussing neurons, it is useful to use some biological terminology albeit, perhaps, with an engineer’s interpretation of what this terminology

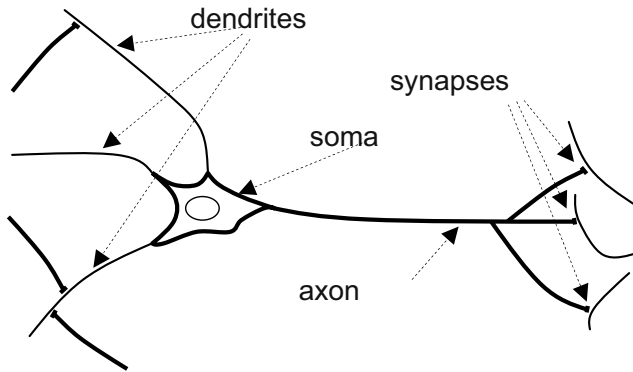


Fig. 2. The four primary structures of a neuron. Inputs are collected via the dendrites and passed to the soma, the main body of the cell. The action potential (spike) generated in the soma propagates along the axon, where it passes through synapses to the dendrites of other neurons

signifies. A neuron may be viewed as comprising the following four structures (see Fig. 2):

- *Dendrites* are the tree-like structures that gather the inputs to the neuron from other neurons or sensory inputs and couple them to the soma.
- The *soma* is the central body of the neuron where the inputs are processed and the output is generated.
- The *axon* carries the output of the neuron through another tree-like structure to couple it to other neurons or physical actuators, incurring a signal-propagation delay that depends on the length of the axon.
- *Synapses* form the coupling between neurons. These can develop wherever the axon from one neuron is physically proximate to a dendrite of another. The coupling process incurs some time delay, but this can generally be added into the axonal delay for modeling purposes.

The synapse is the primary location of adaptation in the neural system: the strength of the coupling between two neurons self-adjusts over time in response to factors such as the correlation between the activities of the two neurons that are coupled through the synapse.

We can now look at how these structures contribute to the three aspects of computation that must be kept in balance: processing, communication and storage of information.

2.1 Processing

The processing function is performed within the neuron. The inputs are combined in the dendrites in some way and passed to the soma which produces output events in response to input events through a non-linear transfer

function, which we will model using suitable differential equations whose complexity is limited only by the available computing power. In some models the dendrites simply sum the inputs, whereas in others they interact in more complex ways.

2.2 Communication

Communication in neural systems is predominantly through the propagation of spike ‘events’ from one neuron to the next. The output from the neuron’s body (its soma) passes along its axon which conveys the spike to its many target synapses. Each synapse uses chemical processes to couple the spike to the input network (the dendritic tree) of another neuron.

Since the spike carries no information in its shape or size, the only information conveyed is in which neuron fired and when it fired.

2.3 Storage

It is in the storage of information that the neuron’s story becomes most complex. There are many processes that can be seen as storing information, some operating over short time-scales and some very long-term. Examples of these processes are as follows:

- the neural dynamics include multiple time constants, each of which serves to preserve input information for some period of time;
- the dynamical state of the network may preserve information for some time;
- the axons carry spikes at low speeds and therefore act as delay lines, storing information as it propagates for up to 20 ms; and
- the coupling strength of a synapse is, in many cases, adaptive, with different time constants applying to different synapses.

In a neural modeling system, we expect the model to capture the neural and network dynamics, and hence the contributions these mechanisms make to information storage. The axon delay-line storage does not come so easily as the high speeds of electronic signalling make spike communication effectively instantaneous. It is likely that the axon delay is functionally important, for example, enabling networks to learn complex spatio-temporal patterns as exhibited in polychronization [19], so we must put these delays back in, either by delaying the issue of the spike or by delaying its effect at the destination.

The primary long-term storage mechanism is synaptic modification (within which we include the growth of new synapses). This is the most fundamental storage mechanism; here, we require long-term stability and support for a range of adaptive mechanisms.

3 The Neuron as a Component

We take the neuron to be a device that, like a logic gate, has several inputs and a single output. The number of inputs will, however, typically be in the thousands rather than the two or three inputs that a logic gate normally has. The general scheme of the component neuron is illustrated in Fig. 3, which may be compared with Fig. 2 to see how it captures the major functional components of the biological neuron.

3.1 Communicating with Spikes

Although for most of its history the field of artificial neural networks has taken the output of a neuron to be a real value that varies from one discrete time-step to the next in a highly synchronous way, we will take the more biologically realistic model of the output as a time-series of action potentials (spikes) which, since the form of the spike is largely invariant, can be viewed as a time-series of asynchronous events. The output of neuron i is then simply

$$y_i = \sum_n \delta(t - t_{i,n}) \tag{1}$$

where $\delta(t)$ is the Dirac delta function representing a unit impulse at time t and $t_{i,n}, n = 0, N_i$, are the times of the spikes on neuron i .

In this model, information is conveyed solely in the times at which the events occur, perhaps in the rate of spiking, but there are other possibilities such as the relative timing of spikes from different neurons.

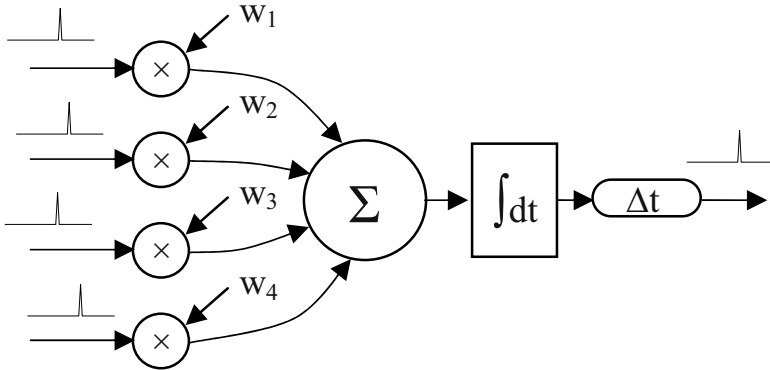


Fig. 3. The neuron as a component. In the leaky integrate-and-fire model, input spikes are multiplied by their respective synaptic weights, summed, and integrated over time. If the integral exceeds a threshold, the neuron fires and the integration restarts. The output spike may be delayed to model the propagation time along the axon

Although spikes appear to represent the primary means of propagating information in mammalian brains, they are clearly not the only means. Specialized neurons cause the emission of chemicals that have a broad effect on adjacent regions of neurons, and these are likely to be important in learning mechanisms. There is also evidence for direct analogue information exchange between neurons whose dendritic trees make contact.

We will proceed on the assumption that Eqn. (1) captures the most important neural information exchange process, but remain aware that there may be other important processes in addition to spiking communication.

3.2 Point-Neuron Models

There are many different models that describe the internal operation of a neuron at different levels of detail. The simplest of these are point-neuron models, which ignore the spatial characteristics of the neuron. The inputs are combined through a weighted summing process to give a single driving force

$$I_i = \sum_j w_{ij} y_j \quad (2)$$

where w_{ij} represents the strength of the synapse coupling neuron j into neuron i , and the sum is taken over all of the inputs y_j to neuron i .

This driving force is then applied to some form of non-linear function to generate the output spikes from the neuron. A simple first-order differential equation is used in the *leaky integrate-and-fire* (LIF) model,

$$\dot{A}_i = I_i - \frac{A_i}{\tau} \quad (3)$$

$$\text{if } A_i \geq \Theta, \text{ fire neuron } i \text{ and reset } A_i = 0 \quad (4)$$

Here, the activation A_i of neuron i decays to its zero rest state with time constant τ . It is increased by an amount w_{ij} every time input neuron j fires, and if at any time it exceeds the threshold Θ it fires and its activation is reset to zero.

The LIF model captures some of the essential behavior of a biological neuron, but is often adapted to greater realism through the addition of features, such as:

- *Habituation.* When presented with a step function in its input stimulus, a biological neuron tends to fire rapidly for a short period but does not sustain this firing rate for long, whereas the LIF model will fire at a steady high rate. Habituation can be added to the LIF model by making the threshold a leaky integrator of the neuron's own output,

$$\dot{\Theta}_i = y_i - \frac{(\Theta_i - \Theta_0)}{\tau_\Theta} \quad (5)$$

so each spike from the neuron increases its threshold above its rest state Θ_0 but this effect decays with time constant τ_Θ .

- *Refractory period.* Immediately after a neuron fires, it is insensitive to further inputs. The LIF neuron can be extended to model this in a number of ways, including simply causing it to ignore inputs for a fixed period after a spike, or resetting its activation after firing (Eqn. (4)) to a negative level.

Adding these ‘bells and whistles’ to the LIF model increases both its biological accuracy and computational complexity. The tradeoff between accuracy and computational complexity is a recurring theme in large-scale neural modeling.

3.3 The Spike Response Model

The spike response model generalizes the LIF model and can describe the behavior of any neuron that responds linearly to its inputs [13]. Each input causes a perturbation to the neuron’s potential, which follows a characteristic course over time (which may depend on when this neuron last spiked). This can be captured by a kernel function $\eta(t)$, and the activation potential of the neuron is then formed from a linear sum of these kernel functions, each scaled by its respective synaptic weight.

Similarly, when this neuron fires, its potential follows a characteristic time course that can be represented by another kernel function $\nu(t)$,

$$A_i(t) = \nu(t - t_{i,N_i}) + \sum_j w_{ij} \sum_n \eta(t - t_{j,n}) \quad (6)$$

As with the LIF model, the neuron fires when the threshold is reached, and the threshold can be dynamic to model habituation. For computational efficiency, the kernel functions can be stored as look-up tables.

3.4 The Izhikevich Model

A rather different approach to the primary function of a point-neuron model is offered by Izhikevich [17]. His approach is to observe that the biological mechanism that gives rise to the neuron’s spike output must have its basis in an instability in the electrochemical process that generates and sustains the spike as it propagates along the axon. He therefore turns to the mathematics of bifurcating processes to identify equations that capture the nature of this bifurcation at the lowest computational cost.

His investigations yielded the following pair of coupled differential equations:

$$\dot{\nu} = 0.04\nu^2 + 5\nu + 140 - u + I, \quad (7)$$

$$\dot{u} = a(b\nu - u), \quad (8)$$

$$\text{if } \nu \geq 30 \text{ then } \nu = c; u = u + d \quad (9)$$

where ν is a ‘fast’ variable corresponding to the neuron’s activation (A in Eqn. (3), scaled to correspond to the activation of a biological neuron measured in millivolts); u is a ‘slow’ variable that adapts the neuron’s dynamics over time; I is the weighted sum of inputs as in Eqn. (2); and a, b, c and d are parameters that define the characteristic spiking patterns the neuron produces.

The test for ν reaching 30 mV in Eqn. (9) is not a firing threshold, as was the test in Eqn. (4), since Eqns. (7) and (8) generate the spike directly. This test is detecting the peak of the spike itself.

An example of the behavior of these equations is shown in Fig. 4, where the input I undergoes a step-function change at time 20 ms. The neuron spikes repeatedly, twice in quick succession and then at a slower, stable rate, displaying habituation. This figure was produced using a 10-bit fixed-point implementation of the equations, showing that relatively simple digital hardware is sufficient for this purpose.

The Izhikevich model is comparable in terms of computational complexity with the LIF model when the latter includes habituation and a refractory period, but it is able to model a much wider range of neural behaviors. As such, it is a very promising basis for large-scale neural modeling at the point-neuron level of complexity. Izhikevich has himself used it for simulating cortical activity with very large numbers of neurons [18].

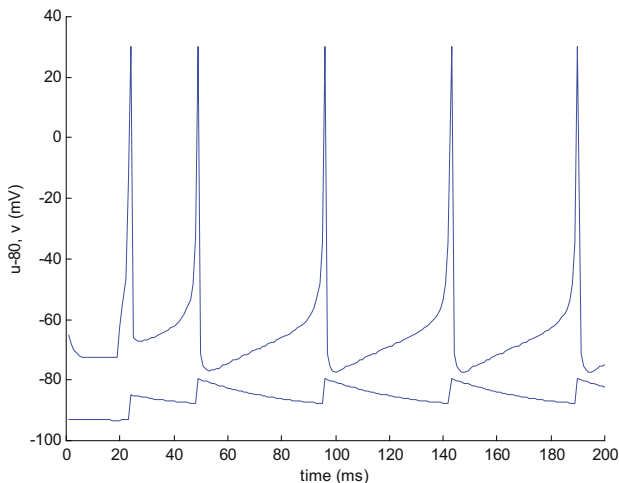


Fig. 4. A solution to Izhikevich’s equations, driven by an input step function at 20 ms. The slow variable, u (lower curve), has been offset by -80 mV for clarity (this solution was obtained using a 10-bit fixed-point approximation to the equations, but is very close to the real-valued solution)

3.5 Axons: The Hodgkin-Huxley Model

Ground-breaking experiments on the giant axon of the squid (chosen because its size minimized the still-considerable experimental difficulties) culminated in the publication in 1952 of a seminal paper that presented equations describing the electrical behavior of a nerve fibre [16].

The equation for the axon membrane potential V is:

$$C_m \dot{V} = -g_L(V - V_L) - \bar{g}_{Na} m^3 h (V - V_{Na}) - \bar{g}_K n^4 (V - V_K) \quad (10)$$

where C_m is the membrane capacitance per unit area; g_L , $\bar{g}_{Na} m^3 h$, and $\bar{g}_K n^4$ are the conductances per unit area of the membrane for various ions that are involved in the axon processes; and V_L , V_{Na} , and V_K are the associated equilibrium potentials. The dimensionless quantities m , h and n represent voltage-dependent channel variables which obey equations of the form:

$$\dot{c} = \alpha(V)(1 - c) - \beta(V)c \quad (11)$$

where each channel variable has different $\alpha(V)$ and $\beta(V)$ functions that involve negative exponentials of the membrane voltage V .

The Hodgkin-Huxley equations offer a very detailed model of the propagation of action potentials along neuronal fibres, but they are computationally demanding. Since the solution is characterized by a sharp transition between a continuous fluctuation for a weak input and a distinct action potential for a stronger input, spiking behavior is often taken to be a good approximation, and the axon process is modelled as a simple delay, or possibly multiple delays to allow for the different lengths of axon to different target synapses.

3.6 Dendritic Trees and Compartmental Models

The dendritic networks that capture the inputs to a neuron are complex trees and may include nonlinear interactions between inputs on different branches. Point-neurons generally ignore such effects and simply sum all the inputs. A more accurate model is to view the dendrites as similar to electrical cables and use transmission line models to compute the dynamics of input propagation towards the soma. As the dendritic trees are non-uniform and include branch points, ‘compartmental’ models split the trees into small cylindrical sections where each section has a uniform physical property [6]. These models can be simple or highly detailed and can incorporate non-linearities for greater accuracy.

As with the Hodgkin-Huxley model, great accuracy is achievable at the cost of considerable computational effort.

3.7 The Synapse

The functional effect of a synapse is to transfer a spike from the axon at its input to the dendrite at its output. The biological synapse does this by releasing a number of packets of chemicals across the synaptic gap in response to the incoming spike, and these packets then affect the membrane properties of the receiving dendrite. The effect is quantized [8] and probabilistic in nature. It is often approximated by a multiplicative ‘weight’ as in Eqn. (2).

Of much greater subtlety is the adaptive nature of the synapse. The ability of the synapse to adjust its effectiveness as a connection (in which we include the ability of the neuron to grow new connections) is believed to be the major long-term memory mechanism in the brain. Precisely how and when these changes take place is not fully understood, but there are a number of theories and explanations on offer.

Hebb postulated that when one neuron was close to another and repeatedly played a causal role in its firing, the coupling between them would strengthen [14]. This postulate has since been validated experimentally, and the term ‘Hebbian learning’ is widely applied to mechanisms that modify the strength of a synapse as a result of correlations of various sorts between the spiking patterns of the two neurons that it connects. Long-term potentiation (LTP) is a term used to describe the most direct experimental confirmation of Hebb’s principle [25]. The opposite effect has also been observed in some areas of the brain: long-term depression (LTD), also known as ‘anti-Hebbian learning’, which describes a circumstance where correlations between two neurons result in a weakening of the synaptic strength.

Investigations into the detailed mechanisms of LTP and LTD have led to the observation of spike-time-dependent plasticity (STDP), where the scale of the synaptic modification has a well-defined dependency on the precise relative timing of the spikes from the input and output neurons, including changing sign if the order is not consistent with causality. Quite subtle models of the biophysical processes involved in STDP have been developed [39].

STDP is unlikely to be the whole story, however. There are reward mechanisms in the brain that release chemicals that may modulate synaptic plasticity in some way, and mechanisms that lead to the growth of new connections (where causality cannot be involved, since before the growth there was no causal connection).

4 Engineering Neural Systems

Now that we appreciate the behavior of an individual neuron as a component and have a choice of models that we can use to represent its functionality, we can begin to consider the construction of systems of neurons. A number of

questions arise in the development of any system of neurons, which reflect the issues listed previously in Sect. 1.5:

- At what level of detail should each neuron be modelled?
- How do populations of neurons jointly encode information?
- How is the connectivity of the network determined?
- How is the connectivity of the network implemented?
- How does the network learn, adapt or tune itself?
- How is the network embodied and the body placed into an environment with which it interacts?

We will address each of these issues in the following Sections, and then offer some examples of neural systems to illustrate how everything can come together.

4.1 Neural Models

Much of the past work on building artificial neural networks has adopted the rate-coding view of neural output, where the only significant information that a neuron conveys about its inputs is in its firing rate [1]. The firing rate can be represented by a real-valued variable, and the spiking behavior of the biological neuron is abstracted entirely into this variable. Eqn. (2) is still used to compute the neuron's activation, but the variables y_j are real valued and no longer a time-series of spikes. The activation is modulated by a nonlinear transfer function to produce the real-valued output. The nonlinear transfer function is often a sigmoid function, which gives a smooth differentiable transition from 0 to 1 as the activation increases from its minimum to its maximum value.

In the most abstract models, the nonlinear transfer function is a simple threshold and all neural states are represented by binary values: 0 when the neuron is not firing and 1 when it is firing [28].

Recently, there has been a return to interest in more biologically accurate models that incorporate spiking behavior. All of the models described in Sect. 3 have been explored, and others too. There is no consensus on the correct level of trade-off between complexity and biological realism, and none is likely to arise until many more questions have been answered about which features of the biological component are essential to its information processing function and which are simply artefacts of its biological origins.

4.2 Population Encoding

The representation of sensory information by populations of neurons has been the subject of study. Eliasmith and Anderson offer detailed analytical tools for understanding how the combined spike firing rates of a heterogeneous population of neurons can represent a continuous physical parameter with

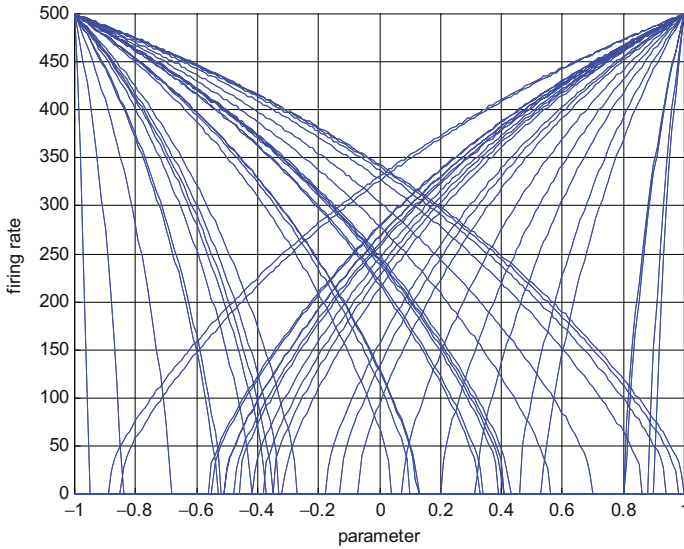


Fig. 5. Representation of a physical parameter (x -axis) by the spike firing rates (y -axis) of a heterogeneous population of 50 neurons. Each curve shows the firing rate of one neuron (after [7])

arbitrary accuracy, and how successive layers of such populations can perform computations on those parameters [7].

An illustration of the representation of a continuously variable physical parameter is shown in Fig. 5. Here, a population of 50 LIF neurons, each with randomly selected sensitivity and offset (but all having the same maximum firing rate), together give an accurate estimate of the parameter even in the presence of noise. The error in the estimate is inversely proportional to the square root of the population size.

In these population codes, each neuron is firing independently and the information is encoded across the individual firing rates of the neurons that form the population. This is not the only way in which a population of neurons can represent information, but the other encoding mechanisms described below all require that it is not only the firing rate of the neuron that is significant but also the timing of the individual spikes. Evidence that precise spike timing is important in biological systems is sparse, but one can point to O’Keefe and Recce’s work on rat hippocampal place cells, where individual neurons fire in a well-defined phase relationship to the theta wave cycle [35].

Van Rullen and Thorpe take observations of the speed with which humans can respond to visual images, which does not allow enough time for any individual neuron to emit more than one spike, as evidence that individual firing rates are an insufficient explanation of the performance of the system [44].

They have postulated rank-order codes as an alternative description. With rank-order codes, a population of neurons – in this case the ganglion cells in the retina – spike in an order determined by their tuning to the current sensory stimulus. The information is carried in the order of firing of the neurons in the population.

It is not necessary for the entire population to fire in a rank-order code, in which case information can be conveyed both in the order of firing of the subset that does fire and in the choice of that subset.

In a final simplifying step, it is possible to use just the subset choice to convey information and to ignore the order of firing altogether. In this case, the result is an N -of- M code, where all of the information is conveyed in the choice of the N neurons that fire from a total population of M neurons. Time has now been abstracted out of the model, and the system can be described and analysed in terms of pure binary patterns.

4.3 Spatio-Temporal Spike Neurons

While unordered N -of- M codes are purely spatial and rely on approximate spike synchrony across the active sub-population, rank-order codes represent a first step towards exploiting the temporal properties of spiking patterns to convey information. This can be taken further and generalized to polychronization [19]. Here, the ability of an individual neuron to detect coincident inputs is combined with the intrinsic delays in the axons of the neurons that connect into its synapses to tune the neuron to respond to a very particular spatio-temporal pattern of activity on those input neurons. These spatio-temporal patterns can be very difficult to identify in neural firing traces, but have considerable information-bearing capacity and may play an important role in biological neural systems.

4.4 Defining ‘Connectivity’

Abstract neural networks have a connectivity that can be defined algorithmically. Full connectivity is commonly used, where every neuron in one layer connects to every neuron in the next layer. In recursive networks, every neuron may connect to all of its peers in the same layer. Such networks have readily defined connectivity and are generally conceived in terms of this connectivity, which often has little to do with biological realism.

The availability of accurate, detailed network connectivity data for biological neural systems is fundamental to the task of building computer models of biologically realistic neural networks. Techniques for producing this connectivity information are improving, and recently Binzegger et al. have developed an approach based upon a statistical analysis of the proximity of the dendritic and axonal processes of different neuron populations in a three-dimensional

reconstruction of the cat neocortex [4]. This approach leads to highly detailed connectivity data that have been used to build computer models of large-scale biological neural systems (for example, [18]).

4.5 Implementing Connectivity

Biology employs massive numbers of low-speed channels to communicate neural *spike* events from their sources to their destinations. Despite the major advances in the density of microelectronic components, artificial neural systems cannot approach the physical connectivity of natural systems.

However, electronic systems do have one advantage here: electronic communication is about five orders of magnitude faster than biology. An axon may carry tens to hundreds of spikes per second; a computer bus can operate at tens of MHz [5]. Therefore, it is reasonable to seek ways to multiplex events from many neurons along the same bus or channel. A neural spike is an asynchronous event which carries information only in its timing, so the multiplexed information need simply identify the neuron that has fired. This led Sivilotti [42] and Mahowald [26] to propose the *address-event representation* of spikes, where each neuron in a system is given a unique number (address), and when the neuron fires this number is propagated through the interconnect to (at least) all neurons to which the firing neuron is connected.

A problem with multiplexing multiple asynchronous events through the same channel is that of collisions; when two events coincide closely in time, either one event must be dropped or they must be serialized, incurring some timing error. Boahen argues that asynchronous arbitration can be used to serialize the events with low timing error, even when 95% of the channel capacity is being used, and this approach scales well to faster technologies [5].

4.6 Learning, Adapting, and Tuning

A key feature of any neural network, biological or engineered, is its ability to (i) learn new responses, (ii) adapt to new stimuli, and (iii) tune itself to improve its performance at the task in hand. These processes are generally achieved through the adjustment of synaptic weights in accordance with some sort of *learning rule*.

The long-standing way of optimizing artificial neural networks to a particular task is through the use of error back-propagation [45]. Error back-propagation compares the outputs of a neural network with the desired output and then reduces the error in the output by adjusting weights and propagating errors backwards through the entire network. There are two problems with this approach for biological or large-scale engineered systems: (i) it assumes that the desired output state is known and (ii) it assumes the existence of an agent external to the system with global control of it. Neither of these is

generally true for the systems of interest here. It is worth noting, however, that with suitable network topologies and local learning rules, biologically plausible systems have been shown to operate in ways that effectively deliver error back-propagation [36].

We will look for local learning rules that are based on Hebbian principles, adjusting weights according to local spike activity along the lines of STDP. In some cases this can be reduced to a simple rule, as in the case of the binary associative memories described in Sect. 4.7. In more general applications this remains an uncertain aspect of the system engineering, where new insights and approaches are likely to lead to significant progress.

4.7 Example Neural Systems

Many applied neural systems are based on abstract neural models and do not depend directly on spike generation or communication. Examples that demonstrate principles of operation that could be employed in spiking systems with local learning rules include Willshaw et al.'s non-holographic memory [46], and its close relative the correlation matrix memory (CMM) [22]. CMMs employ binary output neurons with binary synaptic weights and often use N -of- M population codes. They have proved very effective for building large-scale associative search systems such as the AURA system at the University of York [2], which has found a wide range of industrial applications.

Similar abstract models are employed in sparse distributed memories (SDMs) [21]. Although Kanerva's original SDM was not directly compatible with standard spiking neuron models, our variants of Kanerva's SDM employing N -of- M codes [9] or rank-order codes [12] can readily be implemented with such models.

4.8 Neuromorphic Systems

An approach to engineering artificial neural systems that have been explored in certain application domains is to implement neural models in analogue hardware. Some functions such as multiplication are very much cheaper and less power hungry when implemented in analogue rather than digital electronics, and analogue systems also offer intriguing nonlinearities that offer elegant solutions to certain tricky aspects of neural modeling.

The analogue approach has been applied to vision systems [24, 29, 47] and similar early-stage sensory input processing. The combination of analogue neural models with digital spike communications models the biological solution closely and is probably the most promising microelectronic approach to building large-scale neural networks in the long term. In the shorter term, the uncertainties over the optimal neural model make the inflexibility of the analogue implementation (compared to a programmable digital system) unattractive for the general-purpose neural processor.

5 Large-Scale Projects

It seems reasonable to assume that some of the most challenging and interesting aspects of neural function will be manifest only in systems of considerable scale where, for example, there is scope for very high-dimensional representation and processing of sensory input. This makes the construction of large-scale systems an important component of neural systems engineering research. There have been several projects in recent times aimed at building large-scale computer models of neural systems, using different complexities of model and different techniques to accelerate the computation:

- Software systems run on conventional machines are highly flexible but can be slow, unless run on very powerful computers such as **Blue Brain** [27]. Izhikevich ran a simulation of one second of activity in a thalamocortical model comprising 10^{11} neurons and 10^{15} synapses, a scale comparable with the human brain [18]. The simulation took 50 days on a 27-processor Beowulf cluster machine.
- Field-programmable gate arrays (FPGAs) have the potential to accelerate key software routines [48], though it can be difficult to get the correct system balance between processing and memory. FPGAs are very flexible but somewhat harder to program than software, and high-performance computer manufacturers are very interested in the possibility of integrating FPGAs into their parallel machines. This may result in software tools that deliver the acceleration transparently to the programmer.
- Building bespoke hardware to support neural modeling is an approach that has been tried from time to time with limited lasting success. The fundamental problem is the same as for other areas of application-specific hardware - the commercial momentum behind the progress of the general-purpose computer renders any benefit of special-purpose hardware hard-won and short-lived. In the neural modeling area there is also the issue of deciding how much the neural model should be cast into hardware, optimizing performance but losing flexibility, against making the system as soft and general-purpose as possible.

Here, we give brief descriptions of three projects aimed at large-scale neural modeling: **Blue Brain**, at EPFL, Switzerland; **SPINN**, at the Technical University of Berlin; and our own plans for the **Spinnaker** machine.

5.1 Blue Brain

By far, the largest-scale project aimed at building computer simulations of sections of the brain is the **Blue Brain** project at EPFL in Switzerland [27]. This work is based upon one of the worlds most powerful supercomputers, the IBM Blue Gene/L. This machine delivers up to 360 teraFLOPS of computing power from 8192 PowerPC CPUs each running at 700 MHz and arranged in

a toroidal mesh. Alongside the IBM supercomputer is a sophisticated stereo visualization system based upon SGI graphics computers.

The **Blue Brain** project simulates biological neural networks using detailed compartmental neuron models and aims to deliver biologically accurate models of neural microcircuits such as the neocortical microcolumn. The computations are based upon the Hodgkin and Huxley [16] equations and Rall's [38] cable models of the dendritic and axonal trees, and use the **NEURON** [34] simulator codes extended to use a message-passing interface to communicate action potentials between neurons modelled on different processors.

In addition to exploiting their computational resources, the **Blue Brain** team is also assembling a major database of biological neural data upon which to base their computer models.

The **Blue Brain** project as currently configured has a machine capable of simulating up to 100,000 very complex neurons or 100 million simple neurons. The emphasis is on maximally accurate models of biological neural systems.

5.2 SPINN

The Spiking Neural Network activity at the Technical University of Berlin has yielded a series of hardware systems for the acceleration of neural network modeling over more than a decade of related research projects: **BIONIC**, **NESPINN**, **MASPINN**, **SP²INN** and, most recently, **SPINN** emulation engine (**SEE**).

The Biological Neuron IC (**BIONIC**) project yielded a chip capable of modeling up to 16 neurons each with 16 synapses [37]. The **NESPINN** (Neuro-computer for Spiking Neural Networks) project aimed to model 16,000 neurons each with 83 synapses [20], with the possibility of handling larger systems with multiple accelerator boards.

The Memory Optimized Accelerator for Spiking Neural Networks (**MASPINN** [40]) was a hardware accelerator that connects to a host PC via a standard PCI bus. The neural model treated the dendritic tree as a set of independent leaky integrators, each receiving a number of inputs. The outputs of these integrators then interact in a programmable way to form the driving current for a soma model with a dynamic threshold (again generated by a leaky integrator). Axonal delays are modelled at the neuron's output, so a neuron that connects with different delays to other neurons has multiple outputs, one for each delay value. **MASPINN** employs a number of optimizations to reduce the computational demands. It caches synaptic weights that are used heavily, and it tags inactive components so that they do not consume resource computing the leaky integrator function. The **MASPINN** project aimed to simulate a million relatively simple neurons and had a specific application area (image processing) as its target.

The Synaptic Plasticity in Spiking Neural Networks (SP²INN) project [31] aimed at building hardware to model a million neurons with several million synaptic connections but, at the end of the paper, the authors contemplate the difficulties of designing special-purpose hardware to compete with the relentless advances in the performance of general-purpose computers.

The SEE project abandons custom hardware in favour of FPGAs and exploits the embedded general-purpose processing power incorporated in some of today's FPGA architectures [15]. The system can model half a million neurons each with 1,500 synaptic connections.

Taken together, these projects represent a considerable body of experience in designing hardware to support spiking neural network modeling, and it is instructive to see how each project has built on the ideas displayed in its predecessors but how little the hardware components, presumably designed with considerable effort, have carried forward. This illustrates the difficulty of making bespoke hardware flexible enough to solve more than the problem of the moment.

5.3 SpiNNaker

The SpiNNaker project at the University of Manchester [11] has as its goal the development of a massively parallel computer based on chip multiprocessor technology and a self-timed Network-on-Chip (NoC) communications system [3]. The system is aimed at modeling large-scale systems of up to a billion spiking neurons in real-time and is optimized for point-neurons such as the LIF and Izhikevich models. It is not intended to run models with high biological accuracy, but is much more aimed at exploring the potential of the spiking neuron as a component from which useful systems may be engineered. Biological data are taken as a very useful source of inspiration, but not as a constraint, and useful ideas for novel computation systems will be seen as a positive outcome irrespective of their biological relevance.

The philosophy behind the system architecture is based on the observation that modeling large systems of spiking neurons falls into the 'embarrassingly parallel' class of applications, where the problem can be split into as many independent processing tasks as is useful. The performance of an individual processor in the system is not an important parameter. What matters is the cost-effectiveness of the implementation, which can be broken down into the capital cost and the running cost, which can be assessed, respectively, in terms of:

- MIPS (millions of instructions per second) per mm²: how much processing power can we get on a given area of silicon? and,
- MIPS per watt: how energy-efficiently can this processing power be delivered?

The choice for this system is between employing a small number of high-end processors or a larger number of lower-performance embedded processors on each chip. The performance density (MIPS per mm^2) of both classes of microprocessor is similar, but the embedded processors are an order of magnitude more energy efficient. Hence, the decision is to use embedded processors in large numbers, and the **SpiNNaker** chip will incorporate up to 20 ARM processor cores to execute the neural modeling code.

The organization of the **SpiNNaker** multiprocessor chip is illustrated in Fig. 6. One of the processors on the chip is selected to act as monitor processor, and runs the operating system functions on the chip. The remaining processors act as fascicle processors, each modeling a group of up to a thousand individual neurons where that group is selected to have as much commonality as possible in the neurons that connect into its member neurons and the neurons elsewhere that its member neurons connect to.

Each fascicle processor receives spike events from, and issues spike events into, a packet-switching communications system, with each spike event encoded as a single packet. Within a chip, these spike events converge through the *Communications NoC* to an arbiter, where they are selected and sent in sequence to a router [10]. The router uses internal tables to identify which fascicle processors should receive each event (which can be determined from the connectivity netlist of the neural network that is being modelled) and passes the event on accordingly; this mapping may include none, one or several of the fascicle processors, so a full multi-cast routing mechanism is required that is based on address-event communication as described in Sect. 4.5.

The Communications NoC also extends between chips, so the system can be enlarged by connecting multiple chips together, as illustrated in Fig. 7. Each chip has six transmit interfaces ('Tx i/f' in Fig. 6) and six receive interfaces ('Rx i/f') that effectively extend the Communications NoC to six neighbouring chips through bi-directional links. The Router can direct packets from any on- or off-chip source to any on- or off-chip destination, and has sufficient capacity to support systems comprising very large numbers (up to tens of thousands) of chips.

5.4 Virtual Communication

The **SpiNNaker** architecture illustrates one of the important principles of engineering large-scale neural modeling systems: the need to decouple the physical organization of the engineered system from the physical organization of the biological system it is designed to model.

Biological neural systems develop in three dimensions, though the way the three-dimensional space is used is quite variable. The cortex, for example, is often described as a thin sheet of neurons, where the third dimension is used largely for long-range connections and to enable the large two-dimensional

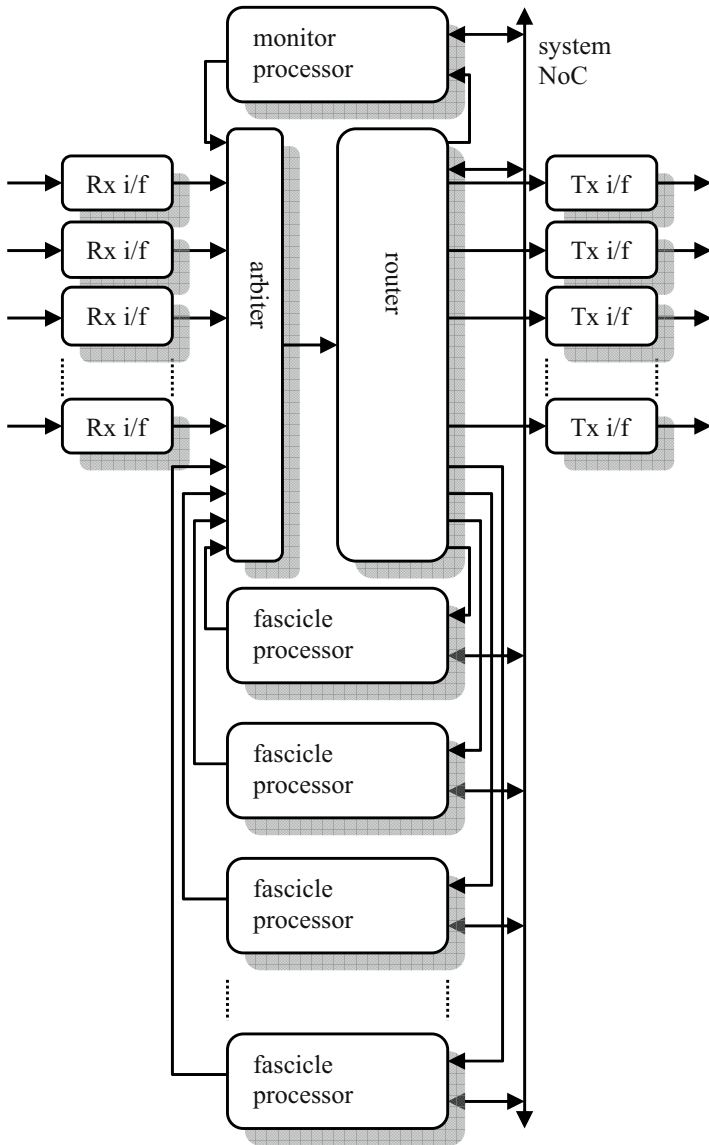


Fig. 6. Organization of a SpiNNaker chip multiprocessor node, illustrating the *Communications Network-on-Chip* (NoC) that is used to carry spike event packets around the system. Each *fascicle processor* models many neurons. Packets from other nodes arrive through the receiver interfaces ('Rx i/f') and are merged with packets issued by the fascicle processors into a sequential stream by the arbiter. Each packet is then routed to one or several destinations, which may include other processing nodes (via the transmit interfaces 'Tx i/f') and/or local fascicle processors. The *monitor processor* carries out operating system functions and provides visibility to the user of on-chip activity

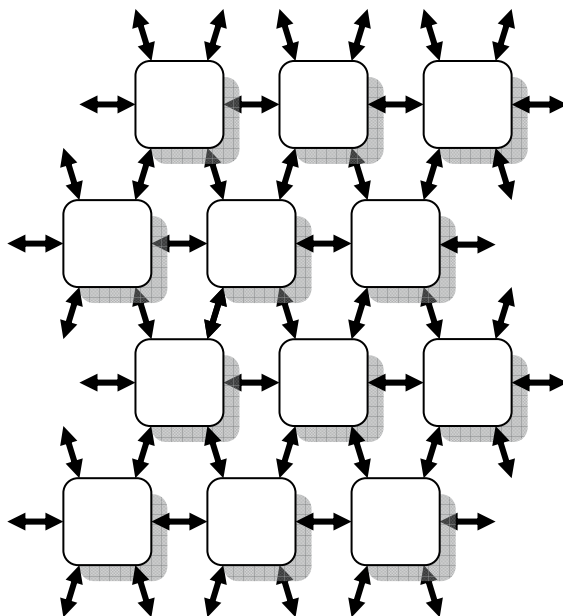


Fig. 7. SpiNNaker system architecture. Each of the chip multiprocessor nodes is connected to its six nearest neighbours by bi-directional links. The left and right sides of the mesh are connected, as are the top and bottom edges, to form a two-dimensional toroidal surface

area to be folded into a convoluted shape in order to fit into a small three-dimensional volume. On a small scale, the sheet does have a thickness which is divided into a characteristic six-layer structure.

The SpiNNaker system architecture, as illustrated in Fig. 7, has a strongly two-dimensional structure. However, this does not imply in any way that it can only model two-dimensional neural structures. Indeed, SpiNNaker can model neural networks that are formed in two, three or even more dimensions. The key to this flexibility is to map each neuron into a virtual address space, which means that each neuron is assigned a unique number. The assignment can be arbitrary, though an assignment related to physical structure is likely to improve the modeling efficiency. Then neurons are allocated to processors; again in principle the allocation can be arbitrary, but a well-chosen allocation will lead to improved efficiency. Finally, the routing tables must be configured to send spike events from each neuron to all of the neurons to which it connects, and this can be achieved using the neurons' addresses.

The dissociation between the physical organization of the computer system and the physical organization of the biological system it is being used to model is possible owing to the very high speed of electronic communications relative to the speed of propagation of biological signals. This means that the delays

inherent in getting a spike event across many chips in the **SpiNNaker** system are negligible on the time-scales of neuronal processes. There is a drawback to the high speed of electronics, however. The physical delays in the biological system are likely to be functionally important; therefore, they must be re-instated in the electronic model. This is one of the more difficult and expensive aspects of the computational task.

5.5 Diverse Approaches

Blue Brain and **SpiNNaker** are both highly parallel systems employing large numbers of general-purpose processors to deliver flexibility (through programmability) in the neuron models that they support. Beyond this apparent similarity, however, there are marked differences in the way these two machines will be used to investigate neural computation.

The **Blue Brain** project emphasizes biological fidelity, and as a result uses high-performance processors with support for high-precision real-valued arithmetic which allows complex equations to be solved efficiently. The **SpiNNaker** design emphasizes the real-time modeling of very large numbers of much simpler equations; therefore, it uses simpler processors which support only integer arithmetic, which can still yield accurate solutions to differential equations as illustrated in Fig. 4, though considerable care must be taken over operand scaling.

There are many other differences between the two machines, with **Blue Brain** using an ‘off-the-shelf’ supercomputer while **SpiNNaker** is a bespoke design, the latter therefore having a lightweight communications architecture highly tuned to the neural modeling application.

While there remains so much uncertainty about the fundamental principles of biological neural processing, the diversity of approach reflected in the differences between **Blue Brain** and **SpiNNaker** (and neuromorphic and FPGA-based systems) is to be welcomed. No one knows which of these approaches is the most promising. It is our belief, based upon our experience as computer engineers, that large-scale complex systems are unlikely to be robust if they depend critically on the fine detail of the components from which they are constructed, so we are looking for explanations and insights at the network level and ignoring much of the biological detail. Whether or not this belief is justified only time will tell, but this is the belief that is driving the current direction of the **SpiNNaker** project.

6 Future Prospects

A great deal is known about the function and behavior of the neuron, but a great deal more remains to be revealed. If estimates of the computing power required to model a neural system of the complexity of the human brain are

not grossly misconceived, computers fast enough to do the job are not far away. But computing power alone will not solve the problem.

It is not predictable when or where a breakthrough in our understanding of brain function will emerge, so there is considerable merit in the diversity of approaches that are now in evidence. The core activities will remain for some time the painstaking bottom-up laboratory work of the neuroscientist and the top-down human-centric approach of the psychologist. But alongside these, there are challenges for computational neuroscientists, computer scientists, and electronic and computer engineers, all of whom can find opportunities to explore the potential of the neuron as inspiration for novel ideas in computation.

There are many possible approaches within the constructionist territory, some of which we have indicated in this paper. The spectacularly well-resourced **Blue Brain** project has the computing power to build highly accurate models of biological systems, and we can expect dramatic insights into the operation of complex neural systems to arise from that work, to complement the exotic images and visualization facilities they have already demonstrated. With our own work, we are leaving a lot of the biological complexity behind and working with more abstract neural models, with the expectation that the world of complex event-driven dynamical systems will yield insights both into the biology that we employ loosely as inspiration for our work and into novel models of computation.

Acknowledgements

Steve Temple is supported by the EPSRC Advanced Processor Technologies Portfolio Partnership at the University of Manchester. Steve Furber holds a Royal Society-Wolfson Research Merit Award. The **SpinNaker** research is supported by EPSRC, and by ARM Ltd. and Silistix Ltd. The support of these sponsors and industrial partners is gratefully acknowledged. This Chapter originally appeared as an article in *J. Royal Society Interface*, 2007, 4: 193–206; permission by the Royal Society to reprint it in the current Handbook is gratefully acknowledged.

References

1. Adrian ED (1964) *Basis of sensation*. Haffner, London, UK.
2. Austin J, Kennedy J, Lees K (1995) The Advanced Uncertain Reasoning Architecture, AURA. In: *Proc. Weightless Neural Network Workshop (WNNW'95)*, 26–27 September, University of Kent, UK.
3. Bainbridge WJ, Furber SB (2002) CHAIN: a delay-insensitive chip area interconnect. *IEEE Micro*, 22: 16–23.
4. Binzegger T, Douglas RJ, Martin KAC (2004) A quantitative map of the circuit of cat primary visual cortex. *J. Neuroscience*, 24(39): 8441–8453.

5. Boahen KA (2000) Point-to-point connectivity between neuromorphic chips using address events. *IEEE Trans. Circuits and Systems*, 47(5): 416–434.
6. Bower JM, Beeman D (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural Simulation System*. Springer-Verlag, New York, NY.
7. Eliasmith C, Anderson CH (2003) *Neural Engineering*. MIT Press, Cambridge, MA.
8. Fatt P, Katz B (1952) Spontaneous subthreshold activity at motor nerve endings. *J. Physiology*, 117: 109–128.
9. Furber SB, Bainbridge WJ, Cumpstey JM, Temple S (2004) A sparse distributed memory based upon N-of-M codes. *Neural Networks*, 17(10): 1437–1451.
10. Furber SB, Temple S, Brown AD (2006) On-chip and inter-chip networks for modeling large-scale neural systems. In: *Proc. Intl. Symp. Circuits and Systems (ISCAS'06)*, 21-24 May, Kos, Greece. IEEE Press, Piscataway, NJ: 1945–1948.
11. Furber SB, Temple S, Brown AD (2006) High-performance computing for systems of spiking neurons. In: Kovacs T, Marshall JAR (eds.) *Proc. Adaptation in Artificial and Biological Systems Workshop (AISB'06) – GC5: Architecture of Brain and Mind 2*, 3-6 April, Bristol, UK. Society for Artificial Intelligence and the Simulation of behavior: 29–36.
12. Furber SB, Brown G, Bose J, Cumpstey MJ, Marshall P, Shapiro JL (2007) Sparse distributed memory using rank-order neural codes. *IEEE Trans. Neural Networks*, 18(3): 648–659.
13. Gerstner W (1995) Time structure of the activity in neural network models. *Physics Reviews E*, 51: 738–758.
14. Hebb DO (1949) *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, NY.
15. Hellmich HH, Geike M, Griep P, Mahr P, Rafanelli M, Klar H (2005) Emulation engine for spiking neurons and adaptive synaptic weights. In: *Proc. Intl. Joint Conf. Neural Networks (IJCNN'05)*, 31 July - 4 August, Montreal, Canada. 5: 3261–3266.
16. Hodgkin A, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiology*, 117: 500–544.
17. Izhikevich EM (2004) Which model to use for cortical spiking neurons? *IEEE Trans. Neural Networks*, 15: 1063–1070.
18. Izhikevich EM (2005) Simulation of large-scale brain models. (available online at: http://vesicle.nsi.edu/users/izhikevich/human_brain_simulation/Blue_Brain.htm#Simulation_of_Large-Scale_Brain_Models – last accessed October 2007)
19. Izhikevich EM (2006) Polychronization: computation with spikes. *Neural Computation*, 18: 245–282.
20. Jahnke A, Roth U, Klar H (1996) A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN). *MicroNeuro*, 96: 232–237.
21. Kanerva P (1988) *Sparse Distributed Memory*. MIT Press, Cambridge, MA.
22. Kohonen T (1972) Correlation matrix memories. *IEEE Trans. Computers C*, 21: 353–359.
23. Laughlin SB, Sejnowski TJ (2003) Communication in neuronal networks. *Science*, 301: 1870–1874.

24. Lichtsteiner P, Posch C, Delbruck T (2006) A 128×128 120 dB 30 mW asynchronous vision sensor that responds to relative intensity change. In: *Proc. Intl. Solid State Circuits Conf. (ISSCC'06)*, 4-9 February, San Francisco, CA: 508-509.
25. Lomo T (2003) The discovery of long-term potentiation. *Philosophical Trans. Royal Soc. B*, 358: 617-620.
26. Mahowald M (1992) VLSI analogs of neuronal visual processing: a synthesis of form and function. *PhD Dissertation*, California Institute of Technology, Pasadena, CA.
27. Markram H (2006) The blue brain project. *Nature Reviews Neuroscience*, 7: 153-160.
28. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin Mathematical Biophysics*, 5: 115-133.
29. Mead CA (1989) *Analog VLSI and Neural Systems*. Addison Wesley, Reading, MA.
30. Mead CA (1990) Neuromorphic electronic systems. *Proc. IEEE*, 78(10): 1629-1636.
31. Mehrtash N, Jung D, Hellmich HH, Schoenauer T, Lu VT, Klar H (2003) Synaptic plasticity in spiking neural networks (SP²INN): a system approach. *IEEE Trans. Neural Networks*, 14(5): 980-992.
32. Moore GE (1965) Cramming more components onto integrated circuits. *Electronics*, 38(8): 114-117.
33. Mountcastle V (1978) An organizing principle for cerebral function: the unit module and the distributed system. In: Edelman GM, Mountcastle VB (eds.) *The Mindful Brain*. MIT Press, Cambridge, MA: 7-50.
34. NEURON (available online at: <http://www.neuron.yale.edu/neuron> - last accessed October 2007).
35. O'Keefe J, Recce ML (1993) Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3): 317-330.
36. O'Reilly RC (1996) Biologically plausible error-driven learning using local activation differences: the generalized recirculation algorithm. *Neural Computation*, 8(5): 895-938.
37. Prange SJ, Klar H (1993) Cascadable digital emulator IC for 16 biological neurons. In: *Proc. 40th Intl. Solid State Circuits Conf. (ISSCC'93)*, 24-26 February, San Francisco, CA: 234-235, 294.
38. Rall W (1959) Branching dendritic trees and motoneuron membrane resistivity. *Experimental Neurology*, 1: 491-527.
39. Saudargiene A, Porr B, Wörgötter F (2004) How the shape of pre- and post-synaptic signals can influence STDP: a biophysical model. *Neural Computation*, 16: 595-625.
40. Schoenauer T, Mehrtash N, Jahnke A, Klar H (1998) MASPINN: novel concepts for a neuro-accelerator for spiking neural networks. In: Lindblad T, Padgett ML, Kinser JM (eds.) *Proc. Workshop on Virtual Intelligence and Dynamic Neural Networks (VIDYNN'98)*, 26-28 June, Stockholm, Sweden: 87-97.
41. Schwartz J, Begley S (2003) *The Mind and the Brain: Neuroplasticity and the Power of Mental Force*. Regan Books, New York, NY.
42. Sivilotti M (1991) Wiring considerations in analog VLSI systems, with application to field-programmable networks. *PhD Dissertation*, California Institute of Technology, Pasadena, CA.

43. Sloman A (2004) GC5: The architecture of brain and mind. In: Hoare CAR, Milner R (eds.) *UKCRC Grand Challenges in Computing - Research*. British Computer Society, Edinburgh, UK: 21–24.
44. Van Rullen R, Thorpe S (2001) Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. *Neural Computation*, 13(6): 1255–1283.
45. Werbos P (1994) *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley, New York, NY.
46. Willshaw DJ, Buneman OP, Longuet-Higgins HC (1969) Non-holographic associative memory. *Nature*, 222: 960–962.
47. Yang Z, Murray AF, Wörgötter F, Cameron KL, Boonsobhak V (2006) A neuro-morphic depth-from-motion vision model with STDP adaptation. *IEEE Trans. Neural Networks*, 17(2): 482–495.
48. Zhu J, Sutton P (2003) FPGA Implementations of neural networks - a survey of a decade of progress. In: Cheung PYK, Constantinides GA, de Sousa JT (eds.) *Proc. 13th Annual Conf. Field Programmable Logic and Applications (FPL'03)*, 1–3 September, Lisbon, Portugal. Springer-Verlag, Berlin: 1062–1066.

Resources

1 Key Books

Adrian ED (1964) *Basis of Sensation*. Haffner, London, UK.

Bower JM, Beeman D (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SIMulation System*. Springer-Verlag, New York, NY.

Eliasmith C, Anderson CH (2003) *Neural Engineering*. MIT Press, Cambridge, MA.

Hebb DO (1949) *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, NY.

Izhikevich EM (2007) *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. MIT Press, Cambridge, MA.

Kanerva P (1988) *Sparse Distributed Memory*. MIT Press, Cambridge, MA.

Mead CA (1989) *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.

Werbos P (1994) *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley, New York, NY.

2 Key Reference Source

Encyclopedia of Computational Neuroscience
http://www.scholarpedia.org/article/Encyclopedia_of_Computational_Neuroscience

3 Research Groups

Advanced Computer Architecture Group at the University of York (UK)

<http://www.cs.york.ac.uk/arch/>

Advanced Processor Technologies Group at the University of Manchester (UK)

<http://www.cs.manchester.ac.uk/apt>

Brains in Silicon at Stanford University (USA)

<http://www.stanford.edu/group/brainsinsilicon/>

Brain-Mind Institute at EPFL (Switzerland)

<http://bmi.epfl.ch/>

Institute of Neuroinformatics at the University of Zurich and ETH Zurich (Switzerland)

<http://www.ini.uzh.ch>

Microelectronics Division at the Technical University of Berlin (Germany)

<http://mikro.ee.tu-berlin.de/spinn>

Neural Networks Group at the University of Edinburgh (UK)

<http://www.see.ed.ac.uk/research/IMNS/neural/>

The Neurosciences Institute, San Diego (USA)

<http://www.nsi.edu/>

<http://www.izhikevich.com>

4 Key International Workshop

Telluride Workshop on Neuromorphic Engineering

<http://www.ine-web.org/>

5 (Open Source) Software

NEURON

www.neuron.yale.edu/neuron

GENESIS

<http://www.genesis-sim.org/GENESIS/>

Neural Engineering Simulator

<http://compneuro.uwaterloo.ca/codelibrary/codelibrary.html>

Artificial Brains: An Evolved Neural Net Module Approach

Hugo de Garis

Director, Artificial Brain Lab, Institute of Artificial Intelligence, Cognitive Science Department, School of Information Science and Technology, Xiamen University, Xiamen, Fujian Province, China, profhugodegaris@yahoo.com

1 Introduction

This Chapter shows how the first author and his research team build artificial brains [9]. An artificial brain is defined to be a collection of interconnected neural network modules (10,000–50,000 of them), each of which is evolved quickly in special electronic programmable hardware, downloaded into a PC, and interconnected according to the designs of human ‘BAs’ (‘Brain Architects’). The neural signaling of the *artificial brain* (A-Brain) is performed by the PC in real time (defined to be 25 Hz per neuron). Such *artificial brains* can be used for many purposes, such as controlling the behaviors of autonomous robots.

If one uses only a PC to perform the evolution of the neural net modules, there will be a problem, and that is that the PC evolution speed is often too slow. Typically, it can take many hours to even an entire day to evolve a single neural net module on a PC. Obviously, evolving several tens of thousands of such modules using only a PC to build an artificial brain will not be practical. Before such *A-Brain*s can be built using this approach, it will be necessary to find ways to accelerate the evolution of such a large number of neural net (NN) modules. This we have done, so that it is now possible for us to execute the evolution of neural net modules in hardware, and thus achieve a speedup factor (relative to ordinary PC evolution speeds) of about 50 times.

We use a **Celoxica** field programmable gate array (FPGA) electronic accelerator board (containing a 3-megagate FPGA, namely a Xilinx **Virtex II** programmable chip) to accelerate the evolution of neural network modules. One of the aims of this Chapter is to report on the measurement of the speedup factor when using this **Celoxica** board to evolve neural network modules, compared to using a PC, as performed on the same set of evolution tasks. In the later parts of this Chapter we provide a more general description

of how we build *artificial brains* (using the accelerator board – an essential component of our method).

The experiments reported in this Chapter involved the evolution of fairly simple neural nets. The tasks chosen to serve as the basis for comparison are described in detail in Sects. 2.1 and 4.

The evolutionary tasks used two different approaches and technologies, namely:

1. a standard GA (Genetic Algorithm) on a PC, and
2. using the **Celoxica** board and the high level language **Handel-C** [1] to program the evolution (<http://www.celoxica.com>).

Once the two execution time measurements were obtained, the speedup factor can be calculated. Final results are given in Sect. 4.

The remaining contents of this Chapter are as follows. Section 2 places the current Chapter in context by describing related work. Section 3 provides an overview of how we evolved our neural network modules. Section 3.1 describes briefly the evolution tasks we used to calculate the speedup factor using the **Celoxica** board compared with a PC; Sect. 3.2 provides some details on how we performed the evolution on the **Celoxica** board; Sect. 3.3 gives a brief description of the ordinary genetic algorithm we used to perform the evolution. Section 4 briefly describes the characteristics of the **Celoxica** board. Section 5 presents the experimental results. Section 5.1 explains the so-called ‘IMSI’ (Inter Module Signaling Interface) – that is, the software used to allow modules to send and receive signals between each other. Section 5.2 answers the question “How Many Modules?” – in other words how many modules can an ordinary PC handle in an *artificial brain*?

The UXO Robot and its Brain-Robot interface is described in Sect. 6. Section 7 makes some general remarks about *artificial brain* architectures, before passing on to a long Sect. 7.1, which presents details of a simple *A-Brain* architecture. Section 7.2 shows how one can increment the design of an artificial brain; Sect. 7.3 answers a basic question “Why Not Just Program Everything?”; Sect. 7.4 gives some (4) concrete examples of how one can evolve individual neural net modules. Section 8 raises a ‘routing time’ problem with the **Celoxica** board and presents a solution we found for this, called *Generic Evolution*. Section 8.1 discusses some limitations of our approach. Section 8.2 talks about what one can do when modules fail to evolve, this being the topic of *evolvability*, which is a key issue in *evolutionary engineering*. Section 8.3 presents ideas on the imminent need for ‘book keeping of modules and circuits’ as their number grows. Section 9 talks about future work, and Sect. 10 concludes.

2 Related Work

This Section describes work related to that of this Chapter. As far as we know, there is no other project on the planet that attempts to build an *artificial brain* by connecting large numbers of evolved neural net modules. Hence we cannot describe closely similar work, but there are other ‘brain building’ projects. We describe some of them in this Section.

It is only recently, thanks to Moore’s Law, that computing capacity has become large enough to make brain building realistic, so not surprisingly, several ambitious research projects have arisen during the past few years to attempt to build *artificial brains*. The topic of Brain Building has become popular recently as evidenced by the number of hits one obtains (November 2007) by a Google search on the term *artificial brain* – namely, about 55,000.

However, in the decades before Moore’s Law made brain building feasible (at least in terms of the very large number of artificial neurons involved – that is, billions) there was nothing to stop people from theorizing about how the human brain works, or how to create cognitive architectures. There is a section in the *Resources Appendix* devoted to a sample of these attempts, divided into the following topics:

1. A-brain Architectures
2. Brain Theory
3. Cognitive Modeling
4. Ethology

The author has several thousand books in his private library devoted to the brain (for instance neuro-anatomy, neural networks, neuroscience, brain theory, cognitive science, computational neuroscience, neurophysiology, ethology, and so forth). Very few of these books attempt explicitly to provide architectural designs for *artificial brains*; most of those that do are in the *Resources Appendix*.

The above number (namely 55,000 Google hits on the term ‘artificial brain’) is greatly exceeded by the number of hits achieved for the term ‘brain theory’ (about 200,000). There have been numerous authors over the past few decades who have attempted to find theoretical principles that are applicable to the functioning of the brain. A sample of such authors is listed in the *Resources Appendix*. The term ‘cognitive modeling’, when Googled, returns around 230,000 hits. So there has been a comparable interest in inventing systems that have artificial cognition, which need not imitate closely the human brain.

Ethology (the study of animal behavior) has provided some fairly detailed models as to how animal or insect brains make decisions amongst their behavioral repertoire.

The above four topics (artificial brain architectures, brain theory, cognitive modeling, ethology) are thought by the author to provide essential knowledge and guidance to any potential brain builder. It is therefore hoped that other brain builders will read such literature before they attempt to construct their own *artificial brains*. The author advises his research team to read widely in these areas for ideas, for inspiration.

2.1 Some Recent Artificial Brain Projects

This sub-Section describes some of the most interesting (in the view of the author) ‘artificial brain’ like research projects currently underway.

Markram’s Blue Brain Project

Henry Markram is a neuro- and computer scientist working at Switzerland’s Ecole Polytechnique Federale de Lausanne (EPFL) in collaboration with IBM. He is currently conducting one of the most interesting *artificial brain* projects in the world, and attracting huge attention from the world’s media. He uses his neuroscience expertise to simulate the behavior at the synaptic level of detail of a neural cortical column. He inserts a micro electrode into the individual neurons of a cortical column of a rat’s brain, electrocutes the neuron, and thereby extracts information about the connectivity of that neuron with other neighboring neurons in the column and the respective synaptic strengths. Painstakingly, this procedure is repeated for the tens of thousands of neurons in the cortical column, and all the detailed connectivity and synaptic strength data is fed into an IBM **Blue Gene** (8000-processor node) supercomputer, one of the most powerful in the world.

Markram hopes to capture the detailed knowledge of how a single cortical column is structured, and then to simulate its behavior in the **Blue Gene** supercomputer, hence the title of his research project, the ‘Blue Brain’ Project. In 15 years he hopes, thanks to Moore’s Law, to be able to use more powerful IBM supercomputers to simulate *all* million or so cortical columns in the human brain. This extremely ambitious and fascinating project obviously deserves the description of being an *artificial brain* project.

It differs from the author’s project in at least one major respect, this being cost. Markram’s project is based on its use of an IBM **Blue Gene** supercomputer costing many millions of dollars, thus making it out of reach to virtually all university research labs. The author’s project, on the other hand, is explicitly intended to be cheap, so that many different brain building groups can use its approach. Admittedly, Markram’s project can simulate the human brain at a far greater level of detail than could the author’s, but he will have few disciples, due to the expense of his hardware. Nevertheless, the author (and the world at large) are following Markram’s progress with avid

interest. His project has been described as “the sexiest artificial brain project on the planet”. For links to Markram’s work, see the *Resources Appendix* at the end of this Chapter.

Artificial Developments ‘CCortex’ Model

Artificial Development (AD) is a San Francisco-based multi-national company with a Spanish CEO, and collaborators mainly from Europe, the US and India. The company employs a 1000-processor node cluster of PCs to model the behavior of 20 billion artificial neurons and trillions of synapses. AD is building what it calls its *CCortex* Model, which quoting their web site (see the *Resources Appendix* at the end of this Chapter for links to AD.com),

“AD is creating a bio-realistic simulation of the whole human brain to enable highly functional computational systems. Our technologies will impact everything from medical research and security to advanced autonomous systems”.

AD’s goal is “to achieve a realistic whole-brain simulation for the purpose of creating new cognitive computational products”, which will be capable of performing such tasks as “pattern recognition, verbal and visual communication, knowledge acquisition, and conscious-approximate decision-making capabilities”.

“AD is currently working on three research projects: *CCortex*, *Cortical DB*, and *NanoAtlas*. *CCortex* is a massive spiking neural network simulation running on a high-performance parallel supercomputer. It accurately represents the billions of neurons and trillions of connections in the human brain with a layered distribution of neural nets. The primary focus of these neural nets is to emulate the 6-layered neocortex and important subcortical structures, such as the thalamus, basal ganglia, hippocampus and amygdala. It achieves this simulation by dynamically employing the vast amounts of neurological data derived from the *Cortical DB* and *NanoAtlas* projects.

The Cortical DB is a comprehensive database of neurological structures that represents multiple levels of the brain, ranging from data on neurotransmitters, synapses, neuron morphologies, and axon and dendrite branching, to connectivity patterns and gross structures. The database contains billions of neurons with trillions of connections, including data on neuron cell types, morphology, connectivity, chemistry, physiology and functionality. The database results from extensive data mining of the neuroscience literature. The *NanoAtlas* is a 100-nm resolution digital atlas of an entire human brain that is built using innovative whole brain histology, imaging and modeling techniques”.

AD’s project is similar in nature to that of Markram’s. Both use a cluster of 1000s of microprocessors to simulate the brain. Both use data bases on the

brain to inform their simulations. One obvious difference between the two is that Markram's project is based at a university, whereas AD is a company (but which collaborates with universities).

The *Blue Brain* and the *CCortex* projects differ from the author's project in that both the former are costly. It costs millions of dollars to build a cluster of PCs with thousands of processors. If one wishes to establish a community of research colleagues working on the same broad brain building approach to create a critical mass of people who can attend workshops and conferences to discuss common research efforts, then that approach will have to be cheap, so that many people can afford it. Cheapness is not a trait of either of these two projects.

Edelman's 'Darwin IV' Robot Brain

Gerald Edelman is a Nobel Prize winning neurophysiologist, Director of the Neurosciences Institute in San Diego and the author of several well known books on the brain, embryogenesis, and consciousness (see the *Resources Appendix* for details). His Institute studies "memory, learning, consciousness, attention, sleep", and is "dedicated to understanding how the human brain 'works' at the most fundamental level".

For the past few decades he has been building a series of computer-controlled robots called *Darwin i*, where *i* represents version number, for example *Darwin IV*). Edelman is famous for suggesting the concept of 'Neural Darwinism', which places the concept of Darwinian evolution inside the brains of individuals. Broadly speaking, Edelman is suggesting that groups of closely associated neurons compete with each other to respond to incoming stimuli. The winning group is (synaptically) reinforced, so that it is more likely to 'win' the next time a similar stimulus enters.

One of the issues that Edelman has taken to heart is his brain models is that of 'categorization', in other words how does an *artificial brain* go about creating categories to classify incoming stimuli (such as sights and sounds) that are useful for its life. This categorization needs to be spontaneous, that is, unsupervised. Edelman has implemented his ideas in a series of increasingly sophisticated software programs to control the behaviors of robots. A fairly recent such concoction was labeled *Darwin IV*, which learned to create 'appropriate' categories concerning the texture of real world wooden blocks and getting positive and negative 'rewards' on touching them.

There are many research projects similar to Edelman's so this one is really only representative of a general category, and is much closer in its goals to the author's. Since Edelman (actually his research assistants) used a PC to control the robot, the *Darwin IV* project was not so expensive. Its primary aim was to confirm Edelman's theories on how the brain works, rather than to build an artificial brain *per se*.

2.2 Some Other Recent Artificial Brain Projects

The remaining topics in this Section on related work, are not explicitly connected to building artificial brains, but are relevant to the tools that this project uses, more specifically, VLSI and evolvable hardware (EHW).

Implementing Neural Networks in Hardware (VLSI)

There have been many attempts over the years to implement neural networks in (digital and analog) hardware, to speed up the signaling speeds of the neural nets. For example, see [13] for a thorough review of neural networks implemented in hardware. A similar logic applies to the author's research project, with one major difference. The hardware is not used to accelerate the signaling speed of the neural nets; the signaling is done in the PC. But special hardware (the *Celoxica* board) is used to accelerate the speed of the neural network's evolution.

Many earlier hardware implementations of neural networks were truly 'hard' in the sense that the designs were 'frozen in silicon', so that changing the design would imply an expensive and slow re-creation of the hardware. Now that FPGAs are large and powerful (thanks to Moore's Law) they allow quick re-programming of the electronic design. In fact, the *Celoxica* company has caused a revolution in allowing non electronic engineers to program the FPGAs without the use of a hardware description language (HDL) such as *Verilog* or *VHDL*. *Celoxica* board users can use a C-like language (*Handel-C*) to configure the FPGA via a *Celoxica*-provided hardware compiler that translates *Handel-C* code into configuring bits for the FPGA. If we want to change our neural net model for example, then we simply change the *Handel-C* code, and re- (hardware) compile.

Evolvable Hardware (EHW)

In the Summer of 1992, the author (who was working in Japan in the 1990s), happened to be in the US talking with an American electronic engineering colleague about ideas on how to speed up the evolution of neural networks, using hardware. This conversation led the author to the idea of 'evolving hardware' by conceiving the configuring bit string of a programmable chip (FPGA) as a random chromosome in a Genetic Algorithm. The random configuring bit string would create a random circuit in the FPGA, whose performance could be measured by a conventionally programmed FPGA to determine the quality of the performance of the random circuit, in other words, its 'fitness' [3,6,10]. See the *Resources Appendix* below for lists of Evolvable Hardware (EHW) books, conference proceedings, journals, and the like.

The research field of EHW had its first workshop in Switzerland in 1995, its first conference in 1996 in Japan, and then every year or two since. The first

American EHW conference was held in 1999, under the sponsorship of NASA and the American Department of Defense (DoD). The NASA/DoD sponsored conferences have dominated the field ever since. Research efforts in the field of EHW have covered such topics as fault-tolerant electronic circuits, embryonic (that is, self constructing) hardware, the evolution of digital and analog circuits, and so on.

The author uses evolvable hardware techniques in his current research project only in an indirect sense, which differs from his original conception of 1992. By using a *Celoxica* board to accelerate the execution of a Genetic Algorithm (GA), one is not strictly speaking evolving hardware. The gates of the FPGAs are not directly under the control of evolving configuring bits. Instead the gates are controlled by a *non*-evolving hardware compiler provided by *Celoxica*.

3 The Evolution of Neural Network Modules

This Section gives a brief description of the approach that we normally use to evolve our neural network (NN) modules that become components for building *artificial brains* (For a list of books, conference proceedings, and journals on Genetic Algorithms, or more generally ‘Evolutionary Computation’, see the *Resources Appendix* at the end of this Chapter.) We use a particular neural net model called *GenNet*.¹ A *GenNet* neural network consists of N (typically $N = 12\text{--}20$) fully connected artificial neurons. Each of the N^2 connections has a weight, represented as a signed, binary fraction, real number, with p (typically $p = 6\text{--}10$) bits per weight. The bit string chromosome used to evolve the N^2 weights will have a length of $N^2(p + 1)$ bits. Each neuron j receives input signals from the N neurons (including a signal from itself). Each input signal S_{ij} is multiplied by the corresponding connection weight W_{ij} and summed. To this sum is added an external signal value E_j . This final sum is called the activation signal A_j to the neuron j .

$$A_j = \sum_{i=1}^N W_{ij} S_{ij} + E_j \quad (1)$$

This activation value is fed into a sigmoid g that acts as a ‘squashing’ function, limiting the output value S_j to have a maximum absolute value of 1.0.

$$S_j = g(A_j) = \frac{A_j}{|A_j| + 1.0} \quad (2)$$

¹ <http://www.iss.whu.edu.cn/degaris/coursestaught.htm> (CS7910).

3.1 The Evolutionary Tasks

In order to compare the evolution times for the two different approaches (that is, using an ordinary GA on a PC, and using the `Celoxica` board), two simple neural net evolutionary tasks were evolved. The first was very simple, namely to output a constant signal value of 0.8 over 100 clock ticks (where a ‘tick’ is defined to be one loop in the neural network signaling software, in which all neurons in the *artificial brain* calculate their output signals once). The second task is described in Sect. 4.

Each neuron of a neural network module has a weighted connection to (usually) a single output neuron, whose output signal is considered to be the output signal for the whole module. This output signal $S(t)$ is compared to a target (desired) signal value $T(t)$ for some number (say 100) of ‘ticks’. The *fitness function* used in the genetic algorithm (GA) to perform the evolution of the neural net module is usually defined as follows:

$$f = \frac{1}{\sum_{t=1}^{100} (T(t) - S(t))^2} \quad (3)$$

In our evolutionary task, the target value $T(t)$ is constant, namely $T(t) = 0.8$.

In order to compare the evolution times of the above task, a concrete ‘cutoff’ (or threshold) fitness value was used. This was found empirically, by evolving the standard GA version of what we felt was a reasonably ‘saturated’ fitness value. This fitness value was then used in the `Celoxica` evolution experiments. Once the evolution, as specified by the `Handel-C` program (see the following Section) executed on the `Celoxica` board reached the same fitness value, its evolution time (in seconds) was recorded. The task was evolved 10 times and the average value calculated.

3.2 Our Evolutionary Approach

The approach our group uses to increase the evolution speed of neural network modules is to perform the evolution in special hardware, for instance using a `Celoxica` board.² In this approach, a high-level language, called `Handel-C` is used. The `Handel-C` code used to program the genetic algorithm to evolve a neural net module is ‘hardware-compiled’ (‘silicon-compiled’) into the FPGA (Xilinx’s Virtex II chip).

The main reason for using an electronic programmable accelerator board was to accelerate the slow evolution speed of neural networks on a PC, which can take from many hours to over a day. Evolving 10,000s of modules is obviously not practical at such slow speeds, hence the need to speed up the evolution.

² <http://www.celoxica.com>

As the early Sections of this Chapter will show, by using the **Celoxica** board we get a typical speedup factor of about 50, which is important. This speedup factor makes brain building as we define it (namely by interconnecting large numbers of individually evolved neural net modules) practical.

3.3 The Standard Genetic Algorithm

The standard GA (Genetic Algorithm) used to help calculate the speedup factor consists of the following steps:

Algorithm 1 Standard Genetic Algorithm (GA)

(a) Randomly generate 100 bit string chromosomes of length $N^2 \times (p + 1)$ (over the years we have used values $N = 16$, $p = 8$, so our chromosomes (bit strings) were 2304 bits long).

repeat

(b) Decode the chromosome into the N^2 signed binary fraction weights, and build the neural network for each chromosome.

(c) Perform the fitness measurements for the task concerned (for details, see the next Section).

(d) Rank the fitnesses from ‘best’ to ‘worst’.

(e) Throw out the inferior half of the chromosomes; replace with the superior half.

(f) Mutate all the chromosomes except the top one. No crossover was performed in these experiments.

(g) Go to (b).

until the evolution saturates at the target (desired) fitness value (of the elite chromosome).

4 The Celoxica Board

The aims of lowering the price of high-speed evolution, and achieving higher performance in evolving hardware led us to use FPGAs (Field Programmable Gate Arrays). FPGAs are specially made digital semiconductor circuits that are often used for prototyping. The several million logic gates in modern FPGAs (for instance, the Xilinx Virtex II chip) make it possible to have multiple copies of the same electronic sub-circuit running simultaneously on different areas of the FPGA. This parallelism is very useful for a genetic algorithm. It allows the program to process the most computationally expensive weight calculations in parallel, and this can speed up the overall evolution by a factor of about 50 times.

We chose the **Celoxica** FPGA board for our project (Fig. 1). Our **Celoxica** board (an RC203) cost about \$1500. With such a board, a design engineer is able to *program* electrical connections on site for a specific application, without paying thousands of dollars to have the chip manufactured in mass quantities.



Fig. 1. The Author's Celoxica board with central FPGA (photographed by the author)

The RC Series Platforms of Celoxica are complete solutions for FPGA design and ASIC/SoC (system-on-a-chip) verification and prototyping. The boards combine very high-density FPGA devices with soft-core and hard-core processors, together with an extensive array of peripherals. They provide easy access to programmable SoC's from the Electronic System Level (ESL) and consistently deliver fast and efficient access to very high-density reconfigurable silicon.

The RC203 FPGA board is a desktop platform for the evaluation and development of high performance applications. The main FPGA chip is a Xilinx Virtex II that can be configured without using an HDL (Hardware Description Language). Instead, as mentioned earlier, it uses a much easier high-level 'C-like' language called Handel-C (after the composer Handel). This language is very similar to standard C (there being an overlap of approximately 80% between the two languages), with a few extra features, particularly those involved with specifying which functions ought to be executed *in parallel*.

A Celoxica board attaches to a PC, with two-way communication, so that instructions to the board come from the PC, and results coming from the board can be displayed on the PC screen.

The main aim of the earlier Sections of this Chapter is to show that using the Celoxica board makes brain building practical, due to the considerable speedup factor in the evolution of the individual neural network modules

used to make *artificial brains*. These early sections report on experiments we undertook to determine how much faster the evolution of neural net modules on the *Celoxica* board can be, compared to using a software approach in a PC. The value of this speedup factor is *critical* to this whole approach. If a substantial speedup factor can be achieved, then it becomes practical to evolve large numbers of neural net modules in a reasonable time, and connect them together to build *artificial brains* inside a PC.

5 Experimental Results

In this Section we report on results concerning two experiments. The first was to see how many neurons N we could fit into the FPGA chip, using a more compact GA (in other words the cGA [12]). By using fewer lines of code to specify the GA, as is the case in a cGA, there would be a lot more room on the FPGA for more neurons N in the neural network. (One simple variant of a compact genetic algorithm is that there are only two chromosomes in the population, these being the ‘elite’ chromosome and the ‘test’ chromosome. The *test* chromosome is mutated and if it has a higher fitness than the *elite* chromosome, it becomes the *elite* chromosome.) This can be expressed in pseudo-code as follows:

Algorithm 2 Compact Genetic Algorithm (CGA) (after [12])

```

Initialize and store two chromosomes  $C_e$  (‘elite’) and  $C_t$  (‘test’) randomly;
Measure the fitnesses of  $C_e$  and  $C_t$ , giving values  $F_e$  and  $F_t$ ;
Store  $F_e$  and  $F_t$ ;
while evolution has not stagnated do
  if  $F_t > F_e$  then
    replace  $C_e$  by  $C_t$ ;
    replace  $F_e$  by  $F_t$ ;
  Mutate  $C_t$ ;
  Measure  $F_t$ .

```

For the first experiment mentioned in Sect. 3 (namely evolving a constant output neural signal value), using the cGA, we were able to place $N = 28$ neurons on the chip. This is far more than we needed, so we undertook a second and slightly more demanding experiment. The aim this time was to evolve (using the cGA mentioned above) a sine curve output for half a wavelength. The number of ticks of the clock used for the curve (one ‘tick’ being one calculation cycle of the output signal for each neuron in the network) was 45. The number of bits in the weights was increased to 8 (1 for the sign, and 7 for the weight value).

The fitness definition (function) was the sum of the squares of the errors between the target sine half curve (in other words, $y(t) = \sin(\pi \frac{t}{45})$), and the

actual output signals over the 45 ticks t . The number of neurons was 12, population size was 256, and bit string chromosome length was $12 \times 12 \times 8 = 1152$ bits. The number of generations used was 128,000. This half sine curve evolved well, showing that a non-trivial neural net could be evolved in the 3 mega-gates available in the FPGA of the **CeLoxica** board. The speedup factor was about 57 times, compared to evolving the same task on the same PC used to control the **CeLoxica** board.

We felt that the success of this initial experiment was important because it showed us that it would be possible to evolve many other single neural network modules using our **CeLoxica** board. (More on this in later Sections.)

5.1 IMSI (Inter Module Signaling Interface)

In order that each neural net module can calculate the strength of its output signal, it needs to know the strengths of all its input signals, including not only from *intra*-module connections, but also from *inter*-module connections – in other words, either from the ‘external world’ (for example, from sensors), or from the output of other modules. Each module therefore needs a lookup table (LUT) which lists the sources of its external input signals (from sensors), and the integer IDs of the modules from which it receives their output signals. A similar lookup table is needed to specify to which other modules each module sends its output signal to.

One of the jobs of the *BAs* (Brain Architects) is then to specify the interconnections between the modules, and then enter them into these LUTs.

Special software was written for the PC, called IMSI (‘Inter Module Signaling Interface’) which calculates the neural signaling of each neuron in each module of the artificial brain. An ordinary PC is used to run the IMSI program, which calculates sequentially the output signal value of each module, for all modules in the artificial brain.

The IMSI loops through each module sequentially, using its input LUT to find the signal values of its external inputs, as well as its internal signal values. As was mentioned briefly in an earlier Section, but is presented here in more detail, an intermediate *activation function* A is calculated according to Eqn. (4):

$$A_i = \sum_{j=1,N} W_{ij} \times S_j + \sum_{j=1,P} E_i \quad (4)$$

where W_{ij} is the weight value of the connection between neurons i and j in the module, S_j is the value of the signal strength on that connection, E_i is the value of an external neural signal, N is the number of neurons in the module, and P is the number of external input signals for that module. The convention used above is that a signal travels ‘from’ neuron j ‘to’ neuron i .

The output signal S_i is calculated using the activation function A_i above, together with a *squashing function* f , according to:

$$S_i = f(A_i) \quad (5)$$

This non-linear function is used to prevent runaway positive feedback of the signaling. The function f creates an output signal value S which has a maximum absolute value of 1.0, found from the following simple equation, and which is also calculated quickly by the computer:

$$f(X) = \frac{x}{1.0 + |x|} \quad (6)$$

Each module has a table of its weight values W_{ij} (in other words, N^2 of them), together with a list of its external input signals (namely P of them, where P usually varies from module to module).

The IMSI software calculates the S_i signal value for each module and stores it in its output signal register (OSR). This is done sequentially for all modules in the artificial brain. The IMSI then places the output signal value of each module into the input signal registers of those modules that the outputting module signals to. For example, let us assume that module M1432 sends its output signal value (0.328) to two other modules M3729 and M3606. Then the IMSI would use the ‘Output Modules LUT’ of M1432 to place the value 0.328 into one of the input signal registers (ISRs) in each of the modules M3729 and M3606.

Thus the output signals of a given clock cycle (that is, the time taken for all modules to calculate their output signal values) become the input values at the next clock cycle.

Each module also has a table of its internal neural signal values S_j , where j ranges from 1 to N , the number of neurons in the module. These N values are calculated in each clock cycle, to be used in the following clock cycle. In the IMSI code, two such ‘S tables’ are used, in a ‘ping-ponging’ style. For example, table S_t (calculated in clock cycle t) is used to calculate table S_{t+1} (in clock cycle $t + 1$), which in turn is used to calculate table S_{t+2} , which actually overwrites table S_t (hence the term ‘ping-ponging’).

Within each clock cycle t , the IMSI calculates for each module m , its output signal value S_m , and updates its S table. The value S_m is transferred to m ’s connecting modules. Thus all the data is in place for calculating the S_m values in the next clock cycle. Performing all these calculations is the job of the IMSI software.

Some outputs of modules are not sent to other modules but to external *effectors*, such as to the motors of a robot, or to a transducer that generates a radio signal, and so on. The IMSI treats these special external outputs (not other modules).

Actually, the IMSI has two roles. Certainly its main role is as described above, in other words performing the neural signaling in the PC. Another, secondary role is to allow the *BAs* (Brain Architects) to specify the connections between the modules. Thus for example when *BAs* want to add a new module to the *artificial brain*, they will need to use IMSI to specify the module's

- external inputs (from sensors)
- inputs from other modules (that is, their integer IDs)
- outputs to other modules (that is, their integer IDs)
- external outputs (to effectors)

As the size of the *artificial brain* grows, special book-keeping software is used to describe each module, for example its function, its fitness definition, its size (the number of neurons), its evolutionary strategy, its connections with other modules, its position in the whole artificial brain design, and so on. (For more on this 'book-keeping', see Sect. 6.3). This software has links with the IMSI software.

5.2 How Many Modules?

The sequential calculation of the output signal value for each module led to the following interesting question: "Just how many modules can an ordinary PC handle in real time?" More specifically, if one specifies that real-time means that each artificial neuron in the whole brain signals at least 25 times a second (thought to be a reasonable lower limit, since it is a high enough signaling rate to control mechanical robots, or to generate a 'movie effect' with 25 images or frames of a movie per second), just how many interconnected modules could the PC process at that minimum speed?

We called this number the 'magic number' because it is critical to our whole approach. We wanted to use an ordinary PC to perform the neural signaling of the artificial brain, so that the approach is cheap. So we wrote the IMSI code and loaded it up with random inter-module connections, and random weights for each module. We then measured the frequency of signaling (namely the number of output signal value calculations per neuron per second) of each neuron for various magic numbers M .

Prior to actually performing experiments to determine M , we thought that this number would be in the hundreds to maybe over a thousand. We were surprised to find that it is in fact in the *tens* of thousands. For example, with a very modest 1.6 GHz processor laptop computer, M was in the range 10,000–20,000 modules. The computer was several years old, so that a modern dual-processor laptop could probably process 40,000–50,000 modules in real-time. Such a large number will allow an artificial brain to be built with a large number of behaviors and behavioral switching circuits, depending on incoming information from its sensors, and its internal state and memory. A 50,000 module artificial brain is quite substantial.

6 The Robot and Brain-Robot Interface

The following practical *A-Brain* research project is inherited from the author's years in the US. It is called the 'UXO Problem', where UXO means 'unexploded ordnance (for example, unexploded cluster bomblets). A cluster bomb releases smaller bombs that when falling release yet smaller bombs, so that at the ground, thousands of bomblets explode simultaneously, creating a fire storm that cooks anything or anyone trapped in its range. It is a formidable and controversial weapon.

However, cluster bomblets have a problem, namely that a small percentage of them don't explode, so that later, when the 'friendly troops' march over the bombed terrain, they may step on unseen UXO and trigger explosions, losing legs. Defense departments around the world would love to have a robot capable of detecting this UXO, picking it up and depositing it at some safe central point. In order to be able to do something as sophisticated as that, the robot would need quite a high level of artificial intelligence. It would need an artificial brain.

Such a task would provide a focus for the design of the *A-Brain*. The very nature of the task would concentrate the design effort. Obviously such a task would be predominantly visual and motion control oriented. Figure 2 shows a photo of our robot that will detect, collect and deposit UXO (actually orange ping pong balls) at some central position. It is controlled by a 2-way antenna

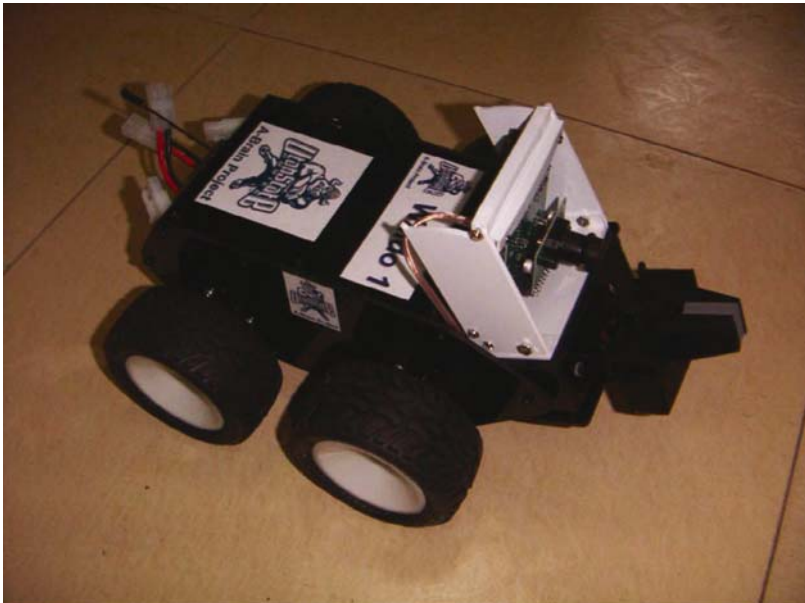


Fig. 2. UXO robot

at the back of the robot, having four wheels, a gripper at the front, and is fitted with a CMU-CAM2 programmable CCD camera.

Visual (and other) data from the robot is transmitted via radio signals to the *A-Brain* in the PC, and *vice versa*. This robot costs less than \$1000 to build, so is not expensive. Its length is about 20 cms. This robot will be designed to perform the UXO task. It will be controlled by an *artificial brain* consisting of at least 10,000-15,000 evolved neural net modules that will allow the *A-Brain* to have many hundreds of pattern recognition circuits, and a similar number of decision circuits.

With any publicly funded research project, one needs to show results. An *artificial brain* ‘hidden’ inside a PC is not visible. Even a large wall chart, consisting of thousands of interconnected neural net modules, may impress people with its complexity, but would not have the visual impact of a moving robot that displays hundreds of behaviors, switching from one to another depending on circumstances both external and internal. If the average person can remain intrigued for half an hour by such a robot and the *A-Brain* that controls it, then we state that the *A-Brain* has passed the ‘China Test’.

Our research team has a robotics/mechatronics/electronic/engineering expert, who is working on the interface problem between the robot and the *artificial brain* in the PC. The CMU-CAM2 camera on the robot is programmable, using an assembler-like language which can be used to process the mega-pixel image that the camera generates. This language contains about 100 different statements. The ‘robot guy’ of our team has the task of becoming an expert in this ‘camera language’, and then to provide the rest of the team, who are mostly *BA*s (Brain Architects) and *EE*s (Evolutionary Engineers) who design and evolve the many neural net modules for the *A-Brain*, with a summary of what the camera outputs, as well as ideas on how these outputs can be translated (interfaced) with possible inputs to neural net modules.

This mapping from mega-pixel camera images to single digit inputs to modules is what we call the robot-brain interface. This mapping, which we have yet to specify in detail, will be executed using normal computer code, in the PC (as will the IMSI), so our *A-Brain* is actually a ‘hybrid’ system, consisting of evolved neural network modules, and standard high-level computer code. Having a clearer idea of the robot-brain interface will give the research team a better idea of what ‘first-layer’ neural net modules to suggest, that take as their inputs the outputs from the interface mapping.

A type of reverse mapping, from the *A-Brain*’s output modules to the motors of the wheels, the camera motor, and the gripper motor, is also needed. Numerical output signal values from neural net modules need to be translated into a form that is suitable for the various motors. Signals between the robot and PC are sent via 2-way radio antenna, so another level of interfacing is needed – namely between the antenna and to the motors – and between the effectors (wheels, camera, and gripper) and the antenna that sends signals

to the PC. Once our roboticist has finished his work, it will be easier for the *BAs* defining the modules to design them to interface appropriately with the hardware aspects of the project. Of course, the major part of the effort involved in the project remains the specification and evolution of the 10,000s of modules used in the *A-Brain*.

7 Artificial Brain Architectures

This number of modules (several 10,000s) is more than adequate to build quite a sophisticated *artificial brain*, that could contain several thousand pattern recognition modules, hundreds of decision modules and many dozens of behavior control modules. Our research project has shown that the above is quite ‘do-able’. All the necessary ingredients are in place. Now all that needs to be done is to ‘bite the bullet’ and actually build the brain.

Since building a 50,000-module *A-Brain* is a major undertaking, our research team (comprising usually around 10 people) intends taking an incremental approach, starting with a very modest *artificial brain* consisting of about 30 modules, as described in reasonable detail below.

Once this ‘mini’ *artificial brain* has been well tested, the experience gained can be carried over to design larger artificial brains with 100, 200, 500, 1000, 2000, 5000, 10,000, 15,000 modules. At each stage, more people – that is, *Brain Architects* – will be needed. To save effort, each brain could be incorporated as a component in the next bigger brain.

To get a feel for the size of the brain building team needed at each stage, let us make some reasonable (‘back-of-the-envelope’) assumptions. Assume a *BA* can conceive and evolve 2 modules per day. How many *BAs* (N) would be needed to build an *A-Brain* of 20,000 modules (all different) over a period of four years?

We can calculate N using the following equation (namely, 4 years, 50 working weeks per year, 5 working days per week):

$$N \times 4 \times 50 \times 5 \times 2 = 20,000 \quad (7)$$

So N equals roughly 10, which means that our current research team could build a 20,000 module *artificial brain* in four years, with the basic assumption that on average about two modules can be evolved per day per *BA* (with each *Brain Architect* working full time).

Once each module is conceived, its actual evolution time is fairly quick, thanks to the use of the *Celoxica* board. What may take more time will be the conception of the modules themselves, depending on the modules. With experience, it is expected that *BAs* will become more efficient at ‘dreaming up’ new modules, their functions, their fitness definitions, and their training examples.

Once large numbers of different modules are evolved and documented, then these can be re-used in multiple copies. The above assumption of 20,000 different modules is quite unrealistic. In practice, many modules would be copies of a single module type. Therefore, the brain size could probably be about five times larger for the same time period (assuming the average number of copies of each different module is about five).

Once the number of modules grows larger than the magic number M of a single laptop computer, then several PCs can be formed into a cluster. If there are P computers in the cluster, the speedup would be almost linear (that is, P times faster), since the artificial brain is readily parallelizable. This is because most of the effort in the neural signaling calculations is devoted to the internal signaling of each module; only a small fraction of signals (under 10%) are sent between modules.

7.1 A Simple Artificial Brain Architecture

This Section describes a simple *artificial brain* architecture of several dozen modules, to illustrate the types of issues that arise when trying to design an *A-Brain* using our approach.

For this simple brain (for the illustrative purposes of this Chapter) we do not use a real robot, but rather a computer simulation of a ‘toy’ robot model, which has a horizontal rectangular body, with four wheels, and a V-shaped antenna, as shown in Fig. 3.

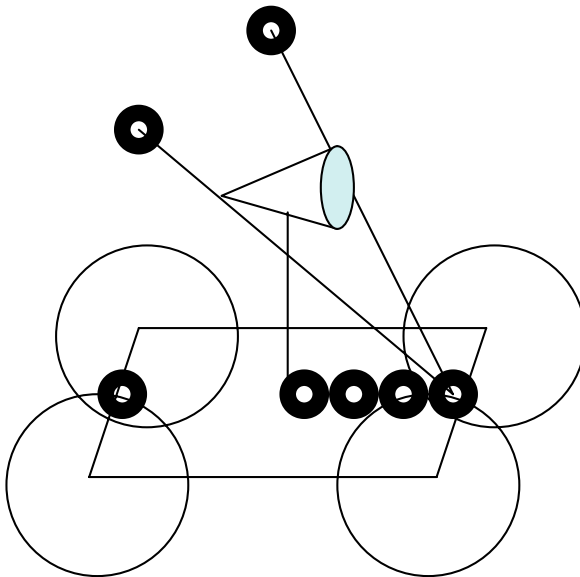


Fig. 3. The simulated robot ‘vehicle’

When designing an *artificial brain*, certain common sense factors come into play. For example, one needs to answer certain basic questions, such as

1. What is the *artificial brain* to control? In other words, what is the ‘vehicle’ that the *A-Brain* controls? (In our case, the vehicle is the (simulated) robot of Fig. 3)
2. What environment does the vehicle operate in?
3. What are the behaviors of the vehicle that the *A-Brain* needs to control?
4. What are the inputs and outputs of the vehicle?

Once these basic questions are answered, the more detailed *A-Brain* design work can begin.

In our simple case, the answers we give to the above questions are as follows:

The environment in this simple simulation model is a flat surface that contains small and large triangles and squares, that project onto the camera eye (the ‘cone’ in Fig. 3). Sounds are also emitted from the environment at two different frequencies (‘high’ and ‘low’).

The behaviors of the robot are simple. It can turn left slowly or quickly; it can turn right slowly or quickly; it can move straight ahead slowly or quickly, and it can stop. Thus there are seven behaviors to be controlled by the *A-Brain*.

The inputs to the robot are a triangle or square on the *retinal grid* of the eye (in other words, 2 possibilities), and a high- or low-frequency sound.

The next set of questions asks how input stimuli map to output behaviors for the vehicle. In our case we place the robot in the context of a ‘story’ – that is, a description of the behaviors of the robot in a ‘coherent’ context. This is rather vague, so will be illustrated twice with concrete examples to clarify the concept.

The first ‘story’ we provide is that the robot uses its detectors to see what the visual image is, namely is it a triangle or a square? It also detects whether the sound has a high or low frequency.

To the robot brain, these input signals are interpreted as follows. If the sound has a low frequency, then that means that the source of the sound (assumed to be some object in the environment) comes from far away, so the robot need not react quickly – in other words, its motions can be slow. If the frequency of the sound is high, then the robot interprets this to mean the object creating the sound is close, so it has to react quickly – therefore its motions are fast, not slow. Implicit in these interpretations is that a near object (high frequency sound) could be a ‘threat’ (such as a predator), so the robot needs to react quickly.

If the image is a triangle, then the robot interprets this as ‘positive’ (such as prey), as food, as something to be approached. If the image is a square,

then the robot interprets this as ‘negative’ (for example, as a predator), as dangerous, as something to be avoided. So the ‘story’ in this case is if the object detected in the retinal grid of the eye is dangerous, then flee. If it is prey, then approach it.

This mapping of sensor input to behavioral output makes sense in the context of the ‘story’ – namely “eat but don’t be eaten” – which also makes biological sense.

The robot has two other detectors (the black circles at the top of Fig. 3) on the tips of its V-antenna. These are *Signal Strength Detectors* (SSDs). If the sound strength drops off as a function of the distance between the robot and the source, then by having two such *SSDs* one can use them to locate the position of the source of the sound. For example, if the sound source lies to the front-left of the robot (that is, as the robot faces the object), then the object lies closer to the *SSD* on the left branch of the antenna than to the *SSD* on the right branch. Thus the signal strength will be stronger than that detected by the *SSD* on the tip of the right branch of the antenna.

One can use this *signal strength difference* in the decision making as to which behavior (of the seven) the robot ‘chooses’ to perform.

To give a concrete example, take the case of the sound having a high frequency, that the image is a triangle, and that the left *SSD* has a stronger signal than the right *SSD* (abbreviated to ‘L > R’). Then this combination of inputs maps to a given output in the form of a *rule*:

if (freq = high) and (image = triangle) and (L > R)
then turn left fast (abbreviated to LF)

In light of the above story, this rule makes sense. The frequency is high, so the sound source is close. Therefore the reaction of the robot should be fast, hence the ‘F’ in the behavioral selection (on the right hand side of the above rule). Since the image is a triangle, that means the object is a prey, to be approached. Since L > R, the object lies to the front-left of the robot, so to approach it, the robot should turn left, and quickly – in other words, the behavior selected is LF.

Once one understands this simple rule, then the rest of the rules listed in Table 1 are easily interpreted. In Table 1, in the Action column, an L means ‘left’, an R means ‘right’, an A means ‘approach’, an F means ‘fast’, an S means ‘slow’. For example, AS means ‘approach slowly’; LF means (turn) ‘left fast’.

Most of the actions in Table 1 are turning behaviors. Once the robot has turned sufficiently, so that the signal strength difference between the two *SSDs* on the V-antenna is effectively zero, the robot then moves straight ahead, in other words it approaches the source of the sound (or flees from it).

Table 1. Results for action selection

Frequency	Image	Position	Action
high	triangle	$L > R$	LF
high	triangle	$L < R$	RF
high	triangle	$L = R$	AF
high	square	$L > R$	RF
high	square	$L < R$	LF
high	square	$L = R$	LF
low	triangle	$L > R$	LS
low	triangle	$L < R$	RS
low	triangle	$L = R$	AS
low	square	$L > R$	RS
low	square	$L < R$	LS
low	square	$L = R$	LS

The stop behavior creates complications, so we will ignore it in this simple explanatory model. So effectively there are six behaviors.

Now that we understand the ‘story’ (in other words, the general behavior of the robot), we can now turn to a more detailed discussion of neural net modules that can implement these behaviors.

These modules are now listed. We need:

1. 2 detectors for the image – these being a ‘triangle detector’ and a ‘square detector’
2. 1 SSD (signal strength detector) – 2 copies, for each branch of the V-antenna
3. 2 frequency detectors (one for the high frequency, one for the low frequency)
4. 3 difference detectors (namely $L > R$, $L < R$, $L = R$)
5. 2 logic gates (‘and’, ‘or’)

There are only ten different modules, but this is enough for the purposes of this Section.

These modules can be combined (interconnected) to form networks of modules (in other words, a form of network of (neural) networks), called ‘circuits’ (or ‘sub-systems’), where the output(s) of some modules become the inputs of other modules.

For example, to implement the above rule, the following circuit could be used (Fig. 4):

Similar circuits can be constructed for the other 11 cases, however we do not need to make 12 separate circuits similar to that of Fig. 3 – that would be wasted effort. Instead we can use OR-gates to aggregate several cases. For example, Table 1 shows that there are three cases that give an LF output.

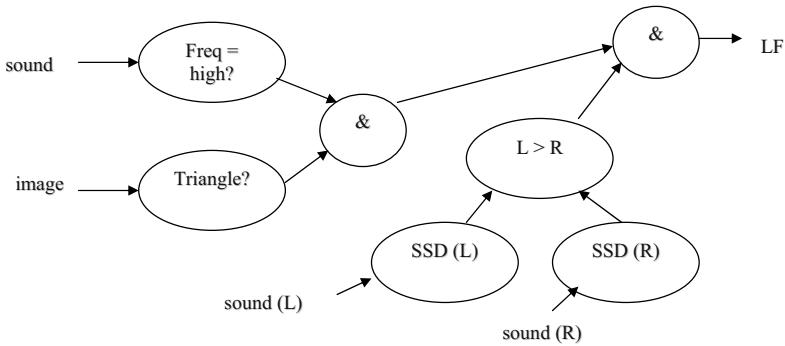


Fig. 4. Circuit for LF rule

Table 2. Control signals to wheels for robot turning

	Left wheel signal	Right wheel signal
LF	0.2	0.4
LS	0.2	0.8
AF	0.4	0.2
AS	0.8	0.2
RF	0.4	0.4
RS	0.8	0.8

So make those three LF outputs become the inputs to a 3-input OR-gate, and similarly for LS (three cases), RF (two cases), and RS (two cases). AF and AS only have single cases, so they don't need an OR-gate.

One need not replicate *SSDs*, nor difference detectors ($L > R$), ($L < R$), ($L = R$). One can simply have the outputs of these detectors branch to become inputs to multiple other circuits.

Putting all these 12 (non redundant) circuits together would generate quite a complex circuit, and is not given in this Section.

How can we implement LF, LS, AF, AS, RF, RS? Common sense says that to get a 4-wheeled vehicle to turn left, one needs to make the right side wheels turn faster than the left side wheels. Therefore use three more modules that output a constant signal of a low value (say 0.2) and a constant signal of a middling value (for instance 0.4), and a constant signal of a high value (such as 0.8). These different output signal values can be used to turn the wheels at different speeds. A high signal (0.8) will turn the wheel it controls quickly. A middling signal (0.4) will turn its wheel at a middling speed, and so on. Table 2 gives the combination of input signals to make the robot turn appropriately.

This brings the total of different modules now to 13.

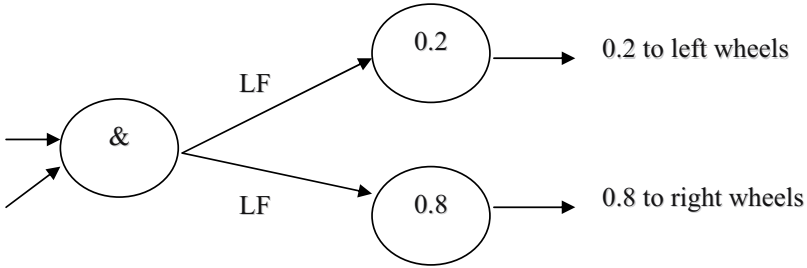


Fig. 5. Control signals to wheels for LF

If the behavior LF is activated, it can then send two output signals which become the inputs to modules that generate the two control signals for the wheels (namely 0.2 and 0.8). This simple circuit is shown in Fig. 5.

There are six different behaviors (LF, LS, AF, AS, RF, RS), with different combinations of control signal strengths to the wheels, so how do we solve the problem that only one set of signals should be sent to the wheels at a time? With six circuits functioning simultaneously, it is possible to have several of these have non negligible output values at the same time. This creates a conceptual problem. We want to have only one of these six to be active (for example, a strong output value of 0.8) and the rest to be inactive (that is, with weak output values of 0.2).

Before proceeding further, it is interesting to note that the problem we are now discussing is fairly typical of the role of a brain builder or a BA. It is analogous to an electronic engineer who designs digital electronic circuits. There is a lot of creativity involved, and there may be ‘many ways to skin a cat’ (in other words, many alternatives to solving a problem, some of them being more intelligent and/or efficient than others).

Returning to the problem: how can only one of the six signals be strong and the other five weak? This sounds like a ‘winner-takes-all (WTA)’ problem. So we suggest creating a circuit that has six inputs, and six outputs. If the *i*th input signal is the strongest of the six input signals, then the *i*th output signal is strong (say 0.8) and all the other output signals are weak (say 0.2), as shown in Fig. 6. How do we design such a circuit using evolvable neural net modules? (You see the scope for creativity?!)

The very idea of using a WTA circuit may not be the best way to solve the ‘unique signal problem’ (that is, ensuring that only one set of signals is sent to the wheel motors). An alternative might be to add the output signals of the six behaviors (LF+LS+AS+AF+RF+RS). This will generate some inaccuracies for a time, but so long as the behaviors don’t change very fast from one to another, there will probably be time for the contributions of the five non active behaviors to drop to low values, leaving only the active behavior. (On the other hand a weak signal is about 0.2, so the sum of five weak signals is

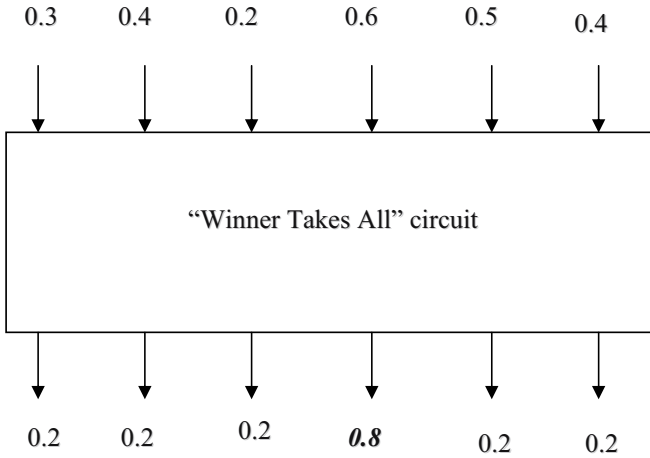


Fig. 6. Winner-takes-all (WTA) circuit

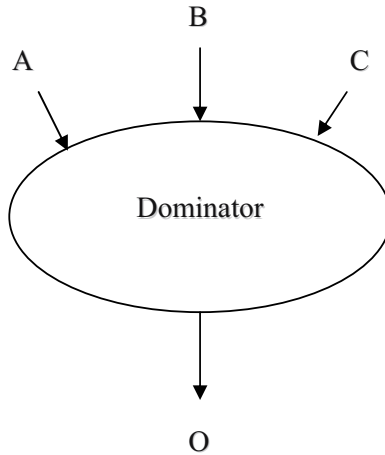


Fig. 7. Dominator circuit

roughly equal to one strong one, so we still have a problem). So let us pursue the WTA approach. How to implement such a circuit?

Trying to evolve a 6-input, 6-output circuit is probably too difficult, so our instinct is to ‘divide-and-conquer’, in other words, evolve simpler modules and then connect them to create the WTA circuit.

Consider a ‘dominator’ circuit, which has three inputs A, B, C, and one output O, as shown in Fig. 7.

If the input signal A (on the left) is the largest of the three input signals, then the output O takes the input value A; otherwise O takes a very low value

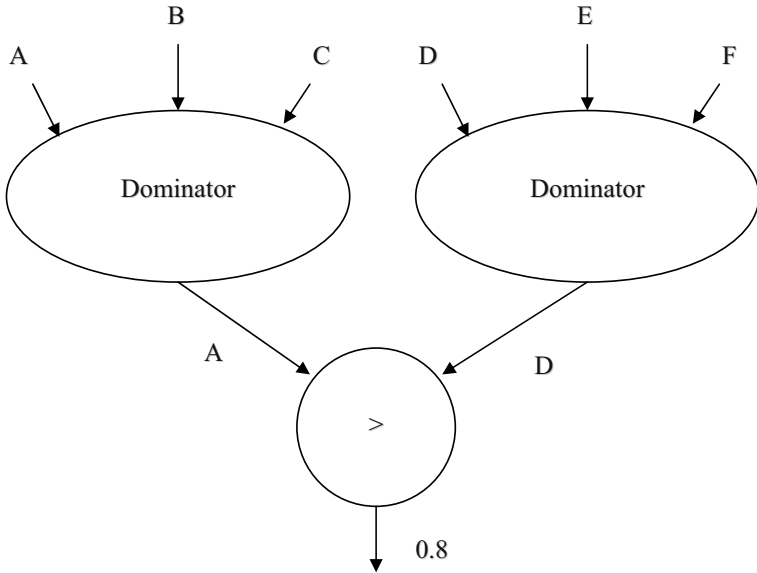


Fig. 8. Winner-takes-all (WTA) circuit component

(say 0.1 or 0.05). With two such circuits we can combine them with a ‘greater than’ circuit to create a WTA circuit (component), provided that the leftmost of the six inputs is the largest, as shown in Fig. 8.

It should now not be too difficult to see that with six such components, we can construct a winner-takes-all (WTA) circuit. For example, if the input-B has the largest input signal strength, then if we swap the inputs A and B, then using the circuit of Fig. 8, we should get a strong output (say 0.8), indicating that B was the strongest input. Similarly, if we swap A and C, then if C is the strongest, again the output O will be strong (else weak), and so on for the remaining three cases.

With six such circuits, we can construct a full WTA. The six outputs from the WTA circuit (one of which will be strong, say 0.8), and the other five very weak (say 0.05) are fed into six circuits of the form shown in Fig. 9. The six ‘left wheel’ output signals and the six ‘right wheel’ output signals can be paired – in other words, two of the left wheel outputs have a value of 0.1, two have a value of 0.4, and two have a value of 0.8 (similarly with the right wheel outputs). However instead of having six circuits as in Fig. 6, we can simplify things a little, as indicated in Fig. 9.

Using the data of Table 2, an LS output needs to send a 0.2 signal to the left motor and a 0.4 signal to the right motor. An LF output sends signals 0.2 and 0.8 respectively to the left and right motors. Hence LS and LF both send a 0.2 signal to the left motor. By connecting the two left motor signals

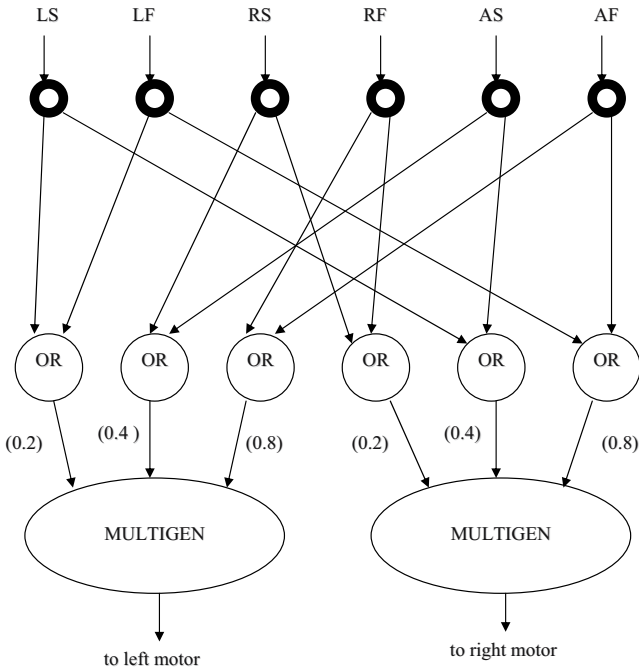


Fig. 9. Motor control circuit

to an OR-gate as shown in Fig. 9, if one of LS or LF is active from the WTA circuit the OR-gate will output a strong signal.

In the case of the LS/LF OR-gate, if one of LS or LF is active, then the output of the OR-gate is strong. This output connects to a new module called ‘Multigen’, which generates a multiple of outputs depending on which input has a strong signal. If the left input is strong (0.8) the output is a constant signal of 0.2; if the middle input is strong, the output is a constant signal of 0.4; if the right input is strong, the output is a constant of 0.8. Outputs go to the motors.

7.2 Incrementing the Design

The 14 different modules mentioned earlier are (presumably) sufficient to explain the behaviors of the robot as specified. But so far there are only 14 different modules involved, yet (as we have already shown) today’s technology allows *tens of thousands* of modules in an *A-Brain*. So how does one go about incrementing the number of modules? There are several possibilities.

One can throw out the previously evolved modules, and start a new larger design from scratch. This seems to be rather inefficient and when one is talking of a large number of modules is totally impractical. This is because each time

when starting from scratch, it will be impossible not to use previously evolved modules, especially once one is familiar with them and know that they work and how useful they are.

One can evolve modules incrementally – in other words, start with a relatively small number (say 20) – then add on a few more, to take the total to 30 say. After a bit more thinking, one could fairly quickly reach 100 modules. With a small team of people, one could then reach 200, 500, even 1000 with quite a bit of effort. Over 1000, one needs project management and a team of *BA*s. This logic becomes even more applicable as one reaches numbers like 2000, 5000, 10000.

Over the 50,000 threshold one may be talking of ‘national’ brain building projects, with teams of dozens of *BA*s (brain architects, brain builders). But up to the 10,000 module size only a small number of *BA*s are actually needed, as can be shown by making another quick ‘back-of-the-envelope’ calculation. We now repeat the same type of calculation previously performed in Sect. 5 (but on a smaller scale – that is, with fewer total modules). How many *BA*s would one need to build a 10,000 module *A-Brain* in three years, making reasonable assumptions? Assume a single *BA* can conceive, evolve and test three modules in a working day. Let N be the number of *BA*s in the team. In three years, these N people can create $N \times 3 \times 48 \times 5 \times 3$ (= 10,000) modules. Hence N is around five, in other words a small team.

This Section shows an example of the incremental addition of modules to an already existing *A-Brain*, namely the one explained already, with its 14 different modules.

As usual, one begins with a ‘story’. The story chosen for this addition was related to ‘boredom control’. The story is that if ‘nothing happens’ for a given time, then the robot becomes ‘bored’ and then makes a random selection of one of its behaviors to stop the boredom. This story translates into the following *boredom control* circuit (or subsystem), containing the modules shown in Fig. 10.

Figure 10 requires some explanation. The multiple OR-gate is fairly self explanatory. If any of the behavioral signals of (say LF, LS, and so on) are high, then the OR-gate will output a high signal. The inputs to the OR-gate (namely B_1, B_2, \dots, B_n) are the behaviors (LF, LS, and similar). The OR-gate is checking to see if anything is ‘happening’, that is if any behavior is occurring. If ‘yes’, and if the output of the OR-gate is high, then a new module, the *Impulse Generator module*, will output a short duration high value signal pulse for say 20 ‘ticks’. This pulse is formed when the output of the OR-gate goes high.

The pulse inputs to a *Boredom Meter module*, whose output is triggered by its input pulse. This output signal starts off at a low value (0.2) and climbs linearly in time to a high value (0.8) and saturates at that value, unless a new

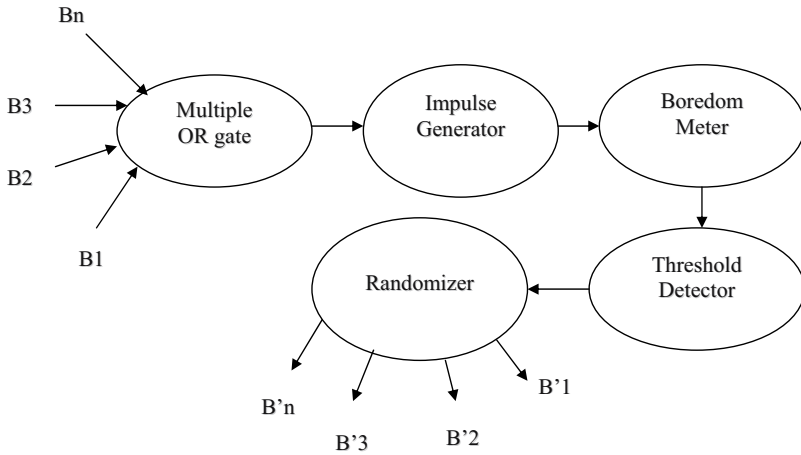


Fig. 10. Boredom control circuit

impulse arrives, which causes the *Boredom Meter module* to ‘reset’ its output – in other words become low again (0.2). Thus the size of the output signal of this module measures how ‘bored’ the robot is, namely how long it has been since any activity has occurred. The boredom tolerance is the number of ‘ticks’ taken for the *Boredom Meter module’s* output to rise from the low value (0.2) to the (saturated) high value (0.8).

This output value is then input to a *Threshold Detector module*, which outputs a high value when the signal value of its input from the *Boredom Meter module* is 0.8 (that is, a ‘high boredom level’). When that happens the *Threshold Detector module* outputs a strong signal (0.8) which is fed to a *Randomizer module*, which outputs a high signal on only one of its outputs, and low signals on all the others. It is called a ‘randomizer’, because the output on which the high signal value appears to be random. (At one time the high output will appear on one output, and then at another time at a different output.) On average, the distribution of the high outputs is even. The outputs (B_1, B_2, \dots, B_n), one of which is high (0.8) are the (LF, LS, and so forth) of Fig. 8.

The aim of the *Randomizer module* is to randomly select one behavior to be generated so as to overcome the robot’s boredom. Once a (random) behavior has been selected, the fact that that behavior has been generated, due to the boredom control circuit, the *Boredom Meter module* will output a low signal value. The robot is no longer bored (for a while at least).

It is easy to see that this (boredom control) circuit can be added to a pre-existing *A-Brain*, to help construct a more elaborate one. This incremental process can be continued indefinitely, creating larger and larger *A-Brains*, up

to the limit of what an ordinary PC can handle in real-time (namely, several 10,000s of modules).

7.3 Why Not Just Program Everything?

A question which may have occurred to some readers is that a lot of the modules discussed so far seem rather simple in their function and could be readily programmed in conventional high level computer software code. That being the case, why not just program everything, instead of evolving modules for everything? What is the point?

The answer is that it may be true for certain simple modules that they can be quickly and easily programmed, but that is not true for many modules, especially pattern recognition modules. One of the advantages of *evolutionary engineering* is that ‘EEs’ (evolutionary engineers), *BAs*, can often evolve a function without understanding at all how the evolved neural net module does what it does.

To a large extent *BAs* usually don’t care much how a particular evolved neural net module performs its function. For a start, there are too many of them. An ordinary PC can perform the neural signaling in real time of an *A-Brain* containing several tens of thousands of modules, so why would an EE take a particular interest in a particular module when it is only one of 10,000s? Even if a complete dynamical analysis of the signaling of that single module were undertaken, of what use would such knowledge be to the EE?

If a module can be evolved quickly, using evolvable hardware (EHW) techniques, and in a time comparable to using traditional programming techniques, then the objection that it might be quicker and easier to just program a function that a module performs, loses some of its sting. (For a list of books, conference proceedings, journals, and the like on Evolvable Hardware, see the *Resources Appendix* at the end of this Chapter.)

In practice, evolving a simple function module will usually take longer than just programming it, but this is largely irrelevant to the whole evolutionary engineering approach. In an *A-Brain* with 10,000s of modules, a non negligible proportion of them will be more or less ‘non-programmable’, that is it will not be obvious how to program their functions. For example, if a function is to detect triangles on a grid and output a strong signal, but to output a weak signal on seeing a square, and the figures can be placed anywhere on the grid, how would you program that, and quickly? There are many such examples.

One could use a hybrid approach, namely conventional programming and evolved modules, but that would be a different research project. This project uses a uniform approach, that is where all functions are performed with evolved neural net modules. This uniformity allows a single piece of software (the IMSI described in Sect. 4.1) to process all modules in the same uniform manner, which is conceptually simpler than a hybrid system.

7.4 Evolving Individual Modules

This Section describes how a selection (a subset) of the modules discussed in the previous Section can be evolved. The author is currently writing a book [11], in which will be descriptions of actual successful evolutions of many modules (probably many dozens), to give readers and potential brain builders a fairly thorough introduction as to how to evolve individual neural net modules, that is so critical to this approach of building *artificial brains*. In this Section we give only a few, because describing each one takes quite a bit of time, and there are already nearly 20 different modules introduced in this Chapter.

We start with a rather simple example, this being the *Multigen module* of Fig. 8. Remember its specification. It has three inputs and one output. If the leftmost input is high, and the others low, it outputs a constant signal of 0.2; if the middle input is high, it outputs a constant signal of 0.4; if the rightmost input is high, it outputs a constant signal of 0.8.

How can one evolve such a module? We use a ‘multi-test’ approach, which is a technique commonly used in our work. In this case there are three tests (or experiments) – that is, the three cases of input combinations (left high, middle high, right high). Let us run each test (namely, arrange to have the appropriate input signal values) for a given number of ‘ticks’, say 33, for each experiment. Thus the total number of ticks for the ‘multi-test’ experiment will be 100 ticks. Figure 11 shows the target output signal values for the ‘multi-test evolution’.

To evolve this module three experiments or tests are carried out on the same module, hence the word ‘multi-test’. For the first 33 ticks, three inputs

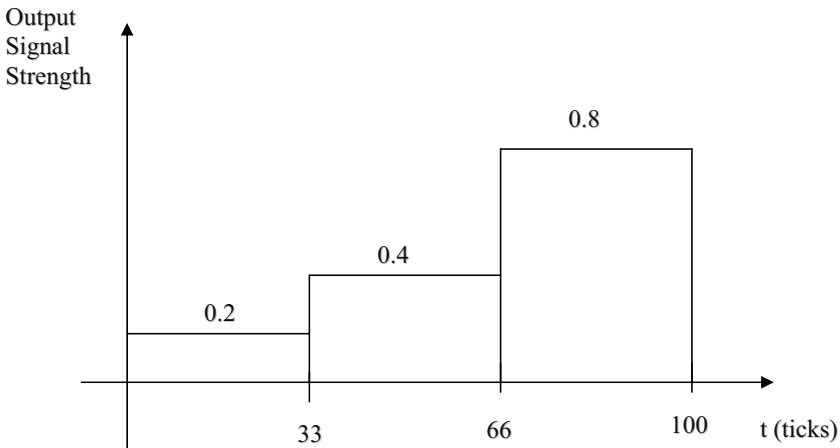


Fig. 11. Target output values for the Multigen module

(left to right) of value (0.8, 0.2, 0.2, respectively) are presented. For the second 33 ticks, the three inputs are (0.2, 0.8, 0.2), and for the third 33 ticks, they are (0.2, 0.2, 0.8). At the junctions of 33, 66 ticks, the internal neuronal signals are reset to zero, so that a ‘fresh’ experiment can begin. The fitness definition (function) of this module is as follows:

$$f = \frac{1}{\sum_{t=1}^{100} (T(t) - S(t))^2} \quad (8)$$

The $T(t)$ is the target (desired) output signal value. It takes the value 0.2 for the first 33 ticks, then 0.4 for the second 33 ticks, and 0.8 for the third 33 ticks. $S(t)$ are the actual output signal values from the evolving module. The closer the $S(t)$ values are to the target values $T(t)$, the higher the fitness score f . The fitness is a simple inverse of the sum of the squares of the differences between the T and S values for each tick t .

As another example of evolving a module, take the case of the *dominator circuit* of Fig. 6. How to evolve this module? Recall its functionality: if the input signal A (on the left) is the largest of the three input signals, then the output O takes its input value A ; else, O takes a very low value (say 0.1 or 0.05). Again we use a multi-test evolutionary approach. (It is difficult to overemphasize the importance of multi-test evolution to evolutionary engineering).

To evolve this module, we create a test set of cases, and divide them into two classes, called ‘positive’ and ‘negative’ (or ‘positive’ and ‘negative examples’, to use Machine Learning terminology). The positive examples are those for which A is indeed the largest of the three input values in which case the target output should be A . Negative examples are those for which A is not the largest input value of the three inputs, in which case the target output should be low (say 0.1).

We now prepare the training set (the positive and negative examples) to be used to evolve the module. Let the input signal values to A be one of 0.1, 0.3, 0.5, 0.7, 0.9, to B be one of 0.15, 0.35, 0.55, 0.75, 0.95, and to C be one of 0.2, 0.4, 0.6, 0.8, 1.0. There are thus $5^3 = 125$ possible input combinations. Of these 125, P are positive examples (where P can be readily calculated). Therefore there are $(125 - P)$ negative examples. We input the 125 cases sequentially, each for say 30 ticks. At the end of each block of 30 ticks, the internal neuronal signals of the modules are reset to zero, to begin another fresh evolutionary test.

The fitness definition of this module is similar to that just above, except that the target output signals $T(t)$ are now $A_i(t)$ for the i th positive example, and 0.1 for the $(125 - P)$ negative examples. Note that in general there are a different number of positive and negative examples, so integer ‘balancing

factors' are used in the fitness definition, to balance the tendency of the evolution to favor the bigger of the sum $((125 - P) \sum (a_i(t) - S(t))^2 + P \sum (0.1 - S(t))^2)$.

There are P positive cases, and $(125 - P)$ negative cases, so multiply the sum of the squares of the differences of the positive cases by the number of negative cases, (that is, $125 - P$), and multiply the sum of the squares of the differences of the negative cases by the number of positive cases (namely P). This should then force the evolution to favor the two sets of cases evenly. This 'balancing the evolution' between positive and negative cases in multi-test evolution, is a commonly used trick in evolutionary engineering.

This second example is more demanding than the first, so its evolvability may be lower. If so, then one may be forced to redesign the module or even the subsystem that the module belongs to. Such problems are a daily occurrence to *brain architects*.

For a third example, take the case of the frequency detector in Fig. 3. Again we create a training set of positive and negative examples. Let us assume that the environmental model in which the robot is situated contains only two frequencies, these being 'high' and 'low' – for example, a high frequency with period of 20 ticks, and a low frequency with period 40 ticks. Assume the strengths of the sound signals received by the robot take the following form:

$$S(t) = A_i \times \sin(2\pi t/T) \quad (9)$$

Let the amplitude of the signal (A_i) have three different values: 0.2, 0.5, 0.8. Hence there are six cases, namely three different possibilities for the amplitude, and two cases for the period T . There will be thus three positive and three negative examples, corresponding to whether the period is small (high frequency) or large (low frequency).

Use a six case multi-test evolution, in which the cases are each presented for 80 ticks. For the positive cases the target output is high (0.8), and for the negative cases the target output is low (0.2).

A similar fitness definition could be used to evolve a *signal strength detector*, our fourth and final example. If the input signal takes the same form as in Eqn. (9), then the target output signal value should be $\frac{A_i}{\sqrt{(2)}}$, in other words, independent of the period T . By having A_i take values 0.2, 0.3, 0.4, \dots , 0.8, 0.9, we have eight training examples. Each is input for 40 ticks in a multi-test fashion. The fitness would be simply the inverse of the sum of the squares of the differences of the actual output values and the corresponding target values.

For more examples of single neural net module evolution, readers will have to wait until publication of [11].

8 The Need for Generic Evolution

The main reason for using the **Celoxica** board is to speed up the evolution of individual neural net modules. If an *A-Brain* contains several 10,000s of modules then common sense says that building such an *A-Brain* will be totally impractical if it takes many hours or even a day to evolve each module. Hence the accelerator board is an essential component of brain building.

However there is a problem. After having written the **Handel-C** code to evolve an individual module, that code needs to be ‘hardware compiled’ into the FPGA. However the programmed routing process of the wiring between the programmable gates of the FPGA can take a full 40 minutes to be completed. This is not quick. If the evolution of every module needs this 40 minutes, then it rather destroys the rationale behind the use of the **Celoxica** board. If the actual evolution time using the **Celoxica** board (once the routing has been completed) is a minute or two, then the 40 minute routing time dominates the evolution time and makes the use of the board less competitive relative to evolving modules using an ordinary PC.

For example, imagine that a module takes one minute to evolve on the **Celoxica** board, and 50 minutes on a PC. So if the routing takes 40+ minutes to perform, then the total evolution time (routing time + genetic algorithm execution time) = 41+ minutes, which is almost equal to the PC evolution time, making the use of the board almost pointless. One may as well just use the PC and not bother with the expense of buying the **Celoxica** board (about \$1500).

What can be done to overcome this problem? As a result of thinking about how to solve this problem we came up with the concept of *generic evolution*. The idea is as follows. It is possible to feed the programmed FPGA with external data. One can then use different data as input, but still use the same programmed (and already routed) FPGA circuit.

This opens up the possibility of creating a general or generic circuit that can be fed different data to evolve different modules. The data that is fed in takes the form of various parameters that specify the nature of the neural net module that is to be evolved. These parameters include such things as the number of input signals, the number of output signals, the number of positive and negative cases in a multi-test evolution, and so on. Also input is the target output signal as a function of time.

Typically in a multi-test pattern recognition evolution, there will be a list of positive examples and negative examples. Typically the positive examples when presented as input should result in a strong output signal from the *pattern detector module*, and negative examples should output weak signals.

For the evolution of a *pattern detector module*, usually the positive examples are fed in as input first, then the negative examples. The internal signals

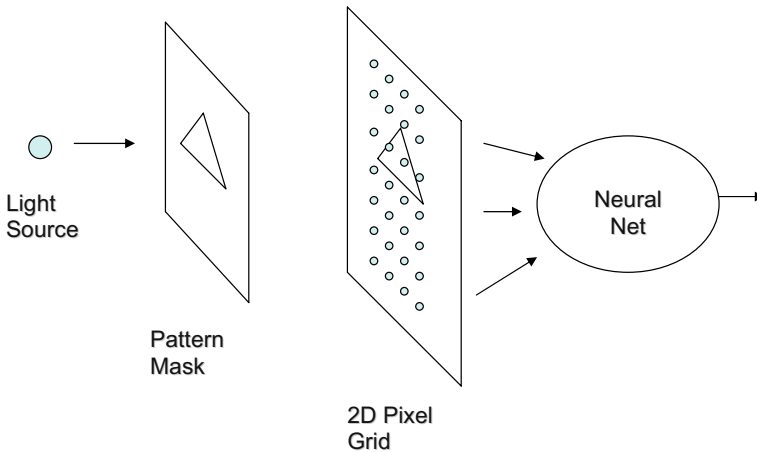


Fig. 12. Evolving a pattern detector

between the neurons are reset to zero whenever a new example is input. This input is usually a pattern of some object (such as a triangle or square) on a 2D grid of pixels (say $8 \times 8 = 64$ pixels). Some of these pixel elements will have strong output values, others weak, depending on how much light falls on the pixel, as shown in Fig. 12. Each pixel output signal is fed to each neuron in the neural net, with each such connection having its own evolvable weight.

The generic input parameters will also include the number of ‘ticks’ (where a tick is one loop in the neural signaling code in which every neuron calculates its output signal value) for the presentation of each example on the pixel grid (say 50 ticks). Let us assume that there are P positive examples and N negative examples (say P triangles, and N squares), and that the target (desired) outputs of the evolving module are high (say 0.8) for triangles, and low (say 0.2) for squares.

Then we can code the target outputs as a binary string where a ‘1’ represents a high desired output (0.8), and a ‘0’ represents a low desired output (0.2). Such a target binary string can be fed into the *Celoxica* board as a parameter string.

The inputs to the neural net – namely the set of positive and negative pattern examples – can be fed into the FPGA in the form of a set of concatenated binary strings, where a ‘1’ represents that light has fallen on that pixel, and a ‘0’ that light has not fallen on that pixel. The 2D pixel grid is mapped (or scanned) left to right, line by line from top to bottom, into a long binary string and input to the FPGA. The parameters P and N tell the genetic algorithm in the FPGA, how many positive and negative input strings (encoded 2D grid patterns) there are.

In this way it is possible to evolve a large number of pattern detector circuits, without having to re-route the FPGA. The same `Handel-C` program that is used to evolve the module can be used for many different sets of parameters. The program is thus ‘generic’, hence the term *generic evolution*. If, occasionally, the generic (`Handel-C`) program needs to be changed and re-routed, then that can be done, but not for every module.

Using generic evolution, the evolution time of a module can then be a mere minute or so, using the speed of the `Celoxica` board (some 50 times faster than a PC).

8.1 Limitations of Our Approach

What are some possible limitations of our approach to building *A-Brains*? At the time of writing, we do not yet have concrete experience of limitations, but we can anticipate these. The immediate problem we anticipate is something we call ‘unwanted synergy’, namely a kind of interference that occurs when adding new modules to those already existing in a given *A-Brain* architecture. By adding these new modules, the signals that they send out may cause unwanted interference with the signals amongst the pre-existing modules, especially if there are feedback loops created by the addition of the new modules to the previous modules.

At the time of writing we have not yet assembled a sufficient number of modules in an *A-Brain* to have noticed such a problem, but we will not be surprised if it does occur in the near future.

Another obvious limit is the total number of modules that can be evolved in a reasonable amount of time (several years, say). By evolving one module at a time and only several per *BA* per day, there is an obvious limit to how big an *A-Brain* can become.

Evolvability considerations imply that only fairly simple functionalities can be evolved in a module. Hence all the modules will be fairly simple. The complexity of the multi-module circuits must therefore emerge from the interconnectivity of the modules into more complex circuits, analogous to the way simple electronic components can be combined to produce complex electronic devices, such as a computer chip or a TV set. Hence a lot of work is needed to achieve complex behaviors from simple modular components.

8.2 Evolvability – A Key Issue

When one is an evolutionary engineer (EE), of which a *brain architect* is an example, the concept of ‘evolvability’ is critical. By definition, the term evolvability expresses the idea whether a certain function can be evolved – namely, is it evolvable? There is almost no theory on the concept of evolvability, but in

practice, for working *BAs*, the concept is critical, because all too often a particular module will just not evolve. From my own experience and that of my team (of about 10 people) I would say that about 10% – 20% of our attempts to evolve modules do not work, in other words they fail to evolve. Just why this is the case, we don't know. As I said, there is very little theory on evolvability to guide us, so we are forced to take an empirical 'hit-or-miss' approach, namely, if a particular function fails to evolve then change the function, the model or the architecture of the subsystem in which the module concerned is a part.

This can be a frustrating business. Sometimes several attempts are needed to get a module, or a subsystem of modules to work. Since there is no effective theory to guide good evolvability, if a particular function fails to evolve, all one can do is hope that by changing the function to some other, using a different approach, that the new function will evolve, and if it doesn't, then try again, and again, if necessary, until it does evolve.

8.3 Book-Keeping of Modules and Circuits

Eventually, over time, a whole library of successfully evolved modules can be accumulated, each properly documented, with such obvious attributes as, the module's name, the module's integer ID, a brief description of its function, its fitness definition, its training set (positive and negative examples), number of neurons, number of input neurons, number of output neurons, a list of its inputs (from the external world, or from other modules), a list of outputs (to the external world, or to other modules), and so on. With several 10,000s of such modules that can be used build an *A-Brain* with today's technology, this 'book-keeping' needs to be thorough, comprehensive and readily accessible.

A systematic approach needs to be used with the classification of modules. Hence such classification criteria need to be developed. It is expected that this will emerge with practice, as the number of modules being placed in the module library (data base) increases beyond the number of names and functions that anyone can readily remember.

Not only must individual modules be placed in the library, but a record has to be kept of the circuitry, that is the module inter-connectivity. If one makes an analogy with a very complex electronic circuit – for example, the latest computer processor chips – then the techniques used to describe the architecture and function of such chips ought to carry over to some extent to *brain building*.

For example, a computer chip will have its subsystems, and its sub-sub-systems, right the way down to single transistors. A similar top-down approach can be taken for brain building, for example the 'boredom' sub-system described earlier. As brain builder teams grow larger, in terms of the

number of *BAs* working on a particular brain building project, the need to specialize becomes obvious. For example, there could be a *vision team*, a *motion control team*, a *memory management team*, and so on.

Large brain building teams with dozens or more of *BAs* will need to be controlled by project managers. Perhaps the brighter, more creative members of the team could be the actual *brain architects*, and the rest be the people who actually evolve the modules, according to the specifications of the *BAs* (fitness definitions, training set, and so forth). When things fail to evolve, the *BAs* and the *MEs* (module evolvers, or let's just call them the *EEs* (evolutionary engineers) in the same way as an ordinary software programmer is given the fancy label of a 'software engineer') will need to consult.

The *EE* can inform the *BA* who conceived the module, that it did not evolve. The *BA* can then try again, or the *EE*, can try a different approach that achieves the same broader goal. In practice, in the author's team (of about 10–12 people, mostly graduate students, it is the author who tends to be the main *BA*, handing out assignments of modules to be evolved. The brighter ones will report back sometimes that their module(s) did not evolve, so they changed the model themselves and got it to work, or they made repeated attempts and still failed to get it to work. The author then either assigns the module to someone else, or rethinks it himself, or tries a whole new approach with a quite different model. Brain building is an art, there is no doubt about that.

9 Future Work

The 50-fold speedup of neural net module evolution enabled by the use of the *CeLoxica* board, compared with that of an ordinary PC will revolutionize brain building. It is now a lot more practical to evolve 10,000s of neural net modules within human patience limits. As mentioned earlier, a relatively small brain building team (say 5 people) could then build an artificial brain (of 10,000 modules) in three years, at a rate of three evolved modules per working day per person.

If somehow the total speedup factor could be made as high as 1000, then it is conceivable that new multi-module evolutionary algorithms could be created, that would make use of this speedup to evolve not only the intra-module weights of several modules, but their inter-module connections as well.

One of the weaknesses with our current brain building approach is that the modules are evolved individually, irrespective of how they will interact with other modules in a network of modules. Very rapid evolution of individual modules would allow new approaches to multi-module evolution.

But, for the moment, we are stuck with single module evolution, so a huge amount of work remains to be done, even when the goal is a rather modest

one of building an *A-Brain* of only 10,000 modules – that is, an *A-Brain* that an ordinary PC (personal computer) can handle – that any Computer Science/Engineering department could undertake (even without a *Celoxica* board). Hopefully, several such research groups will be set up in the near future to build their own *A-Brains*. If enough of them are formed, workshops on the topic could be held. Later, conferences could be organized and journals created (with titles such as *Brain Builder*, or *Artificial Brains*).

For the author’s group, we need more PhD students who can work on this project full time, and fewer Masters students who are busy with their courses, and just don’t have the time to devote hours every day to the brain building task.

One thing is clear, however. Everything is now ready to proceed with the task. The costs involved are modest, totaling only about \$2500 (\$1500 for the *Celoxica* board, and \$1000 for a robot to control, plus a PC which any laboratory will have). So the total cost is fairly minimal.

The method works. The author’s and his group’s experience over the years have shown that evolving individual neural net modules is readily doable. Simulation experiments on a PC have shown that several 10,000s of interconnected evolved neural net modules can have their neural signaling performed by a PC in real-time.

For a small team of *BAs*, several 10,000s of modules is still a lot, and will allow an *A-Brain* to be constructed with probably many hundreds of pattern recognition circuits. One wonders what fraction of the whole brain will consist of logic control modules. Since no one on the planet has yet built a 10,000 module *A-Brain* using this approach, we have little idea what such an *A-Brain* could actually do.

Section 6 described the UXO (unexploded ordnance) task. Implementing this task has yet to be done, so is an obvious task for the future. Figure 2 is a photo of our robot (that costs less than \$1000) that can be used to implement this task. Such a task would provide a focus for the design of the *A-Brain*. The very nature of the task would concentrate the design effort. Obviously such a task would be predominantly visual and motion control oriented.

Looking a bit further into the future, one can imagine larger *A-Brain* projects, even national scale projects – for example, the ‘C-Brain Project’ (that is, the China Brain Project); similarly for A-Brain, E-Brain, J-Brain, I-Brain and other Projects (where the letters stand for America, Europe, Japan, India, and so forth).

Moore’s Law (the doubling of the number of transistors placed on a chip every 18 months) will soon allow a PC to be able to handle the real-time neural signaling of 100,000 modules. This is only a few doublings from now. How could a brain building team cope with 100,000 modules? There are probably two approaches that could be used. One is simply to scale up the number

of BAs/EEs used in the team. For example, if five *BAs* can build a 10,000 module *A-Brain* in three years, then 50 people could probably build a 100,000 *A-Brain* in the same time. With a few more doublings in speed, one would be soon into the ‘million module’ range *A-Brain*. That would take 500 BAs/EEs. Such a large scale *A-Brain* project would be beyond what a university could do (especially in salary terms), so it would need to be undertaken by a national government, and hence could be classified as a national brain building project.

The second approach would be to do more research on the concept of ‘simultaneous multi-module evolution’, namely attempting to evolve several (many) modules at once. By around the year 2020, Computer Science will have an Avogadro number (a trillion trillion or 10^{24}) of components in its systems – the number of molecules in a human-scale object. With such large numbers, it will be possible to build artificial brains with ‘zillions’ of modules, so simultaneous multi-module evolution will become obligatory.

The above extrapolations into the future are assuming one continues to use an evolutionary engineering approach. Is this valid? One could argue that as the number of components in a computing system grows, so does the complexity, hence an evolutionary engineering approach may be even more essential, to cope with that complexity.

Another factor one should take into account when discussing the future of brain building is the impact of superior neuroscience. Any self respecting brain builder ought to be reading neuroscience for inspiration. As new principles are discovered as to how mammalian brains function, these principles can be incorporated into the designs of the *BAs*. One can thus expect that a marriage between the two fields of Brain Building (or *Artificial Brains*) and Neuroscience will become inevitable. Such a marriage should allow the creation of increasingly intelligent artificial brains, since their designs will be based evermore on how our own biological brains function.

9.1 The ‘China Brain’ Project

By 2008, the author will have changed professorial jobs and moved to Xiamen University, in the south of China, where he has been given a four-year 3,000,000 RMB research grant to build China’s ‘First Artificial Brain’, hence the title of the project, namely the ‘China-Brain Project’. The aim will be to build an *artificial brain* consisting of 15,000 evolved neural net modules, following the design principles outlined in this Chapter. It is planned that five PhD students will work full time for four years, designing, evolving and testing three neural net modules per day, 5 days per week, 50 weeks per year, for 4 years (in other words, $5 \times 4 \times 50 \times 5 \times 3 = 15,000$) to build an *artificial brain* to control the hundreds of behaviors and thousands of pattern recognition modules of an autonomous robot.

10 Conclusion

The early Sections of this Chapter presented results showing that a **Celoxica** FPGA board is capable of speeding up the evolution of neural network modules (relative to that on a PC) by a factor of about 50 times, depending on the size of the neural net being evolved. We believe this to be an important advance and an essential step when *artificial brains* – comprising 10,000s of such modules – are to be evolved in a reasonable time, and then run in real-time in interconnected form in an ordinary PC. Prior to the use of the **Celoxica** board, the evolution of a single neural network could take many hours on an ordinary PC, a fact that made brain building according to our PC-based approach quite impractical.

Most of the gates (flip flops) on the Xilinx chip on the **Celoxica** board, were taken up to implement the code of the genetic algorithm. Fortunately, there are smaller GAs in the literature, called ‘Compact Genetic Algorithms’ (cGAs) that by definition are very simple (taking less code) and hence need fewer logic gates for their electronic implementation. We were able to evolve larger NN modules with smaller GAs (with a 50-fold speedup).

Another factor assisting the growth in the size of modules is of course Moore’s Law. For example, the next generation **Celoxica** board, beyond the RC203 that we are currently using, has a Xilinx FPGA chip that contains 6 million logic gates, that is a doubling compared to our RC203 **Celoxica** board. **Celoxica** does indeed have a new board based on the new Xilinx **Virtex 4** chip.

Accordingly, our next research project may be to see how large our neural net modules can become, in other words just how many fully connected neurons can be evolved electronically on the **Celoxica** board. The underlying technology (the electronic evolution of large numbers of neural net modules) will make the production of 10,000s of evolved modules needed to build an *artificial brain*, practical.

Then the real challenge of designing an *artificial brain* can begin (as the latter part of this Chapter illustrates), and will result hopefully in the creation of a new research field, namely ‘Brain Building’ or ‘Artificial Brains’.

As mentioned in Sect.8, one other research challenge we overcame was to design a *generic evolution* approach to overcome the ‘slow routing of the wiring of the FPGA’ problem. We resolved this problem by programming a ‘generic FPGA circuit’ once, and then sending in different fitness definitions as external data from the PC to the circuit. To change the fitness definition one needs only to change the values of the parameters in the data that are input to the ‘changeless’ generic circuit. When one does need to change the generic model, this can be done by re-routing the **Celoxica** board, but it will take 40 minutes. Fortunately, this need not be done very often.

The above paragraphs in this summary were concerned primarily with issues arising in the context of creating the tools to allow brain building using our method. However, the majority of this Chapter introduced our research group's approach to building artificial brains using these tools.

An *artificial brain* was defined to be a network of evolved neural net modules, where the outputs of modules usually become the inputs of other modules. Each module is evolved quickly using a (CeLoXica) electronic accelerator board (about 50 times faster than using a PC). The evolved weights of each neural net module are downloaded into the PC's RAM. This is done thousands of times, one by one – a lot of work – up to a maximum of several 10,000s, which is the limit with which the PC can perform the neural signaling of all the (interconnected) modules sequentially in real-time. Here 'real-time' means the PC calculates the output signal value of every neuron in the whole *A-Brain* 25 times per second (25 Hz).

Once all the modules are downloaded into the PC's memory, they are interconnected according to the designs of human *Brain Architects*, for example the output of module M1593 connects to the second input of module M4295. The interconnected network of neural networks is the *A-Brain*.

Special software, called IMSI (Inter Module Signaling Interface) is used to specify the connections. The data structures of the IMSI software are largely LUTs (lookup tables) which contain such data as the list of modules a particular module receives its input from, and sends its outputs to. When the PC is calculating each neuron's output signal, it uses the IMSI's LUTs to know where its inputs come from, and to where it is to send its output signals.

Each neural net module performs some small function, and is evolved using a genetic algorithm in the CeLoXica board according to the module's fitness function (a mathematical measure of the quality of the performance of its function). The art of brain building is to design 'brain circuits' that consist of interconnected evolved neural net modules. Today's PCs allow the real time neural signaling of *A-Brains* with up to a maximum of several 10,000s of modules. With an *A-Brain* of such size it will be possible to give it many hundreds of pattern detector circuits, and a similar number of decision and behavior switching circuits, and so on. To observe a 10,000 module *A-Brain* controlling the hundreds of behaviors of a robot should give the impression to human observers that the robot 'has a brain behind it'.

Longer term, the research field of *Artificial Brains* should grow in size to generate big budget national projects equivalent to the national space agencies (NASA, ESA, and similar). Large national brain building projects would have the potential for building artificial brains comprising a million modules.

10.1 Final Word

We believe that the major point of this chapter is that it describes a new tool to build *artificial brains*. The tool is cheap (less than \$3000), quick (using the Celoxica board, one can evolve neural net modules 50 times faster than using a PC), and the tool works. We therefore hope that in a few years, dozens of brain builder groups will be established to build their own *artificial brains* using this tool. If there are dozens of groups, there will be dozens of different *artificial brain* architectures. A particular *artificial brain* architecture is not so significant in itself, especially if there are many of them. What is more significant is the tool that allows many different *artificial brain* architectures to be built. To make an historical analogy, which is more important, dozens of astronomical observations, or the invention of the telescope that enabled those observations?

Acknowledgements

The author would like to thank the members of his research team during the year and a half that he was at Wuhan University, Hubei Province, China. These people were: TANG Jian Yu (Brain Builder Group, International School of Software, Wuhan University, and the Computer Science School, Hubei University of Economics, Wuhan, Hubei Province) and HUANG Di (Computer Science School, University of Geosciences, Wuhan), who were the Handel-C and Celoxica board experts; HUANG Zhiyong (Computer Science School, Wuhan University) who was the robot (mechatronics) expert; and BAI Lu, CHEN Cong, CHEN Shuo, GUO Junfei, TAN Xianjin, TIAN Hao, TIAN Xiaohan, WU Xianjian, XIONG Ye, YU Xiangqian (all of the Brain Builder Group, International School of Software, Wuhan University), all of whom helped conceive neural net modules and evolved them. Finally, the author would like to thank Prof. John Fulcher for his assistance in preparing this Chapter.

References

1. Celoxia (2006) *The ‘Handel-C’ High Level Language for Programming the Celoxica board.* (available online at <http://www.celoxica.com> – last accessed November 2007).
2. de Garis H, Gers F, Korkin M, Agah A, Nawa N (1998) ‘CAM-Brain’: ATR’s Billion Neuron Artificial Brain Project: A Three Year Progress Report. *Artificial Life and Robotics J.*, 2: 56–61.
3. de Garis H (1999) Review of Proc. 1st NASA/DoD Workshop on Evolvable Hardware. *IEEE Trans. Evolutionary Computation*, 3(4): 304–306.
4. de Garis H, Korkin M (2000) The CAM-Brain Machine (CBM): Real Time Evolution and Update of a 75 Million Neuron FPGA-Based Artificial Brain. *J. VLSI Signal Processing Systems* (Special Issue on VLSI on Custom Computing Technology), 24(2-3): 241–262.

5. de Garis H, Korkin M, Gers F, Nawa E, Hough M (2000) Building an Artificial Brain Using an FPGA Based CAM-Brain Machine. *Applied Mathematics and Computation J.* (Special Issue on Artificial Life and Robotics, Artificial Brain, Brain Computing and Brainware), 111: 163–192.
6. de Garis H (2001) The Second NASA/DoD Workshop on Evolvable Hardware. *IEEE Trans. Evolutionary Computation*, 5(3): 298–302.
7. de Garis H, Korkin M, Fehr G (2001) The CAM-Brain Machine (CBM): An FPGA Based Tool for Evolving a 75 Million Neuron Artificial Brain to Control a Lifesized Kitten Robot. *J. Autonomous Robots*, 10(3): 235–249.
8. de Garis H (2002) Guest Editorial. *Neurocomputing* (Special Issue on Evolutionary Neural Systems: Prof. Hugo de Garis, Guest Editor), 42(1–4): 1–8.
9. de Garis H, Korkin M (2002) The CAM-Brain Machine (CBM): an FPGA-based hardware tool which evolves a 1000 neuron net circuit module in seconds and updates a 75 million neuron artificial brain for real-time robot control. *Neurocomputing*, 42(1–4): 35–68.
10. de Garis H (2004) Evolvable Hardware 2004. *Evolutionary Computation*, 12(3): 397–402.
11. de Garis H (2009) *Artificial Brains: An Evolved Neural Net Approach*. World Scientific, Singapore (in press).
12. Harik GR, Lobo FG, Goldberg DE (1999) The Compact Genetic Algorithm. *IEEE Trans. Evolutionary Computation*, 3(4): 287–297.
13. Lindsey C, Lindblad T (1998) Review of *Hardware Neural Networks: A User's Perspective*. (available online at: <http://www.particle.kth.se/~lindsey/elba2html/elba2html.html> – last accessed November 2007)

Resources

1 Key Books

1.1 Artificial Brain Architectures

Albus JS (1981) *Brains, Behavior, and Robotics*. McGraw Hill, New York, NY.

Ashby WR (1966) *Design for a Brain*. Chapman and Hall, London, UK.

Coward LA (2005) *A System Architecture Approach to the Brain: From Neurons to Consciousness*. Nova Science Publishers, New York, NY.

de Callatay AM (1986) *Natural and Artificial Intelligence: Processor Systems Compared to the Human Brain*. North Holland, Amsterdam, The Netherlands.

Edelman GM (1992) *Bright Air, Brilliant Fire*. Basic Books, New York, NY.

Kent EW (1981) *The Brains of Men and Machines*. Byte/McGraw Hill, New York, NY.

Pollock JL (1989) *How to Build a Person: A Prolegomenon*. MIT Press, Cambridge, MA.

Pollock JL (1995) *Cognitive Carpentry: A Blueprint for How to Build a Person*. MIT Press, Cambridge, MA.

Young JZ (1964) *A Model of the Brain*. Oxford University Press, Oxford, UK.

Young JZ (1978) *Programs of the Brain*. Oxford University Press, Oxford, UK.

1.2 Brain Theory

Calvin WH (1996) *How Brains Think: Evolving Intelligence, Then and Now*. Basic Books, New York, NY.

Calvin WH (1996) *The Cerebral Code: Thinking a Thought in the Mosaics of the Mind*. MIT Press, Cambridge, MA.

Freeman WJ (1999) *How Brains Make Up Their Minds*. Weidenfeld and Nicholson, London, UK.

Minsky M (1986) *The Society of Mind*. Simon and Schuster, New York, NY.

Minsky M (2006) *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon and Schuster, New York, NY.

Palm G, Aertsen A (1986) *Brain Theory*. Springer-Verlag, Berlin.

Shaw GL, Palm G (eds.) (1988) *Brain Theory*. World Scientific, Singapore.

Sporns O, Tononi G (1994) *Selectionism and the Brain*. Academic Press, New York, NY.

Valiant LG (1994) *Circuits of the Mind*. Oxford University Press, Oxford, UK.

1.3 Cognitive Modeling

Anderson JR (1983) *The Architecture of Cognition*. Harvard University Press, Cambridge, MA.

Green HS, Triffet T (1997) *Sources of Consciousness: The Biophysical and Computational Basis of Thought*. World Scientific, Singapore.

Hecht-Nielsen R, McKenna T (eds.) (2003) *Computational Models for Neuroscience: Human Cortical Information Processing*. Springer-Verlag, Berlin.

Kanerva P (1988) *Sparse Distributed Memory*. MIT Press, Cambridge, MA.

Lloyd D (1989) *Simple Minds*. MIT Press, Cambridge MA.

Newell A (1990) *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.

Palm G (1982) *Neural Assemblies: An Alternative Approach to Artificial Intelligence*. Springer-Verlag, Berlin.

Treuhub A (1991) *The Cognitive Brain*. MIT Press, Cambridge, MA.

1.4 Evolvable Hardware (EHW)

Greenwood GW, Tyrrell AM (2006) *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*. Wiley-IEEE Press, Piscataway, NJ.

Higuchi T, et al. (2006) *Evolvable Hardware*. Springer-Verlag, Berlin.

Sanchez E, Tomassini M (1996) *Towards Evolvable Hardware: The Evolutionary Engineering Approach*. Springer-Verlag, Berlin.

Sekanina L (2004) *Evolvable Components: From Theory to Hardware Implementations*. Springer-Verlag, Berlin.

1.5 Gerald Edelman

Edelman G, Mountcastle VB (1982) *Mindful Brain: Cortical Organization and the Group-Selective Theory of Higher Brain Function*. MIT Press, Cambridge, MA.

Edelman G (1989) *The Remembered Present: A Biological Theory of Consciousness*. Basic Books, New York, NY.

Edelman G (1990) *Neural Darwinism*. Oxford Paperbacks, Oxford, UK.

Edelman G (1993) *Bright Air, Brilliant Fire: On the Matter of the Mind*. Basic Books, New York, NY.

Edelman G (1993) *Topobiology: An Introduction to Molecular Embryology*. Basic Books, New York, NY.

Edelman G, Changeux J-P (2000) *The Brain*. Transaction Publishers, Edison, NJ.

Edelman G, Tononi G (2001) *A Universe of Consciousness: How Matter Becomes Imagination*. Basic Books, New York, NY.

Edelman G (2005) *Wider Than the Sky: The Phenomenal Gift of Consciousness*. Yale University Press, CT.

Edelman G (2007) *Second Nature: Brain Science and Human Knowledge*. Yale University Press, CT.

1.6 Ethology

Borrows M (1996) *The Neurobiology of an Insect Brain*. Oxford University Press, Oxford, UK.

Ewert J-P (1980) *Neuro-ethology: An Introduction to the Neurophysiological Fundamentals of Behavior*. Springer-Verlag, Berlin.

Heiligenberg WF (1991) *Neural Nets in Electric Fish*. MIT Press, Cambridge, MA.

Tinbergen N (1989) *The Study of Instinct*. Oxford University Press, UK.

1.7 Genetic Algorithms (GA)

Back T (1995) *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK.

Chambers LD (2000) *The Practical Handbook of Genetic Algorithms: Applications (2nd ed)*. Chapman and Hall, London, UK.

Coello CA, et al. (2002) *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer-Verlag, Berlin.

Coley DA (1997) *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, Singapore.

Davidor Y (1991) *Genetic Algorithms and Robots: A Heuristic Strategy for Optimization*. World Scientific, Singapore.

Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA.

Goldberg DE (2002) *The Design of Innovation*. Springer-Verlag, Berlin.

Haupt RL, Haupt HE (2004) *Practical Genetic Algorithms*. Wiley, New York, NY.

Michalewicz Z (1998) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

Mitchell M (1998) *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.

Sivanandam SN, Deepa SN (2007) *Introduction to Genetic Algorithms*. Springer-Verlag, Berlin.

2 Key Journals

Evolutionary Computation (MIT Press)

IEEE Trans. Evolutionary Computation

Genetic Programming and Evolvable Hardware (Kluwer)

3 Artificial Brain Research Groups

3.1 Markram's 'Blue Brain' Project

Website: <http://bluebrain.epfl.ch/>

Wikipedia article: http://en.wikipedia.org/wiki/Blue_Brain

IBM article: http://www-03.ibm.com/industries/education/doc/content/news/pressrelease/1334727110.html?g_type=rssfeed_leaf

3.2 Adaptive Development's 'CCortex'

Website: <http://www.ad.com>

AD Presentation to 'Accelerating Change' Meeting, 2005:

http://findarticles.com/p/articles/mi_m0EIN/is_2005_Sept_16/ai_n15393907

3.3 Edelman's 'Darwin IV' Robot Brain

Wikipedia biography on Edelman:

http://en.wikipedia.org/wiki/Gerald_Edelman

Edelman's Neurosciences Institute: <http://www.nsi.edu/>

On Darwin IV: <http://www.pnas.org/cgi/content/abstract/89/15/7267>

Neurosciences Institute research report:

<http://www.nsi.edu/uploads/pdf/ScientificReport.pdf>

4 Key International Conferences/Workshops

4.1 Congress on Evolutionary Computation – CEC (IEEE)

Proc. 1999 IEEE Congress on Evolutionary Computation, Washington DC, IEEE Computer Society Press, Los Alamitos, CA.

Proc. 2000 IEEE Congress on Evolutionary Computation, La Jolla, CA, IEEE Computer Society Press, Los Alamitos, CA.

Proc. 2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, IEEE Computer Society Press, Los Alamitos, CA.

Proc. 2002 IEEE Congress on Evolutionary Computation, Honolulu, HI, IEEE Computer Society Press, Los Alamitos, CA.

Proc. 2003 IEEE Congress on Evolutionary Computation, Canberra, Australia, IEEE Computer Society Press, Los Alamitos, CA.

Proc. 2004 IEEE Congress on Evolutionary Computation, Portland, OR, IEEE Computer Society Press, Los Alamitos, CA.

4.2 GECCO – Genetic and Evolutionary Computation Conference

Banzhaf W, et al. (1999) *GECCO'99: Proc. Genetic and Evolutional Computation Conf.*, Morgan Kaufmann, San Francisco, CA.

Whitley D, et al. (2000) *GECCO 2000: Proc. Genetic and Evolutionary Computation Conf.*, Morgan Kaufmann, San Francisco, CA.

GECCO (2001) *GECCO 2001: Proc. Genetic and Evolutionary Computation Conf.*, Morgan Kaufmann, San Francisco, CA.

GECCO (2002) *GECCO 2002: Proc. Genetic and Evolutionary Computation Conf.*, Morgan Kaufmann, San Francisco, CA.

Cantu-Paz E, et al. (2003) *Genetic and Evolutionary Computation – GECCO 2003: Proc. Genetic and Evolutionary Computation Conf.*, Chicago, IL. Lecture Notes in Computer Science 2723(4), Springer-Verlag, Berlin.

Deb K, et al. (2004) *Genetic and Evolutionary Computation – GECCO2004: Proc. Genetic and Evolutionary Computation Conf.*, Seattle, WA., Lecture Notes in Computer Science 3103, Springer-Verlag, Berlin.

ACM Sigevo (2005) *Proc. Genetic and Evolutionary Computation Conference – GECCO2005*, Washington, DC. ACM Press, New York, NY.

Keijzer M (2006) *Proc. GECCO 2006: Genetic and Evolutionary Computation Conf.*, Seattle, WA. ACM Press, New York, NY.

4.3 ICES – International Conference on Evolvable Systems

Hichuci T, et al. (1996) *Evolvable Systems: From Biology to Hardware – Proc. 1st Intl. Conf. (ICES1996)*, Tsukuba, Japan, Lecture Notes in Computer Science 1259, Springer-Verlag, Berlin.

Sipper M, Mange D, Perez-Uribe A (1998) *Evolvable Systems: From Biology to Hardware – Proc. 2nd Intl. Conf. (ICES1998)*, Lausanne, Switzerland, Lecture Notes in Computer Science 1478, Springer-Verlag, Berlin.

Miller J, et al. (2000) *Evolvable Systems: From Biology to Hardware – Proc. 3rd Intl. Conf. (ICES2000)*, Edinburgh, Scotland, Lecture Notes in Computer Science 1801, Springer-Verlag, Berlin.

Liu Y, Tanaka K, Iwata M, Higuchi T, Yasunaga M (2001) *Evolvable Systems: From Biology to Hardware – Proc. 4th Intl. Conf. (ICES2001)*, Tokyo, Japan, Lecture Notes in Computer Science 2210, Springer-Verlag, Berlin.

Tyrrell AM, et al. (2003) *Evolvable Systems: From Biology to Hardware – Proc. 5th Intl. Conf. (ICES2003)*, March, Trondheim, Norway, Lecture Notes in Computer Science 2606, Springer-Verlag, Berlin.

Moreno M, et al. (2005) *Evolvable Systems: From Biology to Hardware – Proc. 6th Intl. Conf. (ICES2005)*, Sitges, Spain, Lecture Notes in Computer Science 3637, Springer-Verlag, Berlin.

Kang L, Liu Y, Zeng S (2007) *Evolvable Systems: From Biology to Hardware – Proc. 7th Intl. Conf. (ICES2007)*, Wuhan, China, Lecture Notes in Computer Science 4684, Springer-Verlag, Berlin.

4.4 NASA/DoD Conferences on Evolvable Hardware

Stoica A, et al. (1999) *Proc. 1st NASA/DoD Workshop on Evolvable Hardware*, Pasadena, CA. IEEE Computer Society Press, Los Alamitos, CA.

Lohn J, et al. (2000) *Proc. 2nd NASA/DoD Workshop on Evolvable Hardware*, Palo Alto, CA. IEEE Computer Society Press, Los Alamitos, CA.

NASA (2001) *Proc. 3rd NASA/DoD Workshop on Evolvable Hardware*, Long Beach, CA. IEEE Computer Society Press, Los Alamitos, CA.

Stoica A, et al. (2002) *Proc. NASA/DoD Conference on Evolvable Hardware*, Alexandria, VA. IEEE Computer Society Press, Los Alamitos, CA.

IEEE (2003) *Proc. 2003 NASA/DoD Conf. Evolvable Hardware*, Chicago, IL. IEEE Computer Society Press, Los Alamitos, CA.

NASA (2005) *Proc. NASA/DoD Conf. Evolvable Hardware (EH-2005)*, Washington DC. IEEE Computer Society Press, Los Alamitos, CA.

Evolutionary Approaches

Evolving Artificial Neural Network Ensembles

Md. Monirul Islam¹ and Xin Yao²

¹ Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh*,
mdmonirulislam@cse.buet.ac.bd

² Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK, x.yao@cs.bham.ac.uk

1 Introduction

Artificial neural networks (ANNs) and evolutionary algorithms (EAs) are both abstractions of natural processes. In the mid 1990s, they were combined into a computational model in order to utilize the learning power of ANNs and adaptive capabilities of EAs. Evolutionary ANNs (EANNs) is the outcome of such a model. They refer to a special class of ANNs in which evolution is another fundamental form of adaptation in addition to learning [52–57]. The essence of EANNs is their adaptability to a dynamic environment. The two forms of adaptation in EANNs – namely evolution and learning – make their adaptation to a dynamic environment much more effective and efficient. In a broader sense, EANNs can be regarded as a general framework for adaptive systems – in other words, systems that can change their architectures and learning rules appropriately without human intervention.

EAs have been introduced into ANNs at roughly three different levels: (i) connection weights, (ii) architectures, and (iii) learning rules. The evolution of connection weights introduces an adaptive and global approach to training, especially in the reinforcement learning and recurrent network learning paradigms, where gradient-based training algorithms often experience great difficulties. Architecture evolution enables ANNs to adapt their topologies to different tasks without human intervention. The evolution of learning rules can be regarded as a process of ‘learning to learn’ in ANNs, where the adaptation of learning rules is achieved through evolution.

There is strong biological and engineering evidence to support the assertion that the information processing capability of ANNs is determined by their

* Portions reprinted with permission, from X. Yao and M.M. Islam, “Evolving artificial neural network ensembles,” *IEEE Computational Intelligence Magazine*, 3(1):31–42, February 2008. Copyright IEEE.

architecture. A large amount of the literature is therefore devoted to finding optimal or near optimal ANN architectures by using EAs (see review papers [48,54,59]). However, many real-world problems are too large and too complex for a single ANN alone to solve. There are ample examples from both natural and artificial systems that show that an integrated system consisting of several subsystems can reduce the total system complexity while satisfactorily solving a difficult problem. Many successes in evolutionary computation have already demonstrated this. A typical example of the success of ANN ensembles in improving classifier generalization is [62].

ANN ensembles adopt the divide-and-conquer strategy. Instead of using a single network to solve a task, an ANN ensemble combines a set of ANNs that learn to subdivide the task and thereby solve it more efficiently and elegantly. It offers several advantages over a monolithic ANN [47]. First, it can perform more complex tasks than any of its components (that is, individual ANNs in the ensemble). Second, it can make an overall system easier to understand and modify. Finally, it is more robust than a monolithic ANN, and can show graceful performance degradation in situations where only a subset of ANNs in the ensemble performs correctly.

There have been many studies in statistics and ANNs which show that ensembles, if designed appropriately, usually perform (generalize) better than any single member system. A theoretical account of why ensembles perform better than single learners is presented in [12]. Although ensembles perform better than their members in many cases, constructing them is not an easy task. As mentioned in [16], the key to successful ensemble methods is to construct individual predictors which perform better than random guessing and produce uncorrelated outputs. This means individual ANNs in the ensemble need to be accurate as well as diverse (also mentioned in one of the seminal works by Hansen and Salamon [23]). Krogh and Sollich formally show that an ideal ensemble is one that consists of highly correct (accurate) predictors which at the same time disagree – in other words, uncorrelate as much as possible (that is, substantial diversity amongst members is exhibited) [28]. This has also been tested and empirically verified [40,41]. Given that ANN ensembles generalize better as compared with a single ANN, ensemble research has become an active research area and has seen an influx of researchers devising myriad algorithms trying to improve the prediction ability of such aggregate systems in recent years.

2 Evolutionary Ensembles

Although there have been many studies on how to evolve ANNs more effectively and efficiently [48,54,59] the issue of how to form the final result from an evolutionary process has been overlooked [60]. The best individual in the last generation or among all the generations is generally considered as the final

result. However, the best individual (that is, ANN) with respect to training or validation data may not be the best for unseen testing data. The remaining individuals in the population may contain some useful information that may improve the generalization performance of ANNs.

The aim of this Section is to present an approach proposed by [62] to form the final result of an evolutionary process. Unlike most previous work, the approach utilizes the population information, rather than an individual's information, to form the final result. It considers each individual in a population as a module. Thus different individuals in the last generation are linearly combined by regarding a population of ANNs as an *ensemble*. The reason for using a linear combination is its simplicity, although non-linear combination methods could be used. The idea of combining different modules is not new and has been studied in both the ANN field and statistics [24, 42]. However, few attempts have been made in evolutionary learning to use population information in forming the final system.

The proposed approach was applied on three real-world problems to demonstrate the effectiveness of using the population information in forming the final result of an evolutionary process. **EPNet** [61] was used to evolve a population of ANNs. Four linear combination methods were used to form the final result. They were majority voting, the rank-based linear combination method, the recursive least-square (RLS) algorithm [38], and the subset method.

2.1 An Evolutionary Design System for ANNs – EPNet

EPNet [61] is an automatic ANN design algorithm based on evolutionary programming (EP) [18, 20]. It uses an EP algorithm for evolving ANN architectures and a hybrid training scheme for learning their connection weights. It puts the emphasis on evolving ANN behaviors, which is the main reason for using EP in its evolutionary scheme. Since **EPNet** evolves architectures and learns connection weights of ANNs simultaneously, it reduces the noise in fitness evaluation [61] unlike some previous studies (for instance, [58]). The main structure of **EPNet** is shown in Fig. 1; a detailed description can be found in [61].

EPNet relies on novel mutations and a rank-based selection scheme [53]. It does not use recombination operators in order to avoid the permutation problem (that is, competing conventions) [6, 9, 22]. This not only makes the evolution inefficient, but also makes crossover operators more difficult to produce highly fit offspring. To determine an ANN architecture for a given problem, an algorithm needs to select the number of hidden nodes and connections required for the ANN in solving the problem. In addition, it needs to assign weights for the architecture. **EPNet** uses five mutations one after another to achieve these goals. If one mutation is successful then other mutations are

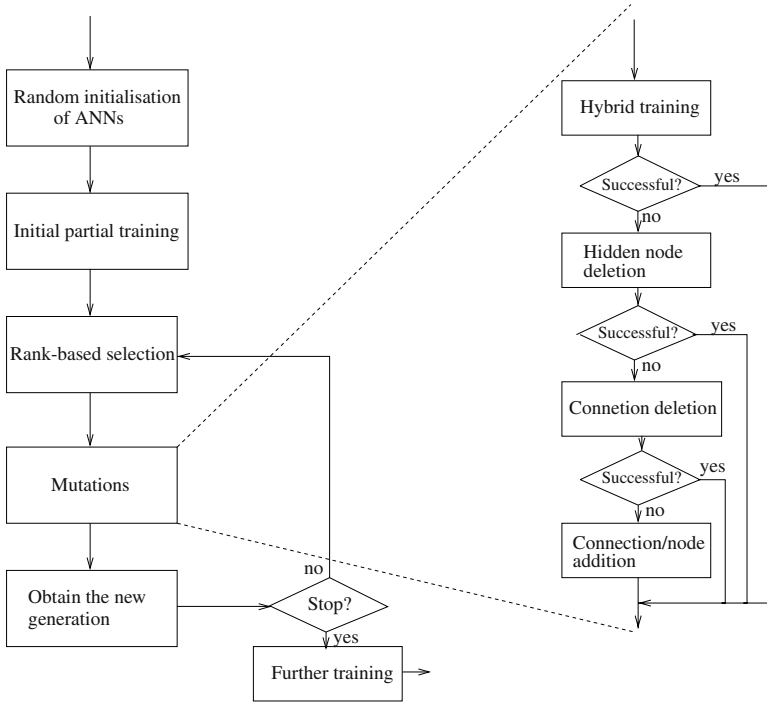


Fig. 1. The major steps of EPNet [61] © 1997

not applied. The mutations used in EPNet are hybrid training, node deletion, connection deletion, connection addition, and node addition. Each mutation operator in EPNet produces one offspring that can replace at most one individual in any generation of a population. This replacement strategy is very similar to the one used in a steady-state GA [51], or in a continuous EP [19]. The advantage of such replacement has been demonstrated in [19, 51].

The hybrid training scheme consists of modified back propagation (BP) [46] and simulated annealing. The modified BP can adapt its learning rate for each individual in a population. Simulated annealing is used to avoid the local minima problem of the BP algorithm. A distinct feature of hybrid training is that it is partial. This means that EPNet does not train each individual until it converges rather it trains the individual for a fixed number of epochs in each generation. The number of epochs is a user specified parameter. The main reason behind partial training is to increase the computational efficiency in fitness evaluation. Hybrid training is always attempted before any architectural mutations (namely, node/connection deletion/addition) because the latter cause larger changes in ANN behavior.

Node deletion in EPNet is done totally at random – in other words, a node is selected uniformly at random for deletion. However, the other three

architectural mutations are not uniformly random. Connection deletion and addition use a nonuniform probability distribution to decide which connection to delete or add based on the importance of the connection [17,61]. Node addition is achieved by splitting an existing node [39], rather than by introducing a random one. The two nodes obtained by splitting an existing node have the same connections as the existing node. The weights of these new nodes have the following values:

$$\begin{aligned} w_{ij}^1 &= w_{ij}^2 = w_{ij}, & i \geq j \\ w_{ki}^1 &= (1 + \alpha)w_{ki} & i < k \\ w_{ki}^2 &= -\alpha w_{ki} & i < k \end{aligned} \quad (1)$$

where \mathbf{w} is the weight vector of the existing node i , \mathbf{w}^1 and \mathbf{w}^2 are the weight vectors of the new nodes, α is a mutation parameter which may take either a fixed or random value, and j and k indicate nodes which have a connection to/from node i . This method helps greatly in maintaining the behavioral link between the parent and its offspring. It also reduces blindness caused by a random node.

To improve the generalization ability of evolved ANNs, validation sets are used in EPNet. Each individual (that is, ANN) is trained on a training set, but it is evaluated on a validation set. All fitness values are calculated based on the validation, not the training set. After the simulated evolution, all the individuals in the last generation are trained further by the modified BP on the combined training and validation set. A second validation set is used to stop this training and select the best individual as the output of EPNet.

2.2 Combination Methods

The four combining methods used in EPNet are all linear. The simplest linear combination method is majority voting. That is, the output of the most number of EANNs will be the output of the ensemble. If there is a tie, the output of the EANN (among those in the tie) with the lowest error rate on the validation set will be selected as the ensemble output. The ensemble in this case is the whole population. All individuals in the last generation participate in voting. The greatest advantage of majority voting is its simplicity. However, the problem of majority voting is that it treats all individuals equally though they are not equally good.

One way to consider differences among individuals without involving much extra computational cost is to use the fitness information to compute a weight for each individual. The rank-based linear combination method is such a scheme that puts weight on each ANN in the population based on their fitness values. More specifically, we can use rankings to generate weights for each EANN in combining the ensemble output. That is, given N sorted EANNs

with an increasing error rate, where N is the population size, and their outputs o_1, o_2, \dots, o_N , then the weight for the i th EANN is:

$$w_i = \frac{\exp(\beta(N + 1 - i))}{\sum_{j=1}^N \exp(\beta_j)} \quad (2)$$

where β is a scaling factor. The ensemble output is:

$$O = \sum_{j=1}^N w_j o_j. \quad (3)$$

One of the well-known algorithms for learning linear combination weights (that is, one-layer linear networks) is the RLS algorithm [38]. The idea behind RLS is to minimize a weighted least squares error. The benefit of using the RLS algorithm is that it is computationally efficient due to its recursive nature. The detailed description of the RLS algorithm implemented here can be found in [38].

In the above three combination methods, all the individuals in the last generation were used in forming ensembles. It is interesting to investigate whether one can reduce the size of the ensembles without too much increase in testing error rates. Such investigation can provide some hints on whether all the individuals in the last generation will contain some useful information and shed some light on the importance of a population in evolutionary learning. As the space of possible subsets is very large ($2^N - 1$) for a population of size N , it is impractical to use exhaustive search to find an optimal subset. Instead, a genetic algorithm (GA) [21] is used to search for a near-optimal subset [62]. The weights for each EANN in each subset were determined by the same RLS algorithm [38] as used in the previous scheme.

2.3 Experimental Studies

EPNet was applied on three real-world problems. They were Australian credit card, diabetes and heart disease. The data sets for these problems were obtained from the UCI machine learning repository [8]. There are 690 examples in the Australian credit card data set. The problem is to assess applications for a credit card based on a number of attributes; the 14 attributes include six numeric values and eight discrete ones. The output has two classes.

The diabetes data is also a two class problem. It has 500 examples of class 1 and 268 of class 2. There are eight attributes for each example. The data set is one of the most difficult problems in machine learning due to many missing attributes. The aim of the heart problem is to predict the presence or absence of heart disease given the results of various medical tests carried out on a patient. The data set of the heart problem has 13 attributes, which

have been extracted from a larger set of 75. The description of the extraction process can be found in [62].

Two validation sets were used in all experiments. One validation set, V-set 1, was used in the fitness evaluation. The other validation set, V-set 2, was used in further training of **EPNet**. The best individual with the minimum error rate on V-set 2 was chosen as the final result. If there was a tie, the individual with the minimum error rate on the combined training set and V-set 1 was the final result. If a tie still existed, the individual with the minimum error on the combined training set and V-set 1 would be the final result. The final individual was then tested on an unseen testing set.

Experimental Setup

For all experiments, each data set was randomly partitioned into four subsets, a training set, V-set 1, V-set 2, and a testing set. According to suggestions provided in [43, 44] to produce results for ANNs, the size of the training set, V-set 1, V-set 2, and testing set were chosen to be 50, 12.5, 12.5, and 25% of all examples, respectively, in a data set. The input attributes used for all problems were re-scaled to between 0.0 and 1.0 by a linear function. The output attributes were encoded using a 1-of- c output representation for c classes. The winner-takes-all method was used to determine the output of the ANNs. In this method, the output with the highest activation designates the class.

The same parameters were used for all data sets. These were as follows: population size (20); maximum number of generations (100); initial number of hidden nodes (2–8, which means the number of hidden nodes in any initial individual was generated at random between 2 and 8); initial connection density (0.75, which means the probability of having a connection between two nodes is 75%; the constraint of feedforward ANNs cannot be violated of course); initial learning rate (0.2); the number of mutated hidden nodes (1, which means only one node would be deleted/added in each mutation); and the number of mutated connections (1–3, which means the number of mutated connections is between 1 and 3). These parameters were selected after a very modest search. It was found that **EPNet** was not very sensitive to these parameters.

Results

Table 1 summarizes the results of [62]. The best individual in the last generation and the ensemble formed by the four combining methods are presented in the table. The majority voting method outperformed the single best individual on two out of three problems. This is rather surprising since majority voting did not consider the differences among individuals. It performed worse than the best individual on the heart disease problem probably because it treated all individuals in the population equally. The t -test comparing the

Table 1. Testing accuracies of the best individual in a population and ensemble formed from individuals in the population by using majority voting, the RLS algorithm [38] and optimal subset. The results were averaged over 30 independent runs. Mean, SD, Min, and Max indicate the mean value, standard deviation, minimum and maximum value, respectively (Note that the results in this table have been summarized from [62])

Problem		Best individual	Rank-based	Error rate majority voting	RLS algorithm	Optimal subset
Credit card	Mean	0.100	0.095	0.095	0.093	0.095
	SD	0.013	0.012	0.012	0.011	0.012
	Min	0.081	0.070	0.076	0.076	0.070
	Max	0.128	0.116	0.122	0.116	0.116
Diabetes	Mean	0.232	0.225	0.222	0.226	0.222
	SD	0.018	0.023	0.022	0.021	0.023
	Min	0.198	0.172	0.172	0.193	0.182
	Max	0.271	0.271	0.255	0.260	0.260
Heart	Mean	0.154	0.154	0.167	0.151	0.164
	SD	0.028	0.031	0.024	0.033	0.030
	Min	0.103	0.088	0.132	0.088	0.118
	Max	0.235	0.235	0.235	0.221	0.221

best individual to the ensemble formed by majority voting indicates that the ensemble is better than the best individual for the credit card and diabetes problems and worse for the heart disease problem at 0.05 level of significance.

It is clear from Table 1 that the results of the ensemble formed by the rank-based linear method are either better than or as good as those produced by the best individual. The *t*-test comparing the best individual to the ensemble indicates that the ensemble is better than the best individual for the credit card and diabetes problems at the 0.05 level of significance. The ensemble also outperforms the best individual for the heart disease problem (no statistical significance, however).

The ensemble formed by the RLS algorithm [38] is better than the best individual for all three problems (Table 1). The results also indicate that a better combination method can produce better ensembles. In fact, the RLS algorithm is one of the recommended algorithms for performing linear combinations [24, 42]. However, other algorithms [7] can also be used. The *t*-test comparing the best individual to the ensemble formed by the RLS algorithm indicates that the ensemble is better than the best individual at the 0.05 level of significance for the credit card and diabetes problems, and better at the 0.25 level of significance for the heart disease problem.

The ensemble formed by the subset method is also better than the best individual for the credit card and diabetes problems at the 0.10 and 0.005 levels of significance, respectively. It is worse than the best individual for

the heart disease problem at the 0.05 level of significance. This worse result might be caused by the small number of generations (only 50) used in the experiments. A large number could probably produce better results, but would increase the search time.

All the above results indicate that a population contains more information than any individual in it. Such information can be used effectively to improve generalization of the learning systems. In a sense, the use of population information provides a natural way of evolving modular ANNs, where each module is an individual in the population. However, no special considerations were made in the evolution of ANNs about modularization in *EPNet*. If the evolution of modular ANNs could be encouraged in the evolutionary process, one can expect to improve the results further. One way to encourage modularization is by speciation. That is, we can use techniques like fitness sharing [14,21] to automatically form species in a population. Each species will be a specialist in dealing with part of a complex problem and will be treated as a module of the final system. In this case, modules are evolved specifically for an integrated system. Co-evolutionary learning is usually used in evolving modular systems. This idea has been tested successfully in a rule-based system [15] and described in the next Section.

3 Automatic Modularization

Many problems are too large and too complex to be solved by a monolithic system. Divide-and-conquer has often been used to tackle such problems. The key issue here is how to divide a large problem into smaller sub-problems. Tedious trial-and-error processes have often been used by human experts in coming up with a good method for breaking up a large problem into smaller components that are easier to solve. However, it is possible to make use of evolutionary computation techniques to divide a large problem into simpler sub-problems automatically.

Darwen and Yao proposed a novel approach to automatic divide-and-conquer, known as *automatic modularization*, in evolving a rule-based system for playing iterated prisoner's dilemma games without any human intervention [15]. Their results have shown clearly that automatic modularization can be used to evolve an integrated rule-based system consisting of several sub-systems, each of which is specialized in dealing with certain aspects of a complex problem (for instance, iterated prisoner's dilemma games). Such sub-systems can be regarded as modules of the integrated system (hence 'automatic modularization').

The main idea behind automatic modularization is a speciated evolutionary algorithm. In the case of evolving game-playing strategies for the iterated prisoner's dilemma games, each individual in the population is a rule-based system representing a strategy. The implicit fitness sharing scheme used in

the speciated evolutionary algorithm will encourage the evolution of species automatically in a population [15]. Each species can be regarded as a sub-system (module) in the integrated system, which is represented by the entire population. The experimental results have shown that automatic modularization can lead to substantial performance gain in evolving game-playing strategies for iterated prisoner's dilemma games [15].

Although the original work on automatic modularization was done using rule-based systems, the idea is equally applicable to neural networks, decision trees and other classifiers. [26] described the most recent work related to automatic modularization using neural networks.

4 Negative Correlation Learning

Although ANN ensembles perform better than single ANN in many cases, a number of issues need to be addressed when using ensembles. Two such important issues are the determination of an ensemble size and the maintenance of diversity among different ANNs in the ensemble. Both theoretical [27, 28] and empirical studies [40, 41] have shown that when individual ANNs are accurate and their errors are negatively correlated, improved performance can be obtained by combining the outputs of several ANNs. There is little to be gained by combining ANNs whose errors are positively correlated and are not accurate.

Liu and Yao proposed a new learning paradigm, called *negative correlation learning* (NCL), for training ANN ensembles. The essence of NCL is that it can produce negatively correlated ANNs for ensembles [33]. A number of works (for example, [13, 34, 37]) have utilized this feature in training ensembles. Unlike previous learning approaches for ANN ensembles, NCL attempts to train individual ANNs in an ensemble and combine them in the same learning process. In NCL, all the individual ANNs in the ensemble are trained simultaneously and interactively through the correlation penalty terms in their error functions. Rather than producing unbiased ANNs whose errors are uncorrelated, NCL can create negatively correlated networks to encourage specialization and cooperation among the individual ANNs.

Suppose that we have a training set

$$D = \{(\mathbf{x}(1), d(1)), \dots, (\mathbf{x}(N), d(N))\} \quad (4)$$

where $\mathbf{x} \in R^p$, d is a scalar, and N is the size of the training set. The assumption that the output d is a scalar has been made merely to simplify exposition of ideas without loss of generality. This Section considers estimating d by forming an ensemble whose output is a simple averaging of outputs of a set of ANNs

$$F(n) = \frac{1}{M} \sum_{i=1}^M F_i(n) \quad (5)$$

where M is the number of the individual ANNs in the ensemble, $F_i(n)$ is the output of ANN i on the n th training pattern, and $F(n)$ is the output of the ensemble on the n th training pattern.

NCL introduces a correlation penalty term into the error function of each individual network in the ensemble so that all the networks can be trained simultaneously and interactively on the same training data set D . The error function E_i for network i in negative correlation learning is defined by

$$\begin{aligned} E_i &= \frac{1}{N} \sum_{n=1}^N E_i(n) \\ &= \frac{1}{N} \sum_{n=1}^N \frac{1}{2} (F_i(n) - d(n))^2 + \frac{1}{N} \sum_{n=1}^N \lambda p_i(n) \end{aligned} \tag{6}$$

where $E_i(n)$ is the value of the error function of network i at presentation of the n th training pattern. The first term in the right side of Eqn. (6) is the empirical risk function of network i . The second term, p_i , is a correlation penalty function. The purpose of minimizing p_i is to negatively correlate each network's error with errors for the rest of the ensemble. The parameter $0 \leq \lambda \leq 1$ is used to adjust the strength of the penalty. The penalty function p_i has the form:

$$p_i(n) = (F_i(n) - F(n)) \sum_{j \neq i} (F_j(n) - F(n)) \tag{7}$$

The partial derivative of $E_i(n)$ with respect to the output of network i on the n th training pattern is

$$\begin{aligned} \frac{\partial E_i(n)}{\partial F_i(n)} &= F_i(n) - d(n) + \lambda \frac{\partial p_i(n)}{\partial F_i(n)} \\ &= F_i(n) - d(n) + \lambda \sum_{j \neq i} (F_j(n) - F(n)) \\ &= F_i(n) - d(n) - \lambda (F_i(n) - F(n)) \\ &= (1 - \lambda)(F_i(n) - d(n)) + \lambda (F(n) - d(n)) \end{aligned} \tag{8}$$

where we have made use of the assumption that $F(n)$ has constant value with respect to $F_i(n)$. The standard BP algorithm [46] has been used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual networks is performed simultaneously using Eqn. (8) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is called an 'epoch'.

NCL from Eqn. (8) is a simple extension to the standard BP algorithm. In fact, the only modification that is needed is to calculate an extra term of the form $\lambda(F_i(n) - F(n))$ for the i th network. From Eqns. (6)–(8), we may make the following observations:

1. During the training process, all the individual ANNs interact with each other through their penalty terms in the error functions. Each ANN i

minimizes not only the difference between $F_i(n)$ and $d(n)$, but also the difference between $F(n)$ and $d(n)$. That is, NCL considers errors that all other ANNs have learned while training an ANN.

2. For $\lambda = 0.0$, there are no correlation penalty terms in the error functions of the individual ANNs, and the individual ANNs are just trained independently. That is, independent training for the individual ANNs is a special case of NCL.
3. For $\lambda = 1$, from Eqn. (8) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - d(n) \quad (9)$$

Note that the empirical risk function of the ensemble for the n th training pattern is defined by

$$E_{ens}(n) = \frac{1}{2} \left(\frac{1}{M} \sum_{i=1}^M F_i(n) - d(n) \right)^2 \quad (10)$$

The partial derivative of $E_{ens}(n)$ with respect to F_i on the n th training pattern is

$$\begin{aligned} \frac{\partial E_{ens}(n)}{\partial F_i(n)} &= \frac{1}{M} \left(\frac{1}{M} \sum_{i=1}^M F_i(n) - d(n) \right) \\ &= \frac{1}{M} (F(n) - d(n)) \end{aligned} \quad (11)$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ens}(n)}{\partial F_i(n)} \quad (12)$$

The minimization of the empirical risk function of the ensemble is achieved by minimizing the error functions of the individual ANNs. From this point of view, NCL provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual ANNs.

4.1 Evolutionary Ensembles with Negative Correlation Learning

Based on NCL [33] and evolutionary learning, Liu and Yao proposed evolutionary ensembles with NCL (EENCL) to determine automatically the number of individual ANNs in an ensemble and to exploit the interaction between individual ANN design and their combination [34]. In EENCL, an evolutionary algorithm based on EP [18, 20] was used to search for a population of diverse individual ANNs for constructing ensembles. This means an evolutionary algorithm is used here for determining automatically the number of ANNs required for constructing an ensemble.

Two schemes are used in EENCL to maintain diversity in different individuals (that is, ANNs) of the population. They are fitness sharing [63] and

NCL [33]. The fitness sharing accomplishes speciation by degrading the raw fitness (in other words, the unshared fitness) of an individual according to the presence of similar individuals. If one training example is learned correctly by n individuals in a population, each of these n individuals receives fitness $1/n$, and the remaining individuals in the population receive fitness zero. Otherwise, all the individuals in the population receive fitness zero. This procedure is repeated for all examples in the training set. The fitness of an individual is then determined by summing its fitness over all training examples.

EENCL uses Gaussian mutation to produce offspring from parents, although non-Gaussian mutation such as Cauchy mutation [64] and Lévy mutation [32] can also be used. The mutation is carried out in two steps: (i) weight mutation, and (ii) further weight training. In the first step, n_b parent networks are selected at random to create n_b offspring. The parameter n_b is a parameter specified by a user. The probability for selecting a parent network is same. The following is used for weight mutation [34]:

$$w'_{ij} = w_{ij} + N(0, 1) \quad (13)$$

where w'_{ij} and w_{ij} denote the weights of offspring i and parent i , respectively, $i = 1, \dots, n_b$, j is the index number of weights. $N(0, 1)$ denotes a Gaussian random variable with mean zero and standard deviation one.

In the second step, the n_b offspring ANNs are further trained by NCL [33]. EENCL selects the fittest M ANNs from the union of M parents ANN and n_b offspring ANN. Here M is the number of individuals in the population. EENCL repeats the process of offspring generation and selection process for the g generation specified by a user.

A population of ANNs is found after the evolutionary process has finished. Now a question arises as to how to form the ensemble from a population of ANNs. The most convenient way is to use *all* ANNs – that is, the whole population in the last generation. The other way is to use a subset of population by selecting one representative from each species in the last generation. The species in the population can be obtained by clustering the individuals in the population using any clustering algorithm (such as the k-means algorithm) [35]. The latter may reduce the size of an ensemble without worsening its performance too much. In EENCL, these two approaches were used to form ensembles.

Three combination methods were used to determine the output of an ensemble from different ANNs used for forming the ensemble, these being simple averaging, majority voting and winner-takes-all. In simple averaging, the output of the ensemble is obtained by averaging the output of individual ANNs in the ensemble. In majority voting, the output of the greatest number of individual ANNs will be the output of the ensemble. If there is a tie, the

output of the ensemble is rejected. In winner-takes-all, the output of the ensemble is only decided by the individual ANN whose output has the highest activation.

4.2 Experimental Studies

EENCL was applied on two benchmark problems: the Australian credit card assessment problem and the diabetes problem. The n -fold cross-validation technique was used to divide the data randomly into n mutually exclusive data groups of equal size. In each train-and-test process, one data group is selected as the testing set, and the other $(n - 1)$ groups become the training set. The estimated error rate is the average error rate from these n groups. In this way, the error rate is estimated efficiently and in an unbiased way. The parameter n was set to be 10 for the Australian credit card data set, and 12 for the diabetes data set.

The same parameters were used for both problems. They are as follows: population size 25, number of generations 200, reproduction block size n_b 2, strength parameter λ for NCL [33] 0.75, number of training epochs 5, minimum number of cluster sets 3, and the maximum number of cluster sets 25. The ANNs used in the population are multilayer perceptrons with one hidden layer and five hidden nodes.

Results

All the results presented in this Section are summarized from results presented in [34]. Table 2 shows the results of EENCL for the two data sets, where the

Table 2. Accuracy rates of EENCL for the Australian credit card and the diabetes data sets. The results are averaged on 10-fold cross-validation for the Australian credit card data set, and 12-fold cross-validation for the diabetes data set. The *Mean*, *SD*, *Min*, and *Max* indicate the mean value, standard deviation, minimum, and maximum value, respectively (Note that the results presented in this table have been summarized from [34])

		Simple averaging		Majority voting		Winner-takes-all	
		Training	Testing	Training	Testing	Training	Testing
Credit card	Mean	0.910	0.855	0.917	0.857	0.887	0.865
	SD	0.010	0.039	0.010	0.039	0.007	0.028
	Min	0.897	0.797	0.900	0.812	0.874	0.812
	Max	0.924	0.913	0.928	0.913	0.895	0.913
Diabetes	Mean	0.795	0.766	0.802	0.764	0.783	0.779
	SD	0.007	0.039	0.007	0.042	0.007	0.045
	Min	0.783	0.703	0.786	0.688	0.774	0.703
	Max	0.805	0.828	0.810	0.828	0.794	0.844

Table 3. Accuracy rates of the ensemble formed by the representatives from species. The results are averaged on 10-fold cross-validation for the Australian credit card data set, and 12-fold cross-validation for the diabetes data set. *Mean*, *SD*, *Min*, and *Max* indicate the mean value, standard deviation, minimum, and maximum value, respectively (Note that the results presented in this table have been summarized from [34])

Accuracy rate	Credit card		Diabetes	
	Training	Testing	Training	Testing
Mean	0.887	0.868	0.783	0.777
SD	0.004	0.030	0.009	0.042
Min	0.881	0.812	0.770	0.719
Max	0.890	0.913	0.798	0.844

ensembles were formed using the whole population in the last generation. The *accuracy rate* refers to the percentage of correct classifications produced by EENCL. Comparing the accuracy rates obtained by the three combination methods, winner-takes-all outperformed simple averaging and majority voting on both problems. In simple averaging and majority voting, all individuals are treated equally. However, not all individuals are equally important. Because different individuals created by EENCL were able to specialize to different parts of the testing set, only the outputs of these specialists should be considered to make the final decision of the ensemble for this part of the testing set. The winner-takes-all combination method performed better because there are good and poor individuals for each pattern in the testing set, and winner-takes-all selects the best individual.

The results of the ensemble formed by the representatives from species are given in Table 3. The combination method used is winner-takes-all. The *t*-test statistics comparing the accuracies of the ensembles using the representatives from species to the ensembles using the whole population are 0.80 for the Australian credit card data set, and -0.36 for the diabetes data set. No statistically significant difference was observed between them for both data sets ($p > 0.05$), which implies that the ensemble does not have to use the whole population to achieve good performance. The size of the ensemble can be substantially smaller than the population size. The reduction in the size of the ensembles can be seen from Table 4, which gives the sizes of the ensembles using the representatives from species.

5 Constructive Approaches to Ensemble Learning

Determination of ensemble size by an evolutionary approach was presented in Sect. 4.1. The problem with ENNCL [34] is that it only determines the number of individual ANNs in the ensemble automatically, but the sizes of the ANNs

Table 4. Sizes of the ensembles using the representatives from species. The results are averaged on 10-fold cross-validation for the Australian credit card data set, and 12-fold cross-validation for the diabetes data set. *Mean*, *SD*, *Min*, and *Max* indicate the mean value, standard deviation, minimum, and maximum value, respectively (Note that the results presented in this table have been summarized from [34])

	Ensemble size			
	Mean	SD	Min	Max
Credit card	13.2	7.8	5	25
Diabetes	16.3	6.4	5	25

need to be specified by the user. It is well known that the accuracy of ANNs is greatly dependent on their size. This means random selection of the ANN sizes may hurt the ensemble performance. This is because the performance of ensembles not only depends on the diversity of individual ANNs but also on ANN accuracy. The aim of this Section is to present a constructive algorithm for training cooperative neural-network ensembles (CNNEs) [37].

Unlike most previous studies on training ensembles, CNNE puts emphasis on both accuracy and diversity among individual ANNs in an ensemble. It uses a constructive approach to determine automatically the number of ANNs in an ensemble and of hidden neurons in the ANNs. The automatic determination of hidden neurons ensures accuracy of individual ANNs in designing the ensemble. CNNE trains each individual ANN with a different number of training epochs, which is determined automatically by its training process. Like ENNCL [34], it also uses NCL [33] to train individual ANNs so that they can learn different aspects or parts of the training data. The use of NCL and different training epochs reflects CNNE’s emphasis on diversity among individual ANNs in the ensemble.

A number of issues – such as the number of individual ANNs in an ensemble, the number of hidden nodes in the ANNs, and the number of epochs required for training ANNs – need to be addressed when designing ensembles. This means the design of ANN ensembles could be formulated as a multi-objective optimization problem. CNNE uses a simple approach based on incremental training in designing ensembles. It tries to minimize the ensemble error first by training a minimal ensemble architecture, then by adding several hidden nodes one by one to existing ANNs, and lastly by adding new ANNs one by one. The minimal ensemble architecture consists of two ANNs with one hidden layer in each ANN and one node in the hidden layer.

The main structure of CNNE is shown in Fig. 2, and a detailed description can be found in [37]. It is not clear from the figure when and how to add hidden nodes and individual ANNs to the ensemble architecture. CNNE uses

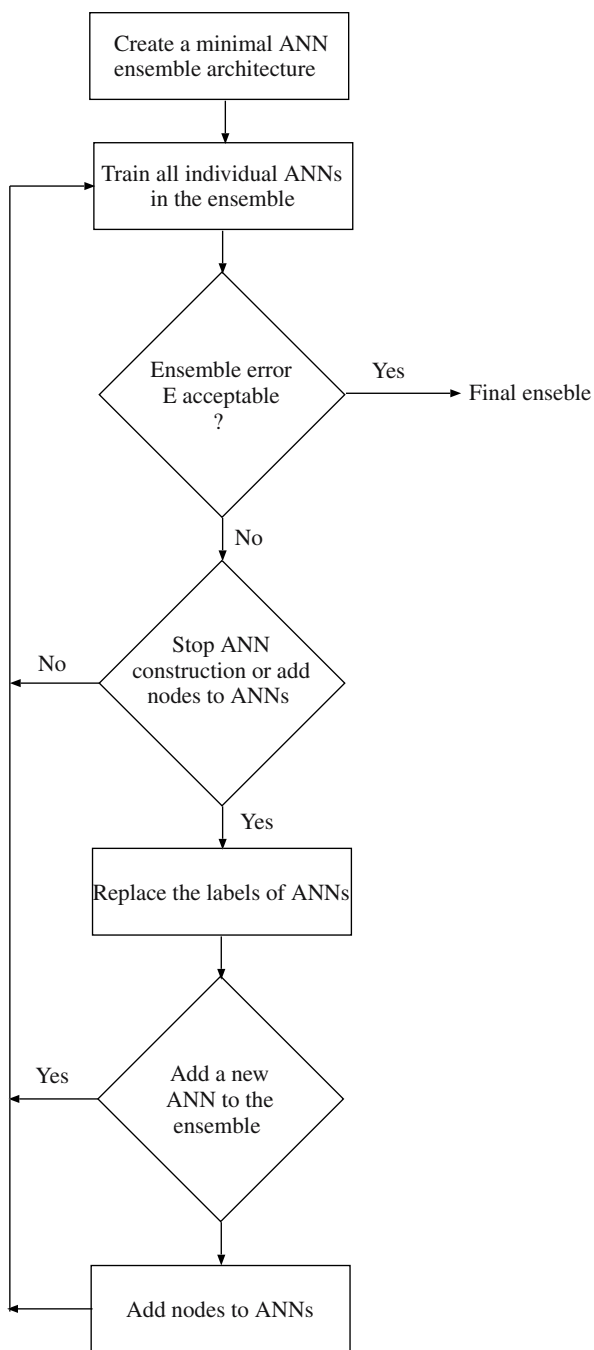


Fig. 2. The major steps of CNNE [37] © 2003

a simple criteria for adding hidden nodes and ANNs, based on the contribution of ANNs to the ensemble. The following is used to determine the contribution:

$$C_i = 100 \left(\frac{1}{E} - \frac{1}{E^i} \right) \quad (14)$$

where E is the ensemble error *including* individual ANN i , and E^i is the ensemble error *excluding* individual ANN i . CNNE adds hidden nodes to an individual ANN when its contribution to the ensemble does not improve much after a certain amount of training. An individual ANN is added to the ensemble when adding several hidden nodes to the previously added ANN have failed to reduce the ensemble error significantly. When a new ANN is added to the ensemble, CNNE stops the construction process of the previously added ANN. This means that no hidden node will be added to the previously added ANN in future.

5.1 Experimental Studies

CNEE was applied to seven benchmark problems: the Australian credit card assessment problem, the breast cancer problem, the diabetes problem, the glass problem, the heart disease problem, the letter recognition problem, and the soybean problem. The data sets representing these problems were obtained from the UCI machine learning benchmark repository. For all our experiments, each data set was partitioned into three subsets: a training set, a validation set and a testing set. The size of the training set, validation set, and testing set was 50, 25, and 25% of all examples, respectively. The only exception is the letter data set, where 16,000 and 2,000 examples were randomly selected from 20,000 examples for the training and validation sets, and the remaining 2,000 examples were used for the testing set.

Initial connection weights for individual ANNs in an ensemble were randomly chosen in the range -0.5 to 0.5 . The learning rate and momentum for training individual ANNs were chosen in the range 0.10 – 0.50 and 0.5 – 0.9 , respectively. The number of training epochs for partial training of individual ANNs was chosen between 5 and 25. The number of hidden nodes used for halting the construction of individual ANNs was chosen between one and five. The threshold value ϵ was chosen between 0.10 and 0.20 . These parameters were chosen after some preliminary experiments; they were not meant to be optimal. The parameter λ used to adjust the strength of the penalty term was set to 1.0 .

Results

Table 5 show the results of CNNE over 30 independent runs on the seven different problems. The results presented in this table are summarized from [37]. The error rates in the table refer to the percentage of wrong classifications

Table 5. Architectures and accuracies of ensembles produced by CNNE for seven different classification problems. The results were averaged over 30 independent runs. M and N indicate the number of ANNs in an ensemble and of hidden nodes in an ANN, respectively (Note that the results presented in this table have been summarized from [37])

	$\tau = 10, m_h = 4$			$\tau = 10, m_h = 2$			$\tau = 15, m_h = 2$		
	M	N	Error rate	M	N	Error rate	M	N	Error rate
Credit card	6.5	5.3	0.090	7.8	4.7	0.092	7.4	4.3	0.091
Breast cancer	3.9	3.6	0.015	4.8	2.9	0.013	4.5	2.5	0.012
Diabetes	4.7	4.5	0.201	6.5	3.4	0.198	6.2	3.2	0.196
Glass	4.9	4.6	0.261	6.2	3.8	0.268	6.0	3.5	0.258
Heart	4.6	6.5	0.140	5.5	4.9	0.134	5.8	4.2	0.138
Letter	11.6	10.6	0.067	15.3	8.5	0.062	13.9	8.1	0.060
Soybean	5.3	5.5	0.081	7.1	4.2	0.076	6.8	3.8	0.078

produced by the trained ensemble on the testing set. M and N represent the number of ANNs in an ensemble and of hidden nodes in an ANN, respectively.

It can be observed from Table 5 that the ensemble architectures learned by CNNE were influenced by the values of user specified parameters τ and m_h . For example, for the credit card problem, when $\tau = 10$ and $m_h = 4$ the average number of individual ANNs and hidden nodes were 6.5 and 5.3, respectively, and the average number of individual ANNs and hidden nodes were 7.8 and 4.7, respectively, when $\tau = 10$ and $m_h = 2$. This indicates that for the same value of τ the number of individual ANNs in an ensemble increases when the number of hidden nodes in the ANNs decreases. This is reasonable because a small ANN has only a limited processing power. CNNE added more ANNs to the ensemble when the size of individual ANNs was small. However, it is worth noting that the testing error rate remained roughly the same for different parameter settings and different ensemble architectures. The choice of different parameters did not affect the performance of the learned ensembles much, which is a highly desirable feature for any ANN training algorithm.

The ability of CNNEs to automatically construct different ensembles for different problems can be clearly seen from Table 5. CNNE produced large ensembles for the letter problem, which is large in comparison with the other problems here, and smaller ensembles for other problems. However, there are other factors in addition to the size of training sets – for example the complexity of the given problem and noise in the training set – that influence the ensemble architecture. For instance, the number of training examples for the diabetes problem was 384, while it was 342 for the soybean problem. In terms of average results, CNNE produced ensembles that had 4.7 individual ANNs with 4.5 hidden nodes for the diabetes problem, while it produced ensembles

that had 5.3 individual ANNs with 5.5 hidden nodes for the soybean problem. In general, all the above examples illustrated the same point, namely CNNEs ability to determine the ensemble automatically for different problems without human intervention.

6 Multi-Objective Approaches to Ensemble Learning

As mentioned previously, ensemble learning could be formulated as a multi-objective optimization problem. The aim of this Section is to introduce multi-objective evolutionary approaches to ensemble learning. The idea of designing ANNs using a multi-objective evolutionary approach was first considered by [3], in which a new algorithm, called memetic Pareto artificial neural network (MPANN) is proposed for training ANNs. It combines a multi-objective evolutionary algorithm and a gradient-based local search in reducing network complexity and training error. MPANN was later applied for learning and formation of ANN ensembles with a different multi-objective formulation [4,5].

When a population of ANNs is evolved using a multi-objective evolutionary approach different ANNs in the population may be good for different objectives. This means we are getting a set of near optimal ANNs that can easily be used for constructing ensembles. In addition, the use of an evolutionary approach would speed up finding a set of near optimal solutions. This is because the evolutionary approach uses a multi-directional search scheme instead of a unidirectional search scheme as used by conventional approaches.

Recently Chandra and Yao proposed an algorithm, called diverse and accurate ensemble learning algorithm (DIVACE), that uses multi-objective evolutionary approach to ensemble learning [13]. DIVACE tries to find an optimum tradeoff between diversity and accuracy by treating them explicitly as multi-evolutionary pressures. [12] give a good account of why diversity is necessary in ANN ensembles, and present a taxonomy of methods that enforce it and which are used in practice. The evolutionary process of DIVACE is quite similar to the one used in pareto differential evolution [1] and in MPANN [4,5]. It also has three main components – namely fitness evaluation, selection and genetic operations, as with conventional EAs.

Fitness evaluation in DIVACE is not straightforward and is based on the non-dominated sorting procedure proposed by [49]. This sorting procedure can be described as follows: Let there are two solutions S_1 and S_2 for a given problem. A solution is considered optimal if it satisfies all n objective of the problem. According to [49], the solution S_1 is said to ‘dominate’ solution S_2 if S_1 is not worse than S_2 in all n objectives and S_1 is strictly better than S_2 in at least one objective. This is the concept of non-domination. After non-dominated sorting, a set of individuals is found which is better than the rest of the individuals in the population.

Since the evolutionary process of DIVACE is similar to one used in Pareto differential evolution [1] – a variant of differential evolution [50] – three parents are randomly selected from the non-dominated set for the crossover and mutation genetic operations. DIVACE incorporates the idea of simulated annealing, which makes the variance of the Gaussian distribution used for crossover adaptive. The following equations are used to produce offspring by crossover:

$$w_{hi} = w_{hi}^{\alpha_1} + N(0, \sigma^2)(w_{hi}^{\alpha_2} - w_{hi}^{\alpha_3}) \tag{15}$$

$$w_{oh} = w_{oh}^{\alpha_1} + N(0, \sigma^2)(w_{oh}^{\alpha_2} - w_{oh}^{\alpha_3}) \tag{16}$$

where w_{hi} and w_{oh} are the weights (input to hidden layer and hidden to output layer, respectively) of the child generated; α_1 , α_2 and α_3 indicate three parents. Mutation is applied on an offspring generated by crossover with probability $1/N$, where N is the size of the population. The following equations are used for mutation.

$$w_{hi} = w_{hi} + N(0, 0.1) \tag{17}$$

$$w_{oh} = w_{oh} + N(0, 0.1) \tag{18}$$

6.1 Experimental Studies

This Section presents some results obtained on testing DIVACE on the Australian credit card assessment and diabetes problems. The experimental setup is similar to that in [4, 5], in order to facilitate comparison with previous work and for consistency. We used 10-fold and 12-fold cross validation for the card and diabetes problems, respectively. Three combining methods – namely simple averaging, majority voting and winner-takes-all are used in the experiments.

Results

During the course of the evolutionary process, it was expected that each member in the Pareto (non-dominated) set (after every generation) would perform well on different parts of the training set. Table 6 shows the performance

Table 6. Performance (accuracy rates) of the ensemble formed using DIVACE on the Australian credit card assessment data set (the results in this table are summarized from [13])

	Simple averaging		Majority voting		Winner-takes-all	
	Training	Testing	Training	Testing	Training	Testing
Mean	0.872	0.862	0.867	0.857	0.855	0.849
SD	0.007	0.049	0.007	0.049	0.007	0.053
Max	0.884	0.927	0.879	0.927	0.864	0.927
Min	0.859	0.753	0.856	0.768	0.842	0.753

Table 7. Performance (accuracy rates) of the ensemble formed using DIVACE on the Diabetes data set (the results in this table are summarized from [13])

	Simple averaging		Majority voting		Winner-takes-all	
	Training	Testing	Training	Testing	Training	Testing
Mean	0.780	0.773	0.783	0.766	0.766	0.766
SD	0.006	0.050	0.005	0.057	0.017	0.049
Max	0.791	0.859	0.791	0.875	0.796	0.843
Min	0.768	0.687	0.772	0.671	0.730	0.671

accuracy of the formed ensemble on the Australian credit card assessment data set. Table 7 shows the same for the Diabetes data set. Good performance can be observed for the DIVACE algorithm.

7 Conclusions

Combining ANNs with evolutionary computation has been a popular topic since the late 1980s. While the early work tended to focus on evolving single ANNs, at the level of weights, architectures and learning rules, recent work has moved towards evolving ANN ensembles. This is a natural trend because it is often impractical to evolve or design a monolithic ANN when the problem to be solved becomes larger and more complex; a divide-and-conquer strategy must be used in practice. ANN ensembles can be regarded as an effective approach to implement the divide-and-conquer strategy in practice. Evolutionary computation provides a powerful method for evolving such ensembles automatically, including automatic determination of weights, individual ANN architectures and the ensemble structure. This Chapter has reviewed some of the latest developments in the area of evolving ANN ensembles.

Acknowledgements

Portions of this chapter originally appeared in X. Yao and Md. M. Islam, Evolving artificial neural network ensembles, *IEEE Computational Intelligence Magazine*, 3(1): 31–42, February 2008. Permission to reprint this material in the current Compendium is gratefully acknowledged.

References

1. Abbass HA, Sarker R, Newton C (2001) PDE: A Pareto-frontier differential evolution approach for multi-objective optimization problems. In: Kim J-H (ed.) *Proc. IEEE Conf. Evolutionary Computation (CEC2001)*, 27–30 May, Seoul, South Korea. IEEE Press, Piscataway, NJ: 971–978.

2. Abbass HA (2002) The self-adaptive Pareto differential evolution algorithm. In: Fogel DB, El-Sharkawi MA, Yao X, Greenwood G, Iba H, Marrow P, Shackleton M (eds.) *Proc. IEEE Conf. Evolutionary Computation (CEC2002)*, 12–17 May, Honolulu, HI. IEEE Press, Piscataway, NJ: 831–836.
3. Abbass HA (2003) Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11): 2705–2726.
4. Abbass HA (2003) Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization. In: Sarker R, Reynolds R, Abbass H, Tan KC, McKay B, Essam D, Gedeon T (eds.) *Proc. IEEE Conf. Evolutionary Computation (CEC2003)*, 8–12 December, Canberra, Australia. IEEE Press, Piscataway, NJ: 2074–2080.
5. Abbass HA (2003) Pareto neuro-ensemble. In: Gedeon TD, Chun L, Fung C (eds.) *Proc. 16th Australian Joint Conf. Artificial Intelligence*, 3–5 December, Perth, Australia. Springer-Verlag, Berlin: 554–566.
6. Angeline PJ, Sauders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks*, 5(1): 54–65.
7. Baldi PF, Hornik K (1995) Learning in linear neural networks: a survey. *IEEE Trans. Neural Networks*, 6(4): 837–858.
8. Blake C, Merz C *UCI repository of machine learning databases*. (available online at <http://www.ics.uci.edu/mllearn/MLRepository.html> – last accessed September 2007).
9. Belew RK, McInerney J, Schraudolph NN (1991) Evolving networks: using genetic algorithm with connectionist learning. *Technical Report CS90-174 (revised)*, Computer Science and Engineering Department (C-014), University of California, San Diego, February.
10. Bollé D, Dominguez DRC, Amari S (2000) Mutual information of sparsely coded associative memory with self-control and ternary neurons. *Neural Networks*, 1: 452–462.
11. Brown G, Wyatt JL (2003) Negative correlation learning and the ambiguity family of ensemble methods. In: Windeatt T, Roli F (eds.) *Proc. Intl. Workshop Multiple Classifier Systems*, 11–13 June, Guildford, UK. Springer-Verlag, Berlin: 266–275.
12. Brown G, Wyatt JL, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. *J. Information Fusion*, 6: 5–20.
13. Chandra A, Yao X (2006) Ensemble learning using multi-objective evolutionary algorithms. *J. Mathematical Modeling and Algorithms*, 5(4): 417–445.
14. Darwen PJ, Yao X (1996) Every niching method has its niche: fitness sharing and implicit sharing compared. In: Ebeling W, Rechenberg I, Schwefel H-P, Voight H-M (eds.) *Parallel Problem Solving from Nature (PPSN) IV*, 22–26 September, Berlin, Germany. Lecture Notes in Computer Science 1141. Springer-Verlag, Berlin: 398–407.
15. Darwen PJ, Yao X (1997) Speciation as automatic categorical modularization. *IEEE Trans. Evolutionary Computation*, 1: 101–108.
16. Dietterich TG (1998) Machine-learning research: four current directions. *AI Magazine*, 18(4): 97–136.
17. Finnoff W, Hergent F, Zimmermann HG (1993) Improving model selection by nonconvergent methods. *Neural Networks*, 6: 771–783.
18. Fogel LJ, Owens AJ, Walsh MJ (1966) *Artificial Intelligence Through Simulated Evolution*. Wiley, New York, NY.

19. Fogel GB, Fogel DB (1995) Continuous evolutionary programming: analysis and experiments. *Cybernetic Systems*, 26: 79–90.
20. Fogel DB (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ.
21. Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.
22. Hancock PJB (1992) Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In: Whitley D, and Schaffer JD (eds.) *in Proc. Intl. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92)*, 6 June, Maryland, IEEE Computer Society Press, Los Alamitos, CA: 108–122.
23. Hansen LK, Salamon P (1990) Neural network ensembles. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(10): 993–1001.
24. Hashem S (1993) Optimal linear combinations of neural networks. *PhD dissertation*. School of Industrial Engineering, Purdue University, West Lafayette, IN, December.
25. Deb K (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK.
26. Khare V, Yao X, and B. Sendhoff B (2006) Multi-network evolutionary systems and automatic problem decomposition. *Intl. J. General Systems*, 35(3): 259–274.
27. Krogh A, Vedelsby J (1995) Neural network ensembles, cross validation, and active learning. *Neural Information Processing Systems*, 7: 231–238.
28. Krogh A, Sollich P (1997) Statistical mechanics of ensemble learning. *Physics Reviews E*, 55: 811–825.
29. Kwok TY, Yeung DY (1997) Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Trans. Neural Networks*, 8: 630–645.
30. Kwok TY, Yeung DY (1997) Objective functions for training new hidden units in constructive neural networks. *IEEE Trans. Neural Networks*, 8: 1131–1148.
31. Lehtokangas M (1999) Modeling with constructive backpropagation,” *Neural Networks*, 12: 707–716.
32. Lee CY, Yao X (2004) Evolutionary programming using the mutations based on the Lévy probability distribution. *IEEE Trans. Evolutionary Computation*, 8(1): 1–13.
33. Liu Y, Yao X (1999) Ensemble learning via negative correlation,” *Neural Networks*, 12: 1399–1404.
34. Liu Y, Yao X, Higuchi T (2000) Evolutionary ensembles with negative correlation learning. *IEEE Trans. Evolutionary Computation*, 4(4): 380–387.
35. MacQueen J (1967) Some methods for classification and analysis of multivariate observation. In: *Proc. 5th Berkely Symp. Mathematical Statistics and Probability*, Berkely, CA, University of California Press, 1: 281–297.
36. Mahfoud SW (1995) Niching methods for genetic algorithms. *PhD Thesis*, Department of General Engineering, University of Illinois, Urbana-Champaign, IL.
37. Monirul Islam M, Yao X, Murase K (2003) A constructive algorithm for training cooperative neural network ensembles. *IEEE Trans. Neural Networks*, 14: 820–834.
38. Mulgrew B, Cowan CFN (1988) *Adaptive Filters and Equalizers*. Kluwer, Boston, MA.

39. Odri SV, Petrovacki DP, Krstonosic GA (1993) Evolutional development of a multilevel neural network. *Neural Networks*, 6(4): 583–595.
40. Opitz DW, Shavlik JW (1996) Generating accurate and diverse members of a neural-network ensemble. *Neural Information Processing Systems*, 8: 535–541.
41. Opitz D, Maclin R (1999) Popular ensemble methods: an empirical study. *J. Artificial Intelligence Research*, 11: 169–198.
42. Perrone MP (1993) Improving regression estimation: averaging methods for variance reduction with extensions to general convex measure optimization. *PhD Dissertation*, Department of Physics, Brown University, Providence, RI, May.
43. Prechelt L (1994) Proben1-A set of neural network benchmark problems and benchmarking rules. *Technical Report 21/94*, Fakultät für Informatik, University of Karlsruhe, Germany, September.
44. Prechelt L (1995) Some notes on neural learning algorithm benchmarking. *Neurocomputing*, 9(3): 343–347.
45. Rissanen J (1978) Modeling by shortest data description. *Automatica*, 14: 465–471.
46. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. In: Rumelhart DE, McClelland JL (eds.) *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, I*. MIT Press, Cambridge, MA: 318–362.
47. Sharkey AJC (1996) On combining artificial neural nets. *Connection Science*, 8(3/4): 299–313.
48. Schaffer JD, Whitley D, Eshelman LJ (1992) Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: Whitley D, Schaffer JD (eds.) *Proc. Intl. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92)*, 6 June, Maryland. IEEE Computer Society Press, Los Alamitos, CA: 1–37.
49. Srinivas N, Deb K (1994) Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, 2(3): 221–248.
50. Storn R, Price K (1996) Minimizing the real functions of the ICEC'96 contest by differential evolution. In: Fukuda T, Furuhashi T, Back T, Kitano H, Michalewicz (eds.) *Proc. IEEE Intl. Conf. Evolutionary Computation*, 20–22 May, Nagoya, Japan. IEEE Computer Society Press, Los Alamitos, CA: 842–844.
51. Syswerda G (1991) A study of reproduction in generational and steady state genetic algorithms. In: Rawlins GJE (ed.) *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA: 94–101.
52. Yao X (1991) Evolution of connectionist networks. In: *Proc. Intl. Symp. AI, Reasoning & Creativity*, Griffith University, Queensland, Australia, 49–52.
53. Yao X (1993) An empirical study of genetic operators in genetic algorithms. *Microprocessors and Microprogramming*, 38: 707–714.
54. Yao X (1993) A review of evolutionary artificial neural networks. *Int. J. Intelligent Systems*, 8(4): 539–567.
55. Yao X (1993) Evolutionary artificial neural networks. *Int. J. Neural Systems*, 4(3): 203–222.
56. Yao X (1994) The evolution of connectionist networks. In: Dartnall T. (ed.) *Artificial Intelligence and Creativity*. Kluwer, Dordrecht, The Netherlands: 233–243.
57. Yao X (1995) Evolutionary artificial neural networks. In: Kent A, Williams JG (eds.) *Encyclopedia of Computer Science and Technology 33*, Marcel Dekker, New York, NY: 137–170.

58. Yao X, Shi Y (1995) A preliminary study on designing artificial neural networks using co-evolution. In: Toumodge S, Lee TH, Sundarajan N (eds.) *Proc. IEEE Intl. Conf. Intelligent Control Instrumentation*, 2–8 July, Singapore. IEEE Computer Society Press, Los Alamitos, CA: 149–154.
59. Yao X (1999) Evolving artificial neural networks. *Proc. IEEE*, 87: 1423–1447.
60. Yao X, Liu Y (1996) Ensemble structure of evolutionary artificial neural networks. In: Fukuda T, Furuhashi T, Back T, Kitano H, Michalewicz (eds.) *Proc. 1996 IEEE Intl. Conf. Evolutionary Computation (ICEC96)*, 20–22 May, Nagoya, Japan. IEEE Computer Society Press, Los Alamitos, CA: 659–664.
61. Yao X, Liu Y (1997) A new evolutionary system for evolving artificial neural networks. *IEEE Trans. Neural Networks*, 8(3): 694–713.
62. Yao X, Liu Y (1998) Making use of population information in evolutionary artificial neural networks. *IEEE Trans. Systems, Man, and Cybernetics B*, 28(3): 417–425.
63. Yao X, Liu Y, Darwen P (1996) ‘How to make best use of evolutionary learning’. In: Stocker R, Jelinek H, Durnota B (eds.) *Complex Systems: From Local Interactions to Global Phenomena*. IOS Press, Amsterdam, The Netherlands: 229–242.
64. Yao X, Liu Y, Lin G (1999) Evolutionary Programming Made Faster. *IEEE Trans. Evolutionary Computation*, 3(2): 82–102.
65. Yao X, Islam MM (2008) Evolving artificial neural network ensembles. *IEEE Computational Intelligence Magazine*, 3(1) (in press).

Resources

1 Key Books

Fogel DB (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ.

Haykin SY (1998) *Neural Networks: A Comprehensive Foundation (2nd ed)*. Prentice Hall, Englewood Cliffs, NJ.

Yao X (ed.) (1999) *Evolutionary Computation: Theory and Applications*. World Scientific, Singapore.

Sharkey AJC (ed.) (1999) *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, London, UK.

Kuncheva LI (2004) *Combining Pattern Classifiers Methods and Algorithms*. Wiley, Hoboken, NJ.

2 Key Survey/Review Articles

Kohonen T (1988) An introduction to neural computing. *Neural Networks*, 1(1): 3–16.

Yao X (1999) Evolving artificial neural networks. *Proc. IEEE*, 87(9): 1423–1447.

Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: a survey and categorization. *J. Information Fusion*, 6: 5–20.

3 Organizations, Societies, Special Interest Groups

IEEE Computational Intelligence Society

<http://www.ieee-cis.org/>

International Neural Network Society

<http://www.inns.org/>

European Neural Network Society

<http://www.snn.ru.nl/enns/>

4 Research Groups

Natural Computation Group at the University of Birmingham, UK

http://www.cs.bham.ac.uk/research/labs/natural_computation/

Machine Learning Research Group (MLRG) at the University of Wisconsin – Madison, USA.

<http://pages.cs.wisc.edu/~shavlik/mlrg/>

Neural Networks Research Group at the University of Texas, Austin

<http://nn.cs.utexas.edu/>

Computer Science and Artificial Intelligence Laboratory, MIT

<http://www.csail.mit.edu/index.php>

Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), UK

<http://www.cercia.ac.uk/>

5 Discussion Groups, Forums

Neural Network Forums

<http://www.makhfi.com/cgi-bin/teemz/teemz.cgi>

Neural Network Discussion Group

http://itmanagement.webopedia.com/TERM/N/neural_network.html

6 Key International Conferences and Workshops

Congress on Evolutionary Computation (CEC)

International Conference on Parallel Problem Solving from Nature (PPSN)

Neural Information Processing Systems (NIPS)

International Joint Conference on Neural Networks (IJCNN)

International Conference on Artificial Neural Networks (ICANN)

European Symposium on Artificial Neural Networks (ESANN)

International Conference on Neural Information Processing (ICONIP)

7 (Open Source) Software

Emergent Neural Network Simulation Software

http://neurobot.bio.auth.gr/archives/000116emergent_neural_network_simulation_software_formerly_pdp.php

Forecasting with artificial neural networks

<http://www.neural-forecasting.com/>

NeuroDimension

<http://www.nd.com/>

Neural archive at FuNet

<http://www.nic.funet.fi/>

The PDP++ Software

<http://www.cnbc.cmu.edu/Resources/PDP++//PDP++.html>

Amygdala for simulating spiking neural networks

<http://amygdala.sourceforge.net/>

8 Data Bases

Birmingham Repository: Evolutionary Computation Benchmarking Repository (EvoCoBR)

<http://www.cs.bham.ac.uk/research/projects/ecb/>

UCI repository of machine learning databases

<http://www.ics.uci.edu/~mllearn/MLRepository.html>

Neural Networks Databases – Benchmarks

<http://www.fizyka.umk.pl/neural/node12.html>

UCI KDD Archive

<http://kdd.ics.uci.edu/>

Netlib Repository

<http://www.netlib.org/>

Statlib

<http://lib.stat.cmu.edu/>

An Order Based Memetic Evolutionary Algorithm for Set Partitioning Problems

Christine L. Mumford

School of Computer Science, Cardiff University, 5 The Parade, Cardiff CF24 3AA, United Kingdom, C.L.Mumford@cs.cardiff.ac.uk

1 Introduction

Metaheuristic algorithms, such as simulated annealing, tabu search and evolutionary algorithms, are popular techniques for solving optimization problems when exact methods are not practical. For example, the run times required to obtain exact solutions to many common combinatorial problems grow exponentially (or worse) with the problem size, and as a result, solving even a relatively modest sized problem of this type may require many centuries of computation time, even on the fastest computer of the day. Such problems are known collectively as NP-Hard, and include the travelling salesman problem (TSP), which is probably the best known problem in the class. The present study will concentrate on a type of NP-Hard combinatorial problem known as the *set partitioning problem*. If we have n objects to partition into m sets, in such a way that each object must be assigned to exactly one set, it follows that there are m^n different ways that the n objects can be assigned to the m sets, for a straightforward unconstrained problem. It is instructive to note that every time the problem size of the set partitioning problem is increased by one object, the corresponding run time for an exhaustive search algorithm will increase by a factor of m , and thus the run time grows exponentially as the number of objects – n – increases. While it is true that much better exact methods than exhaustive search have been developed for most NP-Hard problems, the ‘growth factor’ remains exponential for the run time, and no one in history has so far managed to change that.

The main focus of the current Chapter is a new hybrid (or memetic) evolutionary algorithm specifically developed to solve set partitioning problems. This technique incorporates useful solution improvement heuristics into an evolutionary framework. New genetic operators have been devised to ensure that parent solutions are able to contribute useful features to their offspring, and a simulated annealing schedule has been adopted to help maintain a balance between quality and diversity within the population of candidate

solutions. The effectiveness and versatility of the new hybrid algorithm will be demonstrated by applying variations of the technique to hard literature benchmarks taken from three different set partitioning applications: graph coloring, bin packing and examination timetabling.

It is well known among researchers that effective evolutionary algorithms are notoriously difficult to devise for many types of problems, and set partitioning is particularly challenging, for reasons we will address later. At its best, an evolutionary algorithm will exploit its population structure to explore different parts of the search space for a problem simultaneously, combining useful features from different individuals during reproduction and producing new offspring solutions that may be better, on occasions, than either of that offspring's parents. At its worse, an evolutionary algorithm will take a lot longer than competitive methods to achieve very little. For most other methods the search is propagated through a single focal point. On the other hand, evolutionary algorithms progress from a population of points. The population has to 'earn its living', otherwise it becomes a burden rather than a bonus.

With the limitations and idiosyncrasies of evolutionary algorithms in mind, we will critically evaluate the various components of the new hybrid approach in the present study. In particular, we will endeavor to ensure that every part of the algorithm is making a useful contribution to its overall performance. The main goal is to present a new, and reasonably generic, order based framework that can be applied, with minimum adaptation, to a wide range of set partitioning problems. Although the exact choice of objective or fitness function will very likely depend on the specific problem, it is envisaged that problem-specific heuristics and costly backtracking will largely be avoided. Throughout the Chapter a tutorial approach is adopted, to aid newcomers to the field, and the main aspects of the algorithms and operators are illustrated using simple examples and carefully designed diagrams. However, it is hoped that the more experienced researcher will also find the Chapter of interest.

The remainder of the Chapter is organized as follows. We begin with introductory Sections on evolutionary algorithms, some of which may be safely skipped by the more knowledgeable reader. Section 2 outlines the historical development of evolutionary computing, and Sect. 3 introduces the main elements of a 'standard' genetic approach to problem solving. Section 4 describes the general features of an order based genetic algorithm and, together with Sect. 5 – on steady-state GAs – lays the foundations for the approach used for set partitioning in the present Chapter. Section 6 introduces the three test problems: graph coloring, bin packing, and timetabling, and this is followed by Sect. 7 which presents the reader with the main points that motivated the present study. The next Section covers the grouping and reordering heuristics of [9]. These heuristics are used in the present study to improve the effectiveness of the new crossover operators, and also provide useful local search capability in their own right. Section 9 details the main features of the new memetic approach, covering all the main aspects of the new genetic simulated

annealing algorithm (GSA). The Section also describes new crossover operators and defines the fitness functions that are used, as well as introducing the simulated annealing cooling schedule. Results for the literature benchmarks are presented in Sect. 10 and this is followed by a Chapter summary. Finally, URL links are provided to all the test data in the Resources Appendix.

2 A Brief History of Genetic Algorithms

Several groups of researchers, working independently in the 1950s and 60s, developed optimization techniques inspired by evolution and natural selection. [37] introduced *evolution strategies* and used the method to optimize real-valued parameters for designing aerofoils. [17] developed *evolutionary programming*, a technique that they used to evolve finite state machines. Most of these early techniques were developed with specific applications in mind, however. In contrast, John Holland made an extensive study of adaptation in the natural world and used his insight to create a sound theoretical framework from which his *genetic algorithms (GAs)* emerged [22]. Since these early days, interest in evolutionary-inspired algorithms has grown year by year, and numerous variants have appeared on the scene, some of them very different from anything conceived by Rechenberg, Fogel or Holland. For example, in the early 1990s, John Koza proposed *genetic programming*, an evolutionary style technique for evolving effective computer programs to undertake particular tasks.

Other popular paradigms to have been derived from the more generic approach include artificial life [28], evolvable hardware [21], ant systems [12] and particle swarms [26], to name but a few. In addition, there are many examples of hybrid (or memetic) approaches where problem-specific heuristics, or other techniques such as neural networks or simulated annealing, have been incorporated into a GA framework. Indeed, we shall make use of specialized heuristics and also simulated annealing to improve our results in the present Chapter. Thus, due to the growth in popularity of search and optimization techniques inspired by natural evolution during the last few decades, it is now common practice to refer to the field as *evolutionary computing* and to the various techniques as *evolutionary algorithms*. Inevitably, though, due to the overwhelming influence of John Holland, the term ‘genetic algorithm’ is frequently used interchangeably with the more generic term.

3 A Generic Genetic Algorithm

As suggested above, there is no rigorous definition of the term ‘genetic algorithm’ that everyone working in the field would agree on. There are, however, certain elements that GAs tend to have in common:

1. a population of chromosomes encoding (in string form) candidate solutions to the problem in hand,

2. a mechanism for reproduction,
3. selection according to fitness, and
4. genetic operators.

The *chromosomes* in the population of a GA usually take the form of strings, which may be encoded in binary, denary (that is, base-10), or in some other way. As an example, let us consider a simple optimization problem. Suppose we wish to maximize the following function:

$$f(x_1, x_2) = x_1^2 - x_2^2 + x_1x_2 \quad (1)$$

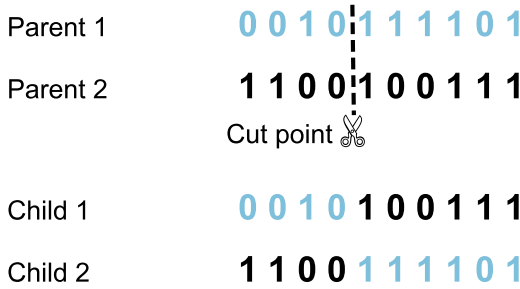
where x_1 and x_2 are both integers that can vary between 0 and 31. If we use a 5-bit binary representation for x_1 and x_2 , our GA strings would need to be 10 bits long. Thus, using this representation the string 0010111101 would encode $x_1 = 00101$, and $x_2 = 11101$, or 5 and 29 respectively in denary.

The *mechanism for reproduction* may consist simply of duplicating strings from the population. However, the choice of strings for reproduction is usually biased by some measure of *fitness* associated with each member of the population. The term ‘fitness’ refers to some estimate of quality allocated to each population member whereby ‘better’ individuals are assigned higher values than poorer individuals. In the above optimization problem the objective function – $f(x_1, x_2)$ – may be used directly as a fitness function. In other situations the allocation of fitness values may not be so straightforward: when solving a minimization problem, for example, or when dealing with qualitative data.

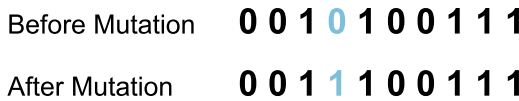
An essential feature of a successful GA implementation is that the average fitness value of a population should increase over time, reflecting an improving trend in the population. To facilitate this improvement we must somehow ensure that superior individuals have a better chance to contribute to future generations than do individuals with poorer fitness values. Probably the most popular way to drive this improvement is to use *selection probabilities* to bias the choice of parents for the next generation. Converting fitness values into probabilities for selection is usually a straightforward matter involving some simple arithmetic. Random numbers can then be generated and the parents of the next generation selected in accordance with a probability distribution derived from the individual fitness values of the population members. Due to the obvious analogy with a popular casino game, this process is widely known as *roulette wheel selection*.

Evolution cannot proceed by selection and reproduction alone, of course. It is essential that a GA is capable of occasionally producing new individuals that are better than their parents. In order to achieve this a mechanism to effect change is needed, and this is the role of *genetic operators*. Holland

a) One Point Crossover



b) Point Mutation



c) Inversion

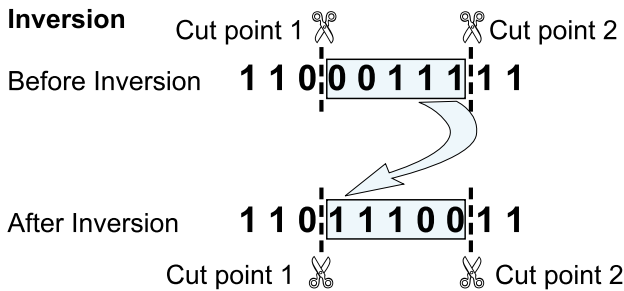


Fig. 1. Examples of genetic operators

describes three types of genetic operators: *crossover*, *mutation* and *inversion*. Examples to illustrate all of these are given in Fig. 1.

One point crossover, shown in Fig. 1(a), is a process involving two parental strings and begins with an alignment of the strings. A cut point is then chosen at random, and the parental material following the cut point is exchanged between the parents, giving rise to two children. Variants of simple crossover include two point and multi-point crossover where more cut points are selected. Point mutation is illustrated in Fig. 1(b). For a binary string a randomly selected bit is simply flipped. For denary or other encodings suitable mutation operators are chosen that produce very small changes. An example of inversion can be seen in Fig. 1(c). Here two cut points are selected at random on a single string, and the sub-string between the cut points is then inverted and replaced. Inversion, however, is not used in practice for simple bit strings

and denary chromosomes. Following reproduction, each genetic operator is applied with its own predetermined probability.

A simple generic GA is outlined in Algorithm 1. Adapting the generic model to make it effective for a particular problem, however, usually requires considerable effort, as well as a large measure of good luck! First, it is necessary to devise a suitable representation so that candidate solutions can be satisfactorily encoded as chromosomes. For many applications it is not immediately obvious how this can be done, and simple bit- or denary-valued strings may not be appropriate. Second, when deviating from a standard bit string or denary representation, special genetic operators may be needed, to avoid the production of infeasible offspring, say. Another potential difficulty is choosing a suitable fitness function for a given problem. Selection bias towards the better individuals needs to be strong enough to encourage ‘survival of the fittest’, but not so strong that all variability is quickly lost from the population (note: loss of diversity early on in the execution of a GA is often referred to as *premature convergence*). Without variability, nothing new can evolve. Finally, tuning the GA and determining the best values for various parameters – such as crossover and mutation rates – population size and stopping criteria, can be a very time consuming process.

Algorithm 1 A Generic Genetic Algorithm (GA)

Generate N random strings { N is the population size}
 Evaluate and store the fitness of each string

repeat

for $i = 1$ to $N/2$ **do**

 Select a pair of parents at random {The selection probability is in direct proportion to the fitness}

 Apply crossover with probability p_c to produce two offspring

if No crossover takes place **then**

 Form two offspring that are exact copies of their parents

 Mutate the two offspring at a rate of p_m at each locus

 Evaluate and store the fitness for the two offspring

 Replace the current population with the new population

until stopping condition satisfied

For further reading on genetic algorithms, I recommend the following introductory texts: [19], [32] or [33].

4 Order Based GAs

In Sect. 3 we saw how chromosomes can be encoded, as bit strings or decimal coded lists, and used to directly represent the variables of a problem. For many combinatorial problems, however, the random processes involved in assigning

values to the variables make direct representations rather prone to constraint violations, and as a result a GA can waste a vast amount of time evaluating infeasible solutions. Consider a set partitioning problem, which involves the assignment of each available item to exactly one set whilst strictly adhering to any problem-specific constraints. For example, with the bin packing problem various sized items are placed in a minimum number of equal sized bins. However, it is likely that assigning items to bins at random may result in some bins becoming overfull. The use of heavy penalty values to discourage the survival of illegal solutions is an approach favored by some researchers, while others prefer to use an heuristic repair mechanism to reduce or eliminate constraint conflicts, following the initial assignment of the GA. Yet another alternative is to use an order based approach with a greedy decoder. This approach will entirely avoid the issue of infeasibility. Starting with an arbitrary permutation of items, a greedy decoder will sequentially assign legal values to all the items in the list. Consider the graph coloring problem (GCP). This involves finding a minimum set of colors for the vertices of a given graph, so that no two adjacent vertices have the same color. If, for example, a GCP instance has n vertices, then order based chromosomes representing potential solutions will consist of permuted lists of the integers $\{1, 2, 3, \dots, n\}$. A decoder will start with the first vertex on the list and work through assigning, to each vertex in turn, the first available color from an ordered set (in other words, each color is identified by an integer label, 0, 1, 2, 3, ...), that is possible without causing conflicts.

Figure 2 illustrates possible encodings for a legally colored 12 node graph, with Fig. 2(b) and (c) showing direct and order based representations, respectively, for the example coloring in Fig. 2(a). It is easy to visualize how disruptive genetic operators could be if applied to a direct representation such as that shown in Fig. 2(b). An order based representation, on the other hand, will always produce a legal coloring when used in conjunction with a greedy decoder.

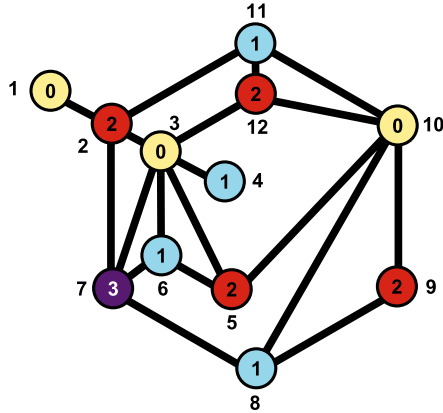
Unfortunately, standard crossover and mutation operators are not appropriate for order based representations, because such operators tend to destroy the permutation property and produce infeasible solutions. The problem with crossover is illustrated in the example below, which shows the production of infeasible offspring with duplicated and deleted values, following application of a two point crossover.

$$\begin{array}{l}
 A = 8 \ 7 \ 6 \ | \ 4 \ 1 \ 2 \ | \ 5 \ 3 \\
 B = 2 \ 5 \ 1 \ | \ 7 \ 3 \ 8 \ | \ 4 \ 6
 \end{array}$$

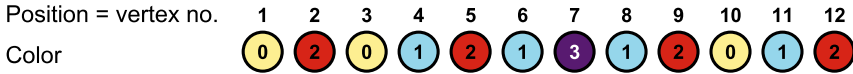
Producing:

$$\begin{array}{l}
 A' = 8 \ 7 \ 6 \ | \ 7 \ 3 \ 8 \ | \ 5 \ 3 \\
 B' = 2 \ 5 \ 1 \ | \ 4 \ 1 \ 2 \ | \ 4 \ 6
 \end{array}$$

a) 12 node graph.



b) Direct representation of coloring. A one dimensional array stores the colors.



c) Order based representation of coloring. A one dimensional array stores a permutation of vertices, and a greedy algorithm allocates the colors.

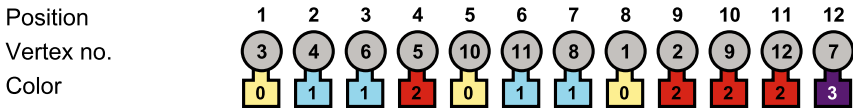


Fig. 2. A direct and order based representation of a coloring for a 12 node graph

Point mutation also produces duplications and deletions. On the other hand, Holland’s inversion operator respects the permutation property, and can indeed prove useful as a reordering operator.

The best known crossovers designed for permutations are probably partially matched crossover (PMX) [20], order crossover (OX) [10] and cycle crossover (CX) [35]. OX and CX are explained below along with another more recent example – merging crossover (MOX) [1]; a description of PMX is omitted because it is less relevant to the present study than the other examples. We will discuss the shortcomings of applying the simple order based approach to set partitioning problems in Sect. 6.

Order Crossover (OX)

This operation always produces legal permutations. It starts by selecting two crossing sites at random:

$$\begin{array}{l}
 A = 8 \ 7 \ 6 \ | \ 4 \ 1 \ 2 \ | \ 5 \ 3 \\
 B = 2 \ 5 \ 1 \ | \ 7 \ 3 \ 8 \ | \ 4 \ 6
 \end{array}$$

Values from the middle segment of one parent are then deleted from the other to leave holes. For example values 4, 1 and 2 will leave holes, marked by 'H' in string B:

$$B' = H \ 5 \ H \ | \ 7 \ 3 \ 8 \ | \ H \ 6$$

In one version of OX, the holes are filled with a sliding motion that starts from the beginning of the string.

$$B' = 5 \ 7 \ 3 \ | \ H \ H \ H \ | \ 8 \ 6$$

The substring from string A is then inserted into string B. The final result of this cross and the complementary cross is:

$$\begin{array}{l}
 A' = 6 \ 4 \ 1 \ | \ 7 \ 3 \ 8 \ | \ 2 \ 5 \\
 B' = 5 \ 7 \ 3 \ | \ 4 \ 1 \ 2 \ | \ 8 \ 6
 \end{array}$$

Cycle Crossover (CX)

The cycle crossover operator ensures that each position in the resulting offspring is populated with a value occupying the same position in one or other of the parents. As an example, suppose we have strings A and B below as our two parents:

$$\begin{array}{l}
 A = 8 \ 7 \ 6 \ 4 \ 1 \ 2 \ 5 \ 3 \ 9 \ 10 \\
 B = 2 \ 5 \ 1 \ 7 \ 3 \ 8 \ 4 \ 6 \ 10 \ 9
 \end{array}$$

We now start from the left and randomly select an item from string A. Suppose we choose item 6 from position 3, this is then copied to position 3 of the offspring we shall call A':

$$A' = - \ - \ 6 \ - \ - \ - \ - \ - \ - \ -$$

In order to ensure that each value in the offspring occupies the same position as it does in either one or other parent, we now look in position 3 of string *B* and copy item 1 from string *A* to the offspring:

$$A' = - \quad - \quad 6 \quad - \quad 1 \quad - \quad - \quad - \quad - \quad -$$

Next we look in position 5 of string *B* and copy item 3 from string *A*:

$$A' = - \quad - \quad 6 \quad - \quad 1 \quad - \quad - \quad 3 \quad - \quad -$$

Examining position 8 in string *B* we find item 6. This completes the cycle. We now fill the remaining positions in *A'* from string *B* thus:

$$\begin{aligned} A' &= 2 \quad 5 \quad 6 \quad 7 \quad 1 \quad 8 \quad 4 \quad 3 \quad 10 \quad 9 \\ B' &= 8 \quad 7 \quad 1 \quad 4 \quad 3 \quad 2 \quad 5 \quad 6 \quad 9 \quad 10 \end{aligned}$$

The offspring *B'* is obtained by performing the complementary operations.

Merging Crossover (MOX)

Merging crossover (MOX) was presented by [1] for use on graph coloring problems. Initially two *n* element parents are randomly merged into a single 2*n* element list. The first occurrence of each value in the merged list gives the ordering of elements in the first child, and the second occurrence in the second child. MOX is illustrated in Fig. 3. [1] point out the following property of MOX: if an element, *a* precedes another element *b* in both parents, then it follows that *a* will precede *b* in both children.

Mutation Operators for Permutations

As happens with crossover, standard mutation will produce duplications and deletions in a chromosome, if an order based representation is used. Fortunately, a number of alternatives have been devised. The simplest of these was named ‘position based mutation’ [11]. This operation, also known as ‘insertion mutation’, simply involves selecting two values at random from a permutation list, and placing the second before the first. Another straightforward mutation suitable for order based representations is ‘order based mutation’ [11] or ‘swap mutation’, which selects two values at random and swaps their positions. Davis also promotes an idea he calls ‘scramble sublist’ [11], in which a substring is selected stochastically and its contents randomly shuffled. In the present work, however, Holland’s inversion operator is used extensively

Merging Crossover (MOX)

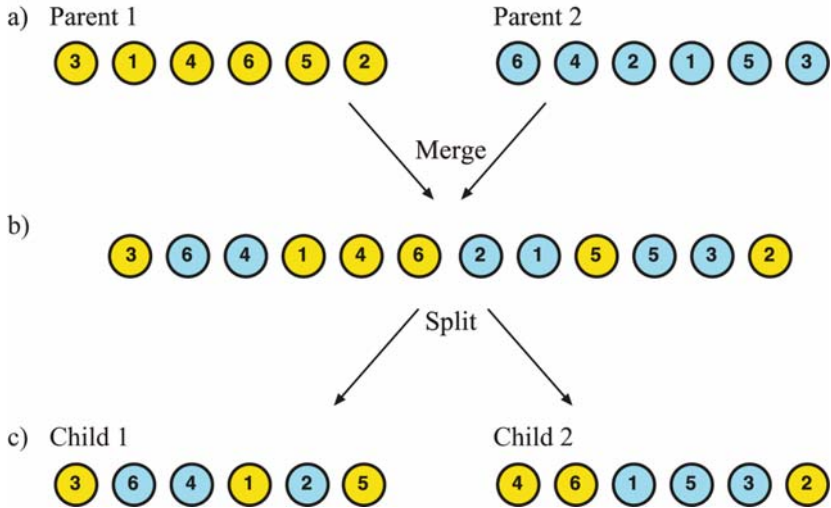


Fig. 3. MOX crossover [1], used as a basis for the new MIS crossover

as a mutation. This operator seems to be particularly effective for certain set partitioning problems. As we shall see later when we look at the grouping and reordering heuristics of [9], the act of reversing a list (or sublist) can have a positive effect on the result, following the application of the greedy decoder.

5 A Simple Steady-State GA

For convenience we will use a simple steady-state GA as a framework for our present study. There are few parameters to set using this approach: no global fitness function is used, for example, thus roulette wheel selection is avoided, and so is the need to manipulate and scale the fitness values. Tuning the fitness function to get the selective pressure just right can be very difficult, and getting it wrong can prove a disaster. The present author favors simple pairwise comparisons to identify whether one individual is better than another, using the result to determine who shall live and who shall die. With this approach we are only concerned with *whether* one individual is better than another, and not with *how much*. The crossover rate in this simple GA is always applied at 100%, and mutation (when used) at one per individual. This one-size-fits-all approach will allow us to concentrate our efforts on representational issues, genetic operators and performance measures (fitness values), substantially reducing the tuning requirements. Other parameters are not quite so easy to standardize as selection, crossover, and mutation rates, however. For example,

the population size and stopping criterion are best adjusted to suit the type and size of problem. The simple steady-state GA is outlined in Algorithm 2.

Algorithm 2 A Simple GA

```

Generate  $N$  random strings { $N$  is the population size}
Evaluate the performance measure for each string and store it
Apply local search to the offspring {optional}
repeat
  for all strings in the population do
    Each string, in turn, becomes the first parent
    Select a second parent at random
    Apply crossover to produce a single offspring
    Apply mutation {optional}
    Apply local search to the offspring {optional}
    Evaluate the performance measure for the offspring
    if the offspring passes its performance test then
      Then it replaces its weaker parent in the population
    else
      the offspring dies
  until stopping condition satisfied
  
```

At the start of the procedure a population of N random strings is generated. Once the initial population is created, the individual members are evaluated, according to some performance measure (or fitness value). Within the main generation loop, each member of the population is selected in turn and paired in crossover with a second individual, selected (uniformly) at random. The performance measure of the resulting single offspring is then compared to that of its weaker parent. In the simplest version of this algorithm, the new offspring replaces its weaker parent if it is better, otherwise it dies. Later on in the Chapter, when the simulated annealing cooling schedule is introduced, the conditions upon which a new offspring is accepted will be relaxed, in an attempt to maintain diversity within the population. A simple stopping condition is applied throughout the present work whereby the GA runs for a fixed number of generations specified in advance, a generation being defined as N trials of crossover, one led by each member of the population in turn.

6 Set Partitioning Problems

Set partitioning problems (also known as grouping problems [15]) were first introduced in Sect. 1, and two examples – graph coloring and bin packing – have been referred to briefly as examples in Sect. 4. Recall that the generic

version involves partitioning n objects into m sets, without violating problem specific constraints, so that each object is assigned to exactly one set and the objective function is optimized. The precise nature of the constraints and objective function will vary depending on the variant concerned. In the following Sections we shall look at some examples of set partitioning problems. First of all we will cover the three problems addressed in the current Chapter: graph coloring, bin packing and examination timetabling. This Section will then conclude with a brief overview of some other important set partitioning problems.

6.1 The Graph Coloring Problem

As mentioned in Sect. 4, the graph coloring problem (GCP) involves finding a minimum set of colors for the vertices of a given graph, so that no two adjacent vertices have the same color. The optimum set of colors for a particular graph coloring instance is often referred to in the literature as its *chromatic number*. A legal coloring for a graph with 12 nodes is illustrated in Fig. 4. Thus, we aim to partition a set of vertices into the minimum number of color classes, so that each vertex belongs to exactly one color class. The restriction that imposes different colors on adjacent vertices is an example of a *hard constraint*, because no coloring where this condition is violated is allowed.

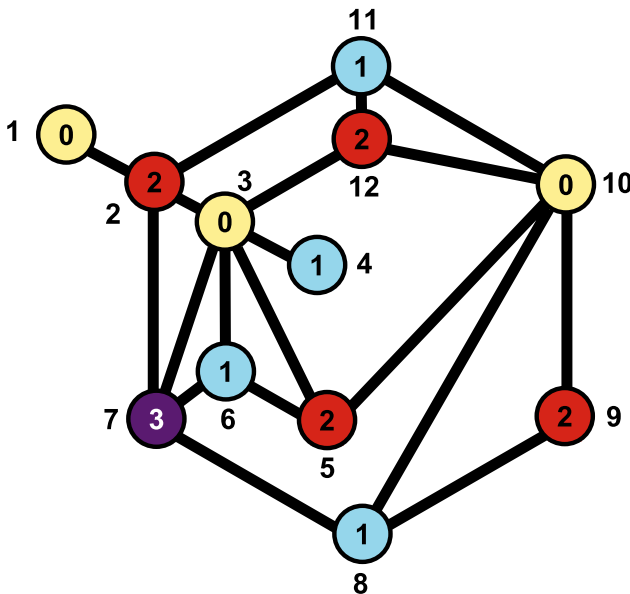


Fig. 4. A legal coloring for a graph with 12 nodes; color labels appear inside the vertices and vertex labels outside

In many ways the GCP is the archetypal set partitioning problem, because it has probably attracted more interest than any other problem of its type. Indeed, the field is highly competitive and in 1993 the problem was the subject of a Discrete Mathematics and Theoretical Computer Science (DIMACS) implementation challenge [24]. This involved pitting the best algorithms of the day against each other on a collection of large and specially devised difficult benchmark instances. The GCP provides a useful test bed for techniques applicable more widely to real world problems such as timetabling [4], frequency assignment [42], and many others.

6.2 The Bin Packing Problem

Bin packing problems are concerned with packing a given number of items, having different sizes, into a minimum number of equal-sized bins. In this Chapter we shall consider only the simplest of these problems, known as the one-dimensional bin packing problem [30]. In this version of the problem we are concerned only with weights of the items, and not with their areas, volumes or shapes. The goal is to partition a set of items, of differing weights, into a minimum number of bins with equal weight capacity. Like graph coloring, the bin packing problem imposes a hard constraint: the total weight of items occupying any bin must not exceed its weight capacity.

6.3 The Examination Timetabling Problem

The examination timetabling problem involves scheduling a set of examinations into a number of time slots in such a way that the schedule obeys any given constraints and also gives due consideration to any other issues conducive to producing a ‘good’ timetable. Many different variants exist for this important real-world problem, and the choice of practical solution method will depend on the types of constraints involved and also on the objectives that need to be optimized (see [4] and [40] for more details). In its most basic version, the examination timetabling problem is identical to the graph coloring problem, with the colors representing time slots and vertices representing examinations. In this model, an undirected edge between vertices indicates that at least one student is taking both exams. The goal of this basic version is to schedule all the examinations in the minimum number of time slots, so that there are no clashes – that is, no student is scheduled to take more than one exam in any time slot.

In practice available resources are finite and additional hard constraints are usually imposed, over and above the need to schedule examinations with no clashes. A university has an upper limit on the number of candidates it can seat in a time slot, for instance. It is instructive to note that the seating capacity limitation is identical to the bin packing constraint: items of various sizes being replaced with examinations with various numbers of candidates.

Thus, a feasible solution to the timetabling problem that seats all students and avoids all clashes requires the simultaneous solution to the underlying graph coloring and bin packing problems. Other common constraints include:

- Candidates taking a particular exam must not be split across rooms.
- Exam A must be scheduled before exam B.
- Exam A must be scheduled at the same time as exam B (because they contain similar material).
- Exam A must take place in a particular room (because special resources are needed).

In addition to the various hard constraints imposed by different institutions, universities have different views as to what constitutes a ‘good’ timetable, as opposed to simply a feasible one. Most commonly these desirable but not essential properties (sometimes called ‘soft constraints’) include some measure of a ‘fair spread’ of examinations for the students taking them. For example, scheduling students to take two examinations in consecutive time slots is usually avoided if possible. Indeed, some institutions will go much further than this to ensure that as many students as possible have good revision gaps between their examinations. Other desirable properties may consider efficiency or convenience related to the staff involved in marking the papers. For example, examinations with large numbers of candidates may be scheduled early to give more time for marking the scripts. The present study is confined to two hard constraints:

1. avoiding clashes, and
2. keeping within the total seating capacity.

Thus, the version of the examination timetabling problem addressed here is a simple combination of graph coloring and bin packing, our other two test problems.

6.4 Other Set Partitioning Problems

There are many set partitioning problems with great practical application. Here are a few examples:

- Equal piles and assembly line balancing.
- Frequency assignment problem.
- Vehicle scheduling problem.

The equal piles problem was first studied by [25] and involves partitioning N numbers into K subsets, such that the sums of the subsets are as near equal as possible. When applied to assembly lines [38], the equal piles problem seeks to assign assembly tasks to a fixed number of workstations in such a way that the workload on each workstation is nearly equal.

The frequency assignment problem can be derived from the graph coloring problem, but the level of conflict (that is, interference) between the nodes is related to their distance apart rather than a simple adjacency conflict [23, 42]. Vehicle scheduling involves assigning customers to vehicles for the pickup and/or delivery of goods [36].

7 Motivation for the Present Study

Set partitioning problems are very challenging for genetic algorithms, and designing effective crossover operators is notoriously difficult. We have already explored some of these issues in Sect. 4. There are two main challenges:

1. solution infeasibility, and
2. representational redundancy.

Solution infeasibility occurs when candidate set partitions violate problem constraints. As an example, consider the twelve node graph coloring problem illustrated in Fig. 2(a) and a direct representation of a legal coloring as shown in Fig. 2(b). It is easy to imagine a simple one point (or multi-point) crossover producing an illegal coloring, with some adjacent nodes having the same color. Infeasible solutions to the bin packing problem are easily produced in a similar way, producing overfull bins. *Representational redundancy* can also be a serious problem in set partitioning problems in which the class labels are interchangeable. The arbitrary allocation of color labels for the GCP and bin labels for the BPP establish these two problems in this category. Representational redundancy can artificially inflate the size of the search space and also reduce the effectiveness of crossover operators. Given these difficulties, it is probably not surprising that, despite the predominance of population-based methods, crossover plays a very minor role in many state-of-the-art approaches to solving set partitioning problems. Standard crossover operators are just not very effective at propagating meaningful properties about set membership from parents to offspring.

Of special note, however, are two evolutionary techniques that have recently appeared in the literature. These, unlike their predecessors, include crossover operators that appear to contribute significantly to the overall success of the algorithms. Furthermore, both of these techniques have produced world-beating results for hard literature benchmarks in their respective fields of application. The first of these, known as the grouping genetic algorithm (GGA) was developed by [16] for the bin packing problem. The second algorithm, the hybrid coloring algorithm (HCA) [18] was written for the graph coloring problem.

A common feature shared by the two novel crossover operators used in each of these methods, is the focus on propagating complete partitions, or sets, from parents to offspring. Essentially, both algorithms rely on a direct encoding

scheme to assign set membership to items in the manner of Fig. 2(b). However with the GGA this direct representation is augmented with a grouping part, which encodes each group (that is, set) with a single gene, and it is to the grouping part only that the genetic operators are applied. In the GGA, the standard part of the chromosomes merely serve to identify which items belong to which group. On the other hand, the *greedy partition crossover (GPX)* used in HCA works on the direct encoding explicitly. The parents take it in turns to contribute a complete color class to the offspring. GPX will always select the largest remaining color class from the chosen parent (hence the term ‘greedy’ in GPX), and following its transfer to the offspring, all copied vertices will be removed from both of the parents to avoid any duplication of vertices in the offspring.

A disadvantage shared by both the above techniques is their need to repair the infeasible solutions that are inevitably produced by the genetic operators. In addition, successful implementations of these methods also make extensive use of local search to further improve the quality of the solutions. The GGA, for example, uses a powerful backtracking technique adapted from [30] to unpack items from some bins and attempt to repack them more favorably in others. The HCA algorithm of [18] relies on small populations of just five or ten individuals and typically applies several thousand iterations of their tabu search algorithm to each new offspring produced by the evolutionary algorithm (one could speculate on the relative contribution evolutionary part to the methods as a whole). Nevertheless, the researchers in each case present convincing evidence to support the inclusion of their evolutionary components.

Interestingly, the GGA has been adapted by various authors to several other set partitioning problems, including the graph coloring problem [13, 14], the equal piles problem [15], and the course timetabling problem [29]. For each application the choice of problem-specific repair and local search heuristics has probably had a major influence on its level of success. To the best of this author’s knowledge the HCA algorithm has not yet been adapted for other applications.

Having now introduced the reader to arguably the best known and most successful evolutionary approaches to set partitioning for which crossover plays a significant role, I will now move on to outline the motivation for the present work. It is clear that a major weakness is shared by all evolutionary techniques that rely on direct encoding for set partitioning problems – this being their need to repair infeasible solutions. In addition, we have also noted an extensive use of problem-specific local search heuristics in the algorithms we have reviewed above. These are not only time consuming, but they also call into question the relative contribution of the evolutionary algorithm. Furthermore, the repair and local search heuristics used by these methods are not very portable from one set partitioning problem to another, and success seems to be quite variable, depending heavily on the quality of supporting heuristics.

The main aim of the present Chapter is to present a new, and reasonably generic, order based framework suitable for application, with minimum adaptation, to a wide range of set partitioning problems. The clear advantage of using an order based approach is that every permutation is decoded as a feasible solution, meaning no costly repair mechanisms are required following a crossover event, however heavily constrained the problem. On the other hand, it is conceivable that techniques that employ direct encoding will find it increasingly difficult to repair the result of a crossover, the more heavily (or multiply) constrained a problem becomes. Historically the downside of order based genetic algorithms is that, to the best of this author's knowledge, nobody has so far been able to come up with a really effective crossover capable of transmitting useful features from parents to offspring for set partitioning problems. It is hoped that the new crossover operators presented in the present work do something to redress the balance and make order based approaches more competitive. Indeed, the new operators share a key property inspired by the GGA and GPX crossovers: they tend to propagate whole partitions or sets from parents to their offspring.

Perhaps the most innovative feature of the new order based approach is the inclusion of some simple grouping and reordering heuristics to preprocess the chromosomes prior to crossover. The idea is to encourage the transmission of whole set partitions, when a suitably designed crossover is used, in a way that is normally not possible with order based crossovers. We shall see in the next Section that the grouping and reordering heuristics of [9], used in the present study to preprocess the chromosomes prior to crossover, can be applied readily to a range of different set partitioning problems and, unlike many of the repair mechanisms used by direct encoding methods are not restricted to one type. Furthermore, no lengthy local search procedures are required and only a very few iterations of Culberson and Luo's heuristics are needed for preprocessing. Thus, although the exact choice of objective or fitness function will very likely depend on the specific set partitioning application, it is envisaged that the complicated problem-specific heuristics and costly backtracking, typical of many other approaches, can largely be avoided. The next Section introduces the heuristics of [9], and explains their power and versatility. It is the opinion of the present author that these very elegant techniques have been rather neglected by researchers.

8 Culberson and Luo's Grouping and Reordering Heuristics

Culberson and Luo's (CL) heuristics were originally devised to solve the graph coloring problem and belong to a family of methods that use simple rules to produce orderings of vertices (or items). Once created, the orderings are presented to a greedy decoder for transformation into legal colorings (or set

partitions). Successful ordering heuristics are distinguished by the production of high quality solutions. The simplest and fastest ordering heuristics, unlike the CL heuristics, generate a solution in one go. For the graph coloring problem the best known one-shot techniques determine the orderings by placing the most heavily constrained vertices (namely, those with many edges connecting them to other vertices) before those that are less constrained. While most of these techniques can be described as static, because the orderings remain unchanged during the greedy color assignment process [31, 43], a somewhat more sophisticated technique, known as *DSatur* [2] operates dynamically. Starting with a list ordered in non-ascending sequence of vertex degree, *DSatur* assigns the first vertex to the smallest color label, then it reorders the remainder of the list to favor vertices adjacent to the newly assigned vertex. The algorithm continues in this way, sequentially assigning the lowest available color label to the next vertex on the list, then reordering the remainder, until every vertex has received a color. Thus, *DSatur* assigns colors to unassigned vertices, giving priority to the vertices with the most neighbors already colored, using vertex degree to break the ties.

One-shot ordering heuristics have also been developed for other set partitioning problems, and they operate in a similar fashion to the graph coloring heuristics discussed above. Ordering heuristics for the frequency assignment problem (the assignment of radio frequencies to reduce interference between transmitters), for example, are almost identical to those used for graph coloring (see [23] for a survey). This is probably not surprising, given that frequency assignment is a derivative of the graph coloring problem. Simple versions of the examination timetabling problem also make use of graph coloring heuristics [4]. A popular method for ordering items for the bin packing problem is to place them in non-ascending sequence of their weights. A simple greedy algorithm can then be used to assign the items to the first available bin, bins being identified by consecutive integer labels. This scheme for bin packing is known as the first fit decreasing weight (FFD) algorithm [7].

Despite their attractiveness in terms of speed and simplicity, however, one-shot ordering heuristics do not always perform very well in practice, although there are exceptions. FFD can solve many large benchmark bin packing instances to optimality, for example, and *DSatur* works well on certain graph coloring instances. In other cases though, such algorithms are only useful to supply upper bounds or provide a starting point for a more sophisticated method.

In a different category are the ordering heuristics of Culberson and Luo (CL) [9]. These methods group and rearrange whole color classes (or sets of items), rather than sequencing the vertices individually. Unlike the one-shot methods discussed above, the CL heuristics can be applied repeatedly, leading to a gradual improvement to a solution. Of particular significance is a rare property of the CL heuristics which ensures that it is impossible to get a *worse* coloring by applying any of their reordering techniques to the GCP, and it

is possible that a *better* coloring (using fewer colors) may result (see [9] for details). They apply a random mix of various reordering heuristics and call the composite algorithm ‘iterated greedy’ (IG).

Two main stages of IG can be identified:

1. grouping, and
2. reordering.

Figure 5 illustrates some key operations from IG applied to a small graph with 12 vertices and 14 edges. Figure 5(b) gives a typical random permutation of the vertices from Fig. 5(a) and also the resulting greedy coloring. Figure 5(c) shows the grouping operation used to sort the list in non-descending sequence of color label, and 5(d) gives the arrangement following the application of one of the CL reordering heuristics called ‘largest first’. The largest first heuristic rearranges the color classes in non-ascending sequence of their size. Note that the positions of color classes 1 and 2 have been reversed in Fig. 5(d). This follows advice in [9] to interchange positions of equal sized color classes. In Fig. 5(f) vertices are randomly ‘shuffled’ within (but not between) the color classes. (Note: shuffle, although mentioned, does not appear to have been extensively used by [9] in the IG algorithm. However it is included here because of its value in the present study). Finally, the greedy algorithm is applied to the new arrangement – Fig. 5(f) – and the result is shown in Fig. 5(g). Note that vertices 4 and 1 are reassigned lower color labels, leading to a reduction in the size of color class 2. Thus, given an initial permutation of vertices, the IG algorithm can be defined by the following repeating sequence:

1. greedy assignment,
2. grouping of color classes,
3. reordering of complete color classes,
4. shuffle within each color class (optional).

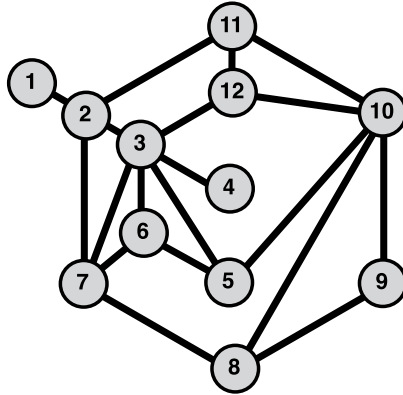
Various properties of the color classes were assessed by [9] as criteria for reordering:

1. *Reverse*: reverse the order of the color classes
2. *Random*: place the groups in random order
3. *Largest first*: place the groups in order of decreasing size (Fig. 5(d))
4. *Smallest first*: place the groups in order of increasing size
5. *Increasing total degree*: place the groups in increasing order by the total degree of the group
6. *Decreasing total degree*: place the groups in decreasing order by the total degree of the group

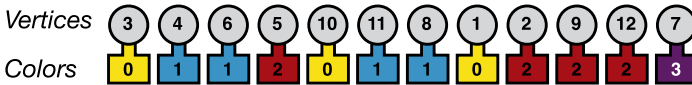
The favored combination turned out to be largest first, reverse and random, used in the ratio 50:50:30. Although [9] applied their IG algorithm to the GCP, many of the reordering heuristics are equally applicable to other set partitioning problems. Reverse, random, largest first and smallest first, for instance,

Local Search Operations

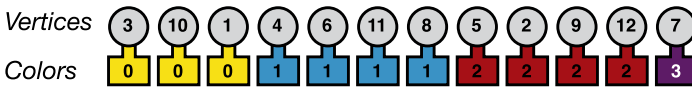
a) Graph with 12 vertices



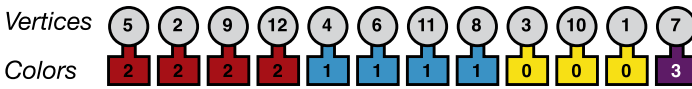
b) Random permutation of vertices with greedy coloring



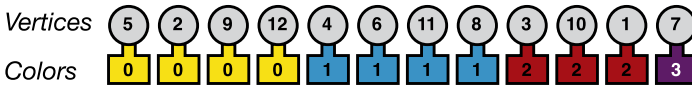
c) Independent sets sorted



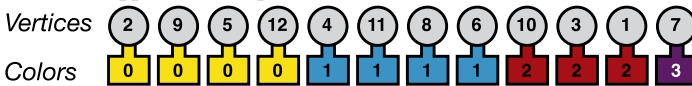
d) Largest set first reordering applied



e) Colors relabelled



f) Shuffle applied to independent sets



g) Greedy assignment applied to new ordering

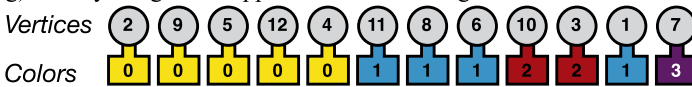


Fig. 5. Various operations by [9] used in the local search procedure

can be used for the bin packing problem. Note though that heuristics based on vertex/item degrees have no meaning in this context, so the increasing and decreasing total degree heuristics are not appropriate for bin packing. Nevertheless, an alternative measure can be used: the total weight of items in each bin. In this way the two CL heuristics that sequence on the total degrees for each group can be replaced with heuristics that do their ordering on the basis of the total weights of items in each bin. CL heuristics can also be applied to a simple version of the examination timetabling problem where the only considerations are to avoid clashes and/or seating capacity violations. As mentioned in Sect. 6.3, clash avoidance and seating capacity compliance simply represent underlying graph coloring and bin packing problems, respectively.

The crucial feature that determines the applicability of the CL reordering heuristics to a particular set partitioning problem is whether a solution is changed if the groups are simply re-labelled. For convenience, groups are usually identified by integer labels, to represent their color class, bin ID or time slot, and so forth. With graph coloring and bin packing it does not matter which integer label is assigned to which group – it will not change the total number of colors or bins required. On the other hand, if revision gaps are required in examination timetables, the sequence of integer labels will be relevant, and will correspond to a sequence of time slots; by contrast, time slots can be shuffled into any sequence to avoid clashes and comply with seating arrangements. Interestingly, the reverse heuristic maintains the relative positions of time slots. The frequency assignment problem has similar limitations, and the applicability of reverse has been proven in [42].

We will now look at some new crossover variations that attempt to preserve color classes, when used in conjunction with the grouping and sorting heuristics described above.

9 Modifications to a Standard Order Based GA for Set Partitioning

Genetic algorithms require crossover techniques that preserve *building blocks* [19] appropriate to the problem at hand. A building block can be viewed as a group of elements on a chromosome that ‘work together’ to produce or influence some identifiable feature in the solution. For example, the ‘sets’ in set partitioning problems come into this category, and thus it makes sense to use a crossover that preserves them. In the present work CL grouping and reordering heuristics are used to preprocess the order based chromosomes to make it easier to preserve the set groupings. Two new crossover operators, POP and MIS (first described in [34]), seem to be particularly effective in maintaining this group integrity when used in conjunction with CL heuristics. Indeed, results for these operators are impressive when compared to those obtained using other order based crossovers for the graph coloring problem [34]. Further evidence in support of these operators is presented later in the Chapter. Note that

no special modifications were necessary for order based mutations, and that inversion and insertion mutations proved the most useful in the present study.

The new crossover operators are compared with three selected order based operators of historical importance: cycle crossover (CX) [35], uniform order based crossover (UOBX) [11], and merging crossover (MOX) [1]. CX, OX and MOX have already been described in Sect. 4. UOBX was developed from OX by [11] with the GCP in mind and is good at preserving relative positions and orderings. CX is good at preserving absolute positions of vertices, and every vertex in the offspring list will occur in exactly the same position in one or other of its parents. CX has proven effective in the related frequency assignment problem [42]. As mentioned previously, MOX is good at preserving relative positions. I will now outline the two new crossover operators, POP and MIS.

Permutation Order Based Crossover (POP)

Permutation order based (POP) crossover uses ideas from the well known order crossover (OX) [35], described earlier, but at the same time it tries to emulate the basic one point crossover of the ‘standard’ bit string GA, which simply selects two parents and a cut point. The first portion of parent 1 up to the cut point becomes the first portion of offspring 2. However, the remainder of offspring 2 is obtained by copying the elements absent from the first portion of the offspring in the same sequence as they occur in parent 2 (see Fig. 6).

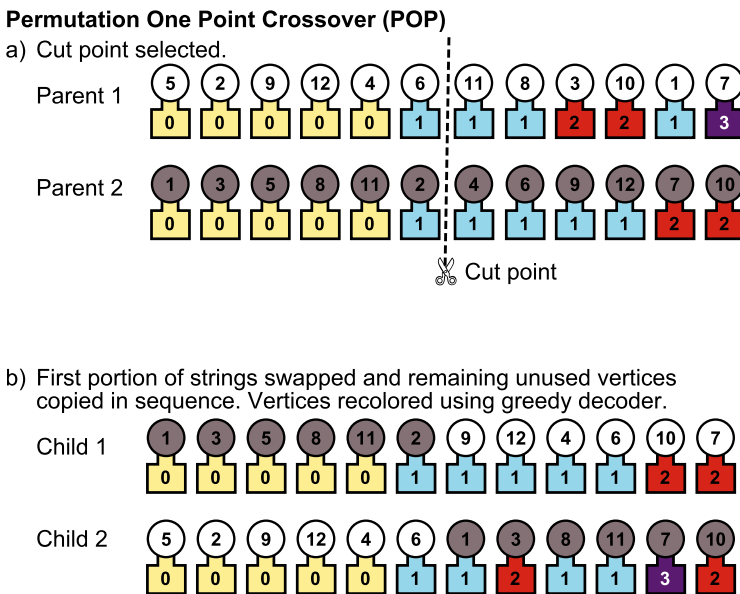


Fig. 6. POP crossover

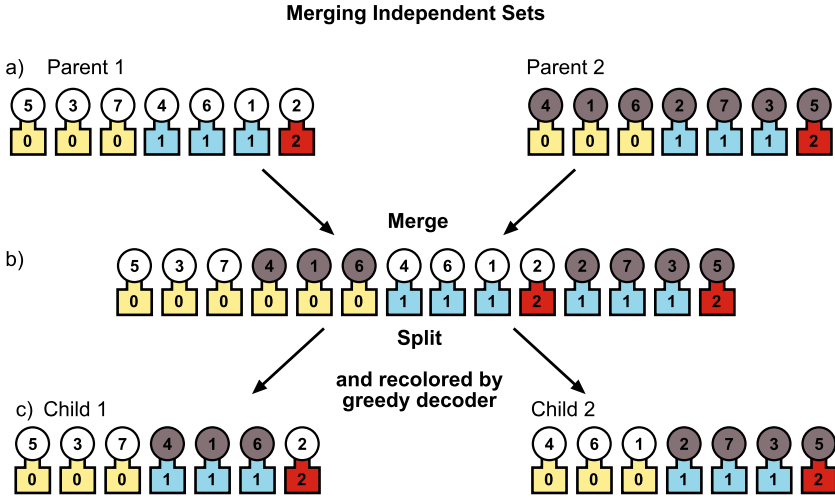


Fig. 7. Merge independent sets crossover, MIS

The same idea was used in [8], although the crossover was not given a specific name. However, the present implementation relies on the CL heuristics for preprocessing the chromosomes, without which it did not work very well, as demonstrated in [34]. We will identify two variants of POP: POP1 and POP2. These differ slightly in the way the cut point is selected: for POP1 it is chosen at random and can appear anywhere in the list, but for POP2 the cut point is restricted to a boundary between two set groupings. Of course application of POP2 is dependent on having previously sorted the color classes.

Merging Independent Sets Crossover (MIS)

Merging independent sets (MIS) is a new crossover, adapted from MOX. It requires that the color sets are first grouped together in both of the parents, as illustrated in Fig. 7(a). MIS then proceeds in the same way as MOX, but whole color sets are copied from the parents to the merged list in one go (Fig. 7(b)), rather than individual vertices. The merged list is split in exactly the same way as for MOX, with the first occurrence of each vertex appearing in the first offspring and the second occurrence in the second offspring, reapplying the greedy decoder to the new offspring (Fig. 7(c)). The idea of MIS is to better preserve the parents' color classes than MOX.

9.1 Performance Measures/Fitness Values

The objective function (namely, the value we are trying to optimize) is not always the best measure of progress for an optimization algorithm to use. For example, a common objective function for set partitioning problems is to count the number of classes – for instance, the number of colors and bins

respectively – for graph coloring and bin packing. Unfortunately, as previously mentioned, representational redundancy can mean that for a given number of classes, a very large number of different assignments (of vertices to colors, or items to bins) is possible. For this reason we will use progress measures that attempt to distinguish between ‘good assignments’ and ‘bad assignments’, for a given class count. We will start by looking at a performance measure that is generally applicable to most set partitioning problems, and then we will examine some more specific measures that can be used for particular problems.

General Set Partitioning

The performance measure in Eqn. (2) was introduced by [9] for the GCP. However, it is equally applicable to other set partitioning problems.

$$P_1 = \sum_1^n c_i + nc \tag{2}$$

Interpreting this formula for the GCP, Eqn. (2) shows the *coloring sum* (that is, $\sum_1^n c_i$, where c_i is the color assigned to vertex i) added to the term nc , where n is the number of vertices and c the number of colors. For bin packing, $\sum_1^n c_i$ represents the ‘bin sum’, with the bins numbered consecutively, $\{1, 2, 3, \dots, c\}$, and item i assigned to bin number c_i . The other term in Eqn. (2), nc , is simply the number of items multiplied by the number of bins. A disadvantage of this performance measure is that it is sensitive to color (or bin) class labelling, and color classes or bin assignments need be sequenced so that the smallest integer label is assigned to the largest class, and the second smallest integer to the second largest class, and so on, for the measure P_1 to work effectively. This performance measure was used in an earlier study by the present author [34].

Graph Coloring

The following equation was devised by [14].

$$P_2 = \frac{1}{c} \sum_1^c D_j^2 \tag{3}$$

In Eqn. (3) $D_j = \sum_{i \in S_j} d_i$ represents the ‘total degree’ for group j with d_i denoting the vertex degree of the i_{th} node. Unlike P_1 , P_2 is insensitive to class labelling, and in Eqn. (3) color classes having a high total degree are favored.

Bin Packing

The final formula we will consider is due to [16].

$$P_3 = \frac{\sum_1^c (W_j/C)^2}{c} \tag{4}$$

Eqn. (4) has a similar structure to Eqn. (3), but the important class measurement is total weight of items in bin j , $\sum_{i \in S_j} w_i$, where w_i is the weight of item i in bin j . The bin capacity is denoted by C .

9.2 Comparing Order Based Crossovers

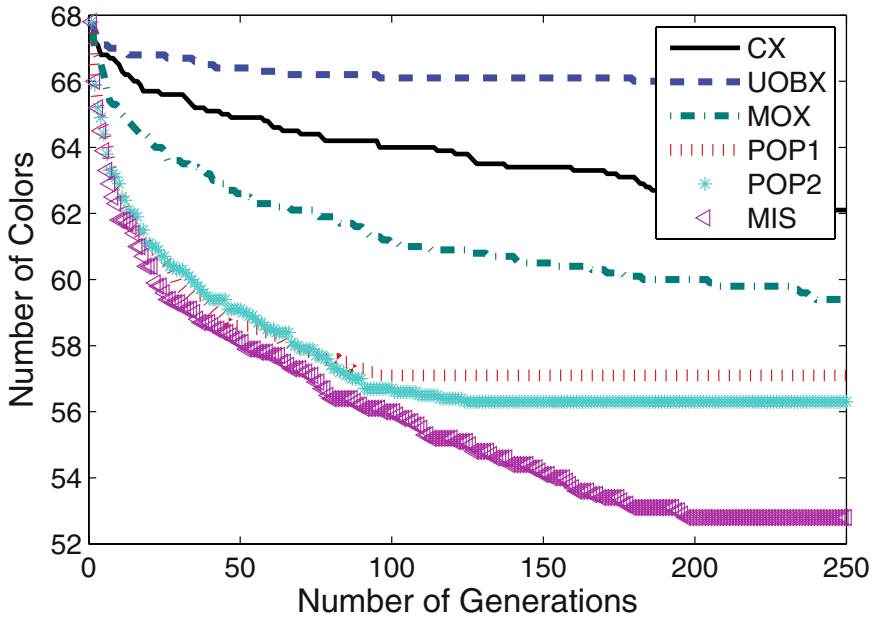
Experiments were conducted to assess the viability of the various crossover operators for the GCP and the BPP. The simple steady-state GA outlined in Algorithm 2 was used as a framework for this, omitting mutation. Two DIMACS (Discrete Mathematics and Theoretical Computer Science Implementation Challenge) benchmarks – DSJC500.5 and 1e450.25c – were used to demonstrate the performance of the crossovers on the GCP, and N4C3w2.A and N4C3W4.A from Scholl and Klein were used for the BPP (see Appendix-A for the data sets). The fitness function devised by Erben (Eqn. (3)) was used for the GCP and Falkenauer’s fitness function was chosen for the BPP. A population size of 300 was used for each experiment, and the GA run for 250 generations. A single iteration of local search steps, similar to those illustrated in Fig. 5, immediately followed each application of crossover. Recall that local search is based on the CL heuristics and consists of a ‘grouping’ and a reordering phase. However, the ‘largest first’ heuristic was changed to reflect features of the different fitness functions that were used for the GCP and the BPP. In the case of the GCP, the classes were sequenced according to the sum of vertex degrees in each color class (that is, decreasing total degree), rather than just counting the number of vertices belonging to each group. For the BPP the classes (which correspond to the contents of the various bins) were sequenced in non-ascending order of bin weights.

Results for all the experiments are displayed graphically as ‘best-so-far’ curves averaged over 10 replicate runs (see Figs. 8 and 9). Clearly the best results are obtained with MIS on the GCP and POP1 on the BPP. Analysis of variance tests show highly significant differences in the performance of the various crossover operators at the 0.01% level.

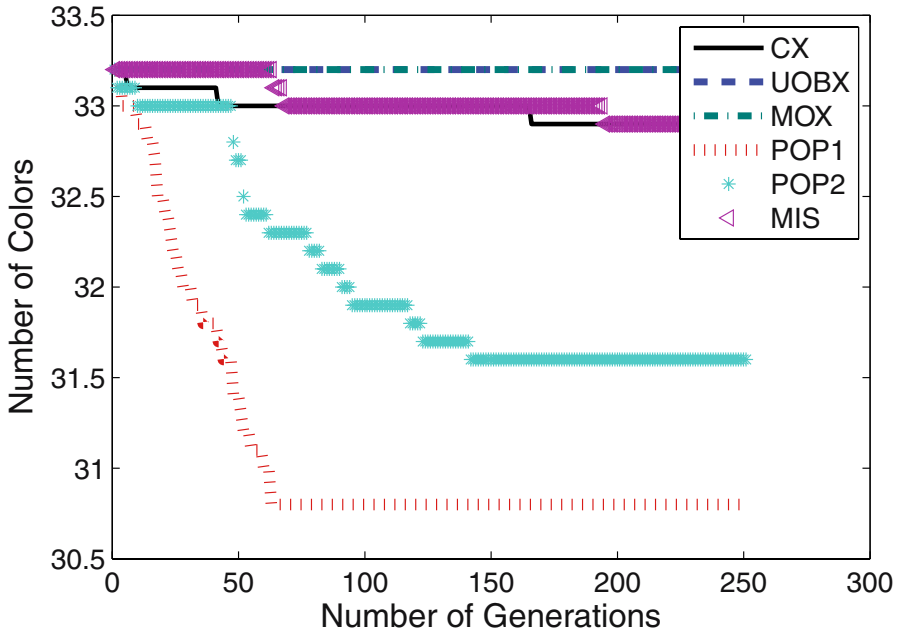
The reader is referred to [34] for a more rigorous set of comparisons. Results presented in this earlier paper also demonstrate the important contribution of the grouping (or sorting) and reordering heuristics.

9.3 The Genetic Simulated Annealing (GSA) Algorithm

Having assessed the relative performance of the various crossover operators, the next stage is to apply a suitably adapted GA to literature benchmark instances for various set partitioning problems, to see how the approach will compete with other published algorithms. In order to obtain really good solutions, it is necessary to balance population diversity with GA convergence, and consider larger populations and/or longer runs than were needed for comparing the crossovers. Clearly, the addition of a mutation operator will

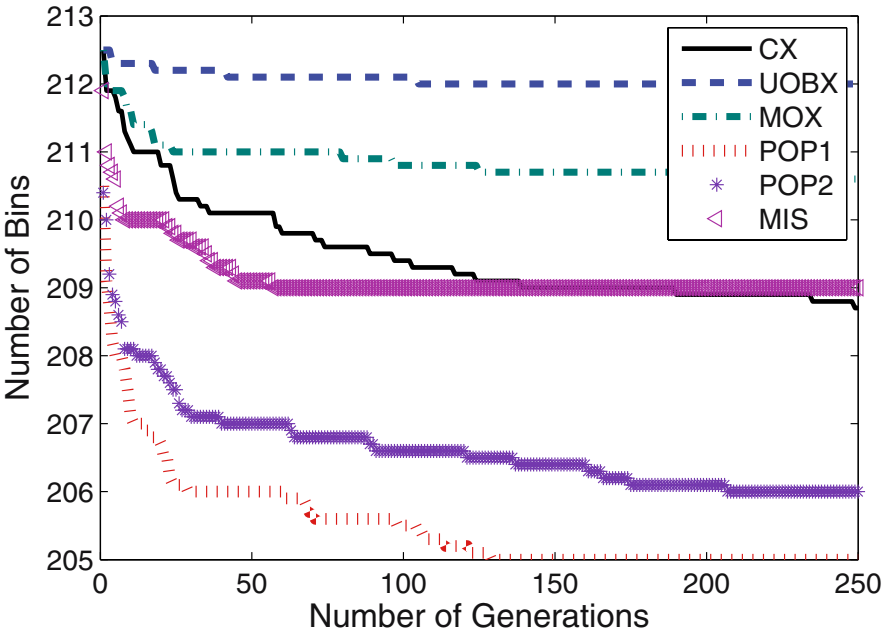


(a) DSJC500.5

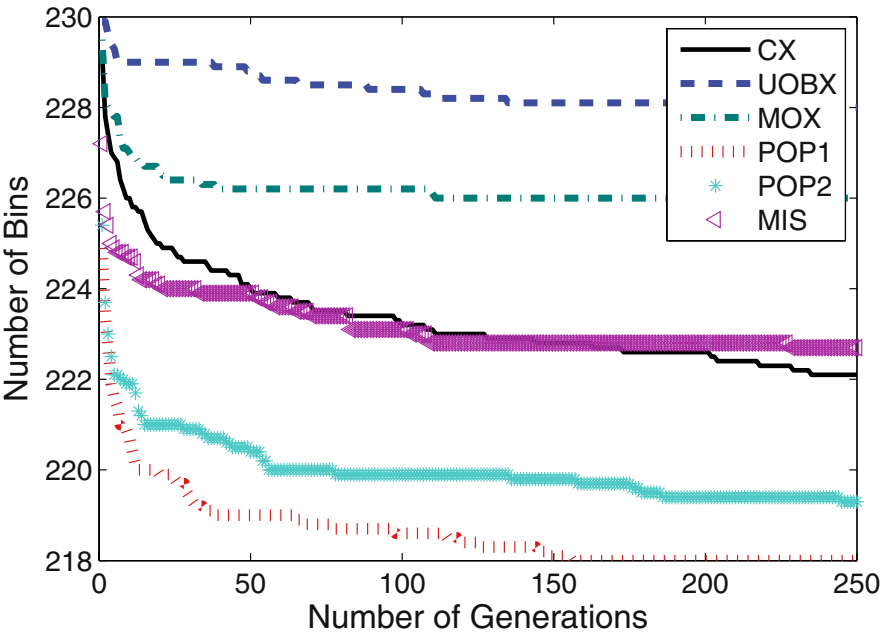


(b) le450_25

Fig. 8. Comparing order based crossovers with sorting of independent sets and largest total degree first



(a) N4C3W2_A



(b) [N4C3W4_A]

Fig. 9. Comparing order based crossovers with sorting of independent sets and largest total weight first

probably help maintain diversity within the population, giving a better opportunity to explore the search space and helping to avoid premature convergence. Mutation was deliberately left out of the previous experiments when we were assessing the crossovers.

Several of the mutation operators discussed in Sect. 4 were tried in some pilot studies: insertion and swap mutation, as well as scramble sublist. Of these, insertion produced the best results with the successful crossovers, producing good solutions quickly. Unfortunately these results were not quite world class, and despite the useful contribution of the mutation operator, the population tended to lose its variability towards the end of the run. In an attempt to improve matters, different ways of injecting extra variability into the population were explored.

Periodic restarts was the first of these ideas to be tried. This involved temporarily halting the GA, once it stagnated. Various ‘super-mutation’ operations were then selected stochastically and applied to the individual permutations in the population, in an attempt to inject some new variability. Operations that were tried include Davis’ ‘scramble sublist’ and Holland’s inversion (see Sect. 4) as well as a simple ‘delete-and-append’ operation, which deletes part of a string and then appends this deleted section to the end of the string. Once this super-mutation stage had been completed, the GA was restarted. Although this approach proved to be very successful on some instances, it required unacceptably long run times on others. A more efficient approach was subsequently found which involves making a small change to the replacement criterion in Algorithm 2: instead of simply replacing the weaker parent by its offspring when offspring is better than its parent, a simulated annealing cooling schedule was introduced, which allows a parent to be replaced occasionally by a poorer offspring. The main features of a simulated annealing cooling schedule are outlined below.

What is Simulated Annealing?

In physics the term ‘annealing’ refers to the very slow cooling of a gas into a crystalline solid of minimum energy configuration. Simulated annealing algorithms (SAs) [27] attempt to emulate this physical phenomenon. The process usually begins with the generation of a random solution, and this will act as the initial focus of the search. The SA will then make a very small change to a copy of this solution, generating a neighborhood solution, in an attempt to produce an improvement. If a better solution is found, then the improved solution will replace the original as the focus. However, it is well known amongst researchers that a simple hill climbing search such as this (accepting only ‘forward’ moves and never ‘backward’ ones) can easily become trapped in a local optimum. The main strength of an SA algorithm is its potential to escape from such traps. This is achieved by the occasional acceptance of a neighborhood solution that is somewhat worse than the current focus. The rate at which this

is allowed to occur is carefully controlled using an ‘acceptance probability’, and this is used to determine whether or not a newly generated neighborhood solution will replace the current focus solution. In general, the poorer the new solution, the less likely it is to be accepted. However, the analogy with the physical situation requires that inferior solutions should be less likely to be accepted as the search progresses. Initially the algorithm will probably accept almost anything, but towards the end of the search, the algorithm will behave more like hill climbing, accepting inferior solutions only on very rare occasions. Values for the acceptance probability – *prob* – for a minimization problem, are evaluated using Eqns. (5) and (6). Δ represents the difference between the objective functions (or costs) of the new solution, $C(S')$, and the focus solutions, $C(S)$. Note that the value of *prob* depends on the value of Δ and also on T , the current temperature, which is determined by the cooling schedule.

$$\Delta = C(S') - C(S) \tag{5}$$

$$prob = \min(1, e^{-\Delta/T}) \tag{6}$$

The new solution is accepted with probability 1 if $\Delta \leq 0$ (in other words, if the neighborhood solution is better than S) and with probability $e^{-\Delta/T}$ if $\Delta > 0$ (that is, if the neighborhood solution is worse than S). Throughout the execution of an SA algorithm, the temperature T is progressively lowered.

The Genetic Simulated Annealing (GSA) Implementation

For the present GSA implementation for set partitioning problems, the precise annealing schedule is determined from user-specified values for the number of cooling steps and the initial and final solution acceptance probabilities. We use n cooling steps to correspond to the number of generations, so that the temperature is decreased between each generation. Thus, knowing n and the initial and final acceptance probabilities, P_0 and P_n , as well as an additional parameter M that signifies an initial number of random trials, the starting temperature T_0 , the final temperature T_n , and the cooling factor α can be calculated, as indicated below.

$$\Delta_i = Perform(offspring) - Perform(weaker) \tag{7}$$

$$\Delta_{ave} = \frac{\sum_{i=1}^{i=M} |\Delta_i|}{M} \tag{8}$$

$$T_0 = -\frac{\Delta_{ave}}{\log P_0} \tag{9}$$

$$T_n = -\frac{\Delta_{ave}}{\log P_n} \tag{10}$$

$$\alpha = \exp \frac{\log T_n - \log T_0}{n} \tag{11}$$

Please note that Δ_{ave} (Eqn. (8)) is obtained by applying the genetic operators and also local search, if appropriate, to M randomly selected pairs of individuals from the initial population. In this way the performance measures of M offspring are compared with those of their weaker parents to obtain an estimate of Δ_{ave} . This estimate is then used to determine the starting temperature, the final temperature and the cooling schedule. The offspring generated during this initialization phase are discarded.

Algorithm 3 provides an outline of the GSA. Although the exact implementation details for the GSA, such as choice of crossover and mutation operators, and type of local search, vary according to the nature of the problem, the basic framework remains the same. Worthy of note is the simple adjustment made to the calculation of Δ_i , necessary because all the fitness functions (in other words, objective functions) used for the problems in the present study involve maximization, yet simulated annealing requires that the objective functions are minimized. We simply set $\Delta_i = Perform(weaker) - Perform(offspring)$ instead of $itPerform(offspring) - Perform(weaker)$.

10 Results on Literature Benchmarks

The versatility of the new techniques will now be demonstrated on some literature benchmarks for graph coloring, bin packing and timetabling. Except where otherwise stated, the following parameter settings were used for the

Algorithm 3 Genetic Simulated Annealing (GSA)

```

Generate  $N$  random strings  $\{N$  is the population size $\}$ 
Evaluate the performance measure for each string and store it
Apply local search to the offspring  $\{\text{optional}\}$ 
Initialize data,  $\{\text{obtaining } T_0, T_n \text{ and } \alpha\}$ 
 $S = S_0$ 
 $T = T_0$ 
for n generations do
  for all strings in the population do
    Each string, in turn, becomes the first parent
    Select a second parent at random
    Apply crossover to produce a single offspring
    Apply mutation to the offspring
    Apply local search to the offspring  $\{\text{optional}\}$ 
    Evaluate the performance measure for the offspring,
     $\Delta = Perform(weaker) - Perform(off)$ 
     $P_t = \min(1, \exp^{-\Delta/T})$ 
    if  $random(0, 1) \leq P_t$  then
      offspring replaces weaker parent
    else
      the offspring dies
   $T = \alpha \times T$ 

```

GSA: $M = 100$ (the number of preliminary trials to help establish the starting temperature), $P_0 = 0.999$ and $P_n = 0.0001$ (the starting and ending probabilities, respectively, of accepting an inferior offspring with an average magnitude of deviation in value from its weaker parent). Population sizes of 200 were used and ten replicate runs carried out on each benchmark instance, with average and best values quoted for the solutions in the results tables. Values of n , the number of generations, vary with different problem instances, as do precise details of the genetic operators and CL heuristics used.

10.1 Graph Coloring

The previously mentioned DIMACS benchmarks [24] provided most of the test instances for the present study, and are dealt with first. Further experiments are then reported on two special types of graphs, based around cliques. All test instances were chosen because they have been reported as ‘difficult’ by previous researchers.

The DIMACS Benchmarks

Seven benchmark instances were taken from the DIMACS challenge benchmark set, [24]. D250.5, D500.5 and D1000.5 are random graphs with edge density 0.5 and unknown chromatic number. Le450_15c and le450_25c are Leighton graphs with 450 vertices, and flat300_28 and flat1000_76 are flat graphs with 300 and 1000 vertices, respectively. The flat and Leighton graphs are structured with known chromatic numbers of 15, 25, 28 and 76, as indicated.

MIS crossover was selected because it worked well in the GSA for most of the instances. However, POP1 proved better for le450_25 so this crossover was used for this instance only. ‘Inversion’ was the selected mutation, which involved inverting the substring between two randomly selected cut points. Erben’s fitness function (Eqn. (3)) provided the performance measure, and three iterations of the local search, based on Fig. 5, seemed to be sufficient for the GCP. Increasing the number of iterations slowed the algorithm down considerably without improving the results. The local search was modified a little from Fig. 5 however (as was the case when comparing the crossovers), with the ‘decreasing total degree’ heuristic replacing the ‘largest first’. This was done to make the reordering criterion tie in better with Erben’s fitness function: both encourage the formation of classes with high values for total vertex degree. Results for the seven DIMACS benchmarks are presented in Table 1.

In Table 1 results for the GSA (columns 4 and 5) are compared with those obtained running a mutation only version (columns 6 and 7), which incorporates all the same parameters and features of the GSA but does not have the crossover. The iterated greedy algorithm was also tried, and the

Table 1. Results for genetic simulated annealing on graph coloring instances

Instance	Order based GSA				Mut GSA		It greed		DSat	Best known
	# Gens	Time	Mean	Min	Mean	Min	Mean	Min		
DSJC250.5	2000	314	29.1	29	31.6	31	30.0	30	37	<i>28</i>
DSJC500.5	3000	1895	49.9	49	56.4	56	53.8	53	65	<i>48</i>
DSJC1000.5	5000	11706	87.2	87	101.0	100	98.0	97	115	<i>83</i>
le450_15c	500	190	<i>15</i>	<i>15</i>	24.9	24	24.0	24	23	<i>15</i>
le450_25c	2000	854	29.3	29	29.9	29	29.0	29	29	<i>26</i>
flat300_28	1000	205	32.6	32	36.0	36	34.3	34	42	<i>31</i>
flat1000_76	5000	11711	86.3	85	100.0	99	97.4	96	114	<i>83</i>

results for this can be found in columns 8 and 9. The penultimate column contains the **Desatur** result for each instance and the best known results are listed, for comparison purposes, in the final column. The best known results were obtained by [18] using their hybrid evolutionary algorithm for graph coloring (HEA). Ten replicate runs were carried out for the GSA, mutation only GSA and also iterated greedy on each benchmark instance.

In more detail, column 2 gives the number of generations for the GSA (and also the mutation only version), and column 3 the average run time in seconds. The average and best results for the GSA and the mutation only version are presented in columns 4, 5, 6 and 7. The results for iterated greedy in columns 8 and 9 are produced by running this algorithm for the same length of time as the GSA, namely 314 seconds for DSJC250.5, 1895 seconds for DSJC500.5, and so forth. Bold font is used to highlight where the best results have been obtained for the current set of experiments, and italic font indicates the best known results.

Clearly, the GSA outperforms **Desatur** and iterated greedy on most instances, and the version with crossover works much better than the one without. However, the GSA results do not match the results obtained by the HEA algorithm, which is clearly state-of-the-art. Nevertheless, the benchmarks are tough and the results are generally good. The HEA algorithm uses many thousands of tabu search iterations following the creation of each new offspring in the population, thus it is a very different type of algorithm from the current order based GSA. The new order based approach introduced in the present Chapter is presented largely for its generic qualities, and its potential for a wide range of set partitioning problems.

Some Further Experiments

In addition to the DIMACs benchmarks, further experiments were undertaken on two special types of graphs first presented by [39] and used by [14] to test his version of the grouping genetic algorithm. These instances are all arranged

around cliques – that is, complete subgraphs present within each instance. The two types are called the ‘pyramidal chain’ and the ‘sequences of cliques’. In all cases the chromatic number c is known beforehand. A simple order based GA (in other words, without the GSA cooling schedule) easily solved all instances tried: one pyramidal chain instance with $c = 6$, 20 cliques, 60 nodes and 200 edges; seven instances of sequences of cliques with $c = 6$, 20 cliques and 120 nodes. All the instances that could be found were kindly supplied by Erben, using email attachment. The pyramidal chain example needed about 1,300 evaluation steps of the order based GA, a similar number to that was reported by [14] for the grouping genetic algorithm. For the 7 sequence of cliques examples, however, the order based approach needed a maximum of 10,000 evaluations, which is less than the 150,000 reported in [14].

10.2 Bin Packing

Two sources of data provide the benchmark instances for the bin packing tests. Once again, an order based GSA is used with inversion providing the mutation operator. POP1 is chosen as the crossover operator for bin packing, because it produced better results than MIS in some preliminary tests. The fitness function adopted for bin packing is the one devised by Falkenauer (see Eqn. (4)) which favors bins that are as full as possible. A single iteration of local search, following each crossover, seems to be sufficient for bin packing. This time the ‘largest first’ reordering heuristic in Fig. 5 is replaced by a reordering scheme based on largest total bin weight (or fullest bin).

Many authors have noted the efficiency of the simple bin packing heuristic algorithm, first fit decreasing weight, FFD, (briefly discussed earlier in Sect. 8). For large numbers of items the worst case solution is $\frac{11}{9} \times OPT$, [7], where ‘ OPT ’ refers to the optimum solution. However, best case and average case behavior tend to be very much better than this, with FFD easily solving many problems to optimality. [41] coined the phrases ‘FFD-easy’, ‘FFD-hard’ and ‘extremely FFD-hard’ to help classify problems with different properties according to the proportion solved by FFD:

- FFD-easy: 80 - 100 % solved
- FFD-hard: 20 - 80 % solved
- extremely FFD-hard: 0 - 20 % solved

The Data Sets of Scholl and Klein (SK)

Scholl and Klein provide three data sets with a total of 1,010 of bin packing instances on their web site, with optimum solutions (that is, minimum number of bins) given for each (see the Resources Appendix). All these instances have been generated in groups of either 10 or 20, so that instances within groups have the same properties regarding bin capacities, numbers of items and ranges of weights for the items. The majority of these instances are easily

solved, however. Indeed the present author found optimum solutions to 781 of the 1,010 instances using FFD. All the test data selected for this Chapter belong to classes where FFD has solved zero instances. The first two instances, N4C3W2_A and N4C3W4_A, are taken from SK's first data set. N4C3W2_A has N = 500 items, bin capacity C = 150, and item weights varying uniformly between 20 and 100. N4C3W4_A has the same values for N and C, but the item weights are between 30 and 100. The next six instances are all taken from the second SK data set. All these instances have N = 500 items and bin capacity = 1,000. The average weight per item varies according to W, with W1 = 1000/3, W2 = 1000/5, W3 = 1000/7, and W4 = 1000/9. Thus, for N4W1B1R0 we would expect to find a maximum of 3 items in each bin, and for N4W2B1R0 about 5 items per bin, and so on. The value of B indicates the maximum deviation of single weight values from the average weight. B1 = 20%, B2 = 50%, and B3 = 90%. For the remaining instances (HARD0 - HARD9 from data set 3) the parameters are: N = 200 items, capacity C = 100,000, and weights range from 20,000 to 35,000. The number of items per bin lies between 3 and 5.

Results presented in Table 2 show that both the GSA and mutation only GSA are able to solve most of the instances to optimality, and get very close for the others. For the SK data, crossover does not appear to make a significant contribution however, although the GSA clearly produces better solutions than FFD and iterated greedy. Bold font is used as previously to highlight the best results for current experiments, and the best known results, respectively.

Table 2. GSA results on Scholl and Klein's Bin packing instances

Instance	Order based GSA				Mut GSA		It greed		FFD	Optimum
	# Gens	Time	Mean	Min	Mean	Min	Mean	Min		
N4C3W2_A	2000	324	204	204	204	204	204.6	204	206	<i>203</i>
N4C3W4_A	2000	340	217	217	217	217	219	219	220	<i>216</i>
N4W1B1R0	1000	134	167	167	167	167	184	184	184	<i>167</i>
N4W2B1R0	1000	89	102	102	102	102	107.3	105	109	<i>101</i>
N4W3B1R0	1000	73	71	71	71	71	73	73	74	<i>71</i>
N4W3B2R0	1000	73	71	71	71	71	71	71	72	<i>71</i>
N4W4B1R0	1000	63	56	56	56	56	56.8	56	58	<i>56</i>
HARD0	1000	30	56	56	56	56	59	59	59	<i>56</i>
HARD1	1000	30	57	57	57	57	58.7	57	60	<i>57</i>
HARD2	1000	30	57	57	57	57	59	59	60	<i>56</i>
HARD3	1000	30	56	56	56	56	57.7	57	59	<i>55</i>
HARD4	1000	30	57	57	57	57	58.9	58	60	<i>57</i>
HARD5	1000	30	56	56	56	56	57.8	57	59	<i>56</i>
HARD6	1000	30	57	57	57	57	58.6	57	60	<i>57</i>
HARD7	1000	30	55	55	55	55	58	58	59	<i>55</i>
HARD8	1000	30	57	57	57	57	58.8	58	60	<i>57</i>
HARD9	1000	30	56	56	56	56	58.7	58	60	<i>56</i>

Falkenauer's Data Sets

Falkenauer's data sets are also included here to make comparisons possible with state-of-the-art algorithms, such as the MTP algorithm by [30] and the hybrid grouping genetic algorithm (HGGA) of [16]. Falkenauer generated two types of data:

1. uniform item size distribution, and
2. triplets.

Both types were produced along similar lines to the hard instances in the SK data sets. For the first set of data, items are uniformly distributed between 20 and 100, with bin capacity 150. Falkenauer generated instances with varying numbers of items (120, 250, 500 and 1,000), producing 20 examples of each, making 80 instances of this type in total.

With the second set of data, item weights were drawn from the range 0.25 to 0.5 to be packed in bins of capacity 1. Given the improbability of finding four items of weight exactly 0.25, it follows that a well-filled bin will normally contain one big item (larger than a third of the bin capacity) and two small ones (each smaller than a third of the bin capacity), which is why the instances are referred to as 'triplets'. What makes this class difficult is that putting two big items or three small items into a bin is possible but inevitably leads to wasted space, because the remaining space is less than 0.25, and thus cannot be filled. Falkenauer generated instances with known optima, based on a bin capacity of 1,000, as follows. An item was first generated with a size drawn uniformly from the range [380, 490], leaving space s of between 510 and 620 in the bin. The size of the second item was drawn uniformly from [250, $s/2$]. The weight of the third item was then chosen to completely fill the bin. This process was repeated until the required number of items had been generated. The number of bins needed was subsequently recorded. Triplets were generated with 60, 120, 249 and 501 items – 20 instances of each. Optimum solutions are 20, 40, 83 and 167 bins, respectively.

Table 3 compares the results obtained by running the GSA with those published in [16]. Results for FFD and iterated greedy are also included. Each algorithm was run only once on each instance, and population sizes for the GSA were set at 100, the same as was used for HGGA. The GSA was run for the same number of generations as the HGGA, 2,000 for the first two data sets, 5,000 for the next two, 1,000 for the first two triplet groups, and 2,000 for the last two.

Once again the GSA clearly outperforms FFD and iterated greedy (run for the same length of time as the GSA), and it would appear that POP1 crossover makes a useful contribution because the GSA with crossover does slightly better than the GSA without it. The GSA clearly performs better than MTP in all columns. However, apart from the uniform instances with 120 items, the HGGA performs slightly better than the GSA. It is worth

Table 3. Results for Falkenauer’s data sets

Type	# items	FFD	MTP	HGGA	GSA	Mut GSA	IG
Uniform	120	49.75	49.15	49.15	<i>49.1</i>	49.45	49.4
Uniform	250	103.1	102.15	<i>101.7</i>	101.9	102.5	102.3
Uniform	500	203.9	203.4	<i>201.2</i>	201.5	202.5	202.65
Uniform	1000	405.4	404.45	<i>400.55</i>	401.3	402.7	403.7
Triplets	60	23.2	21.55	<i>20.1</i>	21	21	22.25
Triplets	120	45.8	44.1	<i>40</i>	41	41	44.95
Triplets	249	95	90.45	<i>83</i>	84	84.15	93.7
Triplets	501	190.15	181.85	<i>167</i>	168	169.1	188.6

noting, however, that the HGGA employs specialized backtracking in its local search, that will unpack up to 3 items per bin and try to repack. On the other hand the order based GSA does not use backtracking and runs very fast – requiring one or two seconds for the smaller problems and up to a maximum of about 22 minutes for some of the uniform problems with 1,000 items. Furthermore, the representation, operators and CL heuristics used in the GSA are more generic, and can equally be applied to other set partitioning problems, as previously mentioned.

10.3 Timetabling

Recall the version of the timetabling problem addressed here combines bin packing with graph coloring. The maximum number of seats per time slot corresponds to the BPP constraint, and the avoidance of clashes to the GCP constraint. Given a set of students to be examined for different courses, we wish to schedule the examinations so that all clashes are avoided and the seating capacity is not exceeded in any time period. A selection of real world instances from Carter’s benchmarks [6] was thought to provide a suitable challenge for the new order based approach (see the Resources Appendix). Only those instances for which maximum seating capacity has been specified have been chosen, and the main characteristics of these six problems are summarized in Table 4. The first five columns of this table are self explanatory, and column 6 lists the best known solutions to the underlying graph coloring instances. The *uta-s-92* best GCP is taken from [5], *pur-s-92* from [3], and the other four graph coloring solutions from [6]. Column 7 presents solutions to the underlying bin packing instances, as calculated with a simple FFD algorithm by the present author. Interestingly, every one of the BPP solutions obtained by FFD match the so called ‘ideal solutions’, found simply by counting the total number of student-examination events and filling up the seats in consecutive time slots, ignoring any clashes, until all the events are used up. Thus, all the solutions in column 7 are optimal for the underlying bin packing problem. Assuming that the graph coloring solutions in column 6

Table 4. Characteristics of timetabling problems

Instance	# exams	# students	# edges	̃seats	GCP slots	BPP slots
car-f-92	543	18419	20305	2000	28*	28*
car-s-91	682	16926	29814	1550	28	37*
kfu-s-93	461	5349	5893	1955	19*	13
pur-s-93	2419	30032	86261	5000	30*	25
tre-s-92	261	4362	6131	655	20	23*
uta-s-92	622	21266	24249	2800	30*	22

Table 5. Results for timetabling problems

Instance	Order Based GSA				Mut GSA		It Greed	
	# Gens	Time	Mean	Min	Mean	Min	Mean	Min
car-f-92	2000	1513	30.5	30	30.7	30	30.9	30
car-s-91	2000	2254	38.0	38	38.0	38	38.0	38
kfu-s-93	2000	1130	<i>19.0</i>	<i>19</i>	<i>19.0</i>	<i>19</i>	<i>19.0</i>	<i>19</i>
pur-s-93	2000	22527	33.6	33	33.3	33	33.7	33
tre-s-92	2000	376	23.8	<i>23</i>	23.4	<i>23</i>	23.8	<i>23</i>
ta-s-92	2000	2277	30.8	<i>30</i>	30.8	<i>30</i>	30.8	<i>30</i>

are also optimal, we can say that the larger solutions of GCP and BPP gives a lower bound for the corresponding timetabling problem (indicated with a ‘*’, in Table 4).

Results for the GSA on Carter’s instances are presented in Table 5. The table also shows results for a mutation-only version of the GSA and a pure iterated greedy algorithm. As before, the same run time was used for the GSA and iterated greedy algorithm. A population size of 200 was used for both versions of the GSA, and each experiment was run for 2000 generations. The form of local search used for the previous experiments in graph coloring and bin packing was altered slightly for the timetabling problem. Recall that we used one to three iterations of a local search based on reordering the classes according to some form of ‘largest first’ criterion – either decreasing total degree (for the GCP) or the ‘fullest bin first’ (for the BPP). Preliminary experiments with the timetabling instances showed that better results could be obtained if 5 iterations of iterated greedy were used, instead of the usual local search, with largest:reverse:random set at 50:50:30. The ‘fullest bin first’ approach, as used for the BPP replaced the ‘largest first’ reordering heuristic in the iterated greedy routine, for the GSA and the iterated greedy proper. Here the fullest bin corresponds to the time slot with the largest number of students taking examinations. Additionally, Falkenauer’s bin packing fitness function of (see Eqn. (4)) was used for the timetabling instances, which favored full time slots. POP1 crossover was used, together with insertion mutation.

It is clear examining Table 5 that the results are very similar for both versions of the GSA and also iterated greedy. The values highlighted in italics denote optimum solutions. It would appear that these particular instances may be easy for all three algorithms.

11 Summary

This Chapter has introduced a new and generic order based framework suitable for application, with minimum adaptation, to a wide range of set partitioning problems. The approach contrasts with other state-of-the-art techniques that rely mostly on direct representations. A clear advantage of using an order based approach is that every permutation is decoded as a feasible solution, meaning no costly repair mechanisms are required, following a crossover event, however heavily constrained the problem. Perhaps the most innovative feature of the new order based approach is the inclusion of some simple grouping and reordering heuristics to preprocess the chromosomes and make them more amenable to crossover. The idea is to encourage the transmission of whole set partitions, from parent to offspring, in a way that is not usually possible with an order based approach. Results presented herein indicate that the new memetic algorithm is highly competitive, yet no lengthy problem-specific local search procedure is required. All that is needed are a very few iterations of Culberson and Luo's heuristics for preprocessing the chromosomes prior to crossover. Thus, although the exact choice of objective or fitness function will vary according to the specific set partitioning application, problem-specific heuristics and costly backtracking – so common in other approaches – can largely be avoided.

A detailed examination of the results reveals that different components of the GSA framework are more or less effective, depending on the nature of the test instances. The six timetabling instances, for example, seem to be rather easy for the iterated greedy algorithm to solve, making it difficult to assess the potential of the GSA – it is possible that more challenging instances are needed here. On the other hand, the GSA with crossover proved very effective on the DIMACS graph coloring instances and also on the bin packing instances supplied by Falkenauer. Yet the need for crossover was not particularly well established for the bin packing data sets of Scholl and Klein.

Future work will concentrate on improving results for set partitioning benchmarks, and undertaking further investigations into the relative contributions of genetic operators versus simple reordering heuristics. The challenge will be to make improvements to the approach, while keeping the techniques as generic as possible, avoiding time-consuming backtracking and repair wherever possible. The present author also plans to extend the new approach to more realistic timetabling problems, incorporating additional hard constraints as well as a range of soft constraints. Order based approaches have a distinct

advantage over techniques that use a direct representation when dealing with multiply constrained problems: a suitable greedy decoder can ensure that only feasible solutions are generated. On the other hand, a directly encoded method may struggle to find *any* feasible solution in similar circumstances. Multi-objective optimization is of particular interest when several soft constraints conflict. For example, a favorable spread of examinations to allow students revision gaps may conflict with the interests of the teaching staff who may prefer examinations with many students to be held early on, to give sufficient time for marking. Extending the order based techniques to other set partitioning problems is another interesting priority for the future.

References

1. Anderson PG, Ashlock D (2004) *Advances in ordered greed*. (available online at <http://www.cs.rit.edu/~pga/abstracts.php> – last accessed: 28 February 2007)
2. Brélaz D (1979) New methods to color the vertices of graphs. *Communications ACM*, 24(4): 251–256.
3. Burke E, Newell J (1999) A multi-stage evolutionary algorithm for the timetabling problem. *IEEE Trans. Evolutionary Computation*, 3(1): 63–74.
4. Burke E, Petrovic S (2002) Recent research directions in automated timetabling. *European J. Operational Research*, 140(2): 266–280.
5. Caramia M, Dell’Olmo P, Italiano G (2000) New algorithms for examination timetabling. In: *Proc. 4th Intl. Workshop Algorithm Engineering*, 5–8 September, Lecture Notes in Computer Science 1982, Springer, London, UK: 230–240.
6. Carter MW, Laporte G, Lee SY (1996) Examination timetabling: algorithms, strategies and applications. *European J. Operational Research*, 47: 373–383.
7. Coffman EG, Garey MR, Johnson DS (1984) Approximation algorithms for bin packing – an updated survey. In: Ausiello G, Lucertini M, Serafini P (eds) *Algorithm Design for Computer System Design*: Springer-Verlag, Berlin: 49–106.
8. Croitoru C, Luchian H, Gheorghies O, Apetrei A (2002) A new genetic graph coloring heuristic. In: *Proc. Computational Symp. Graph Coloring and its Generalizations*, Ithaca, NY: 63–74.
9. Culberson J, Luo F (1996) Exploring the k -colorable landscape with iterated greedy. In: Johnson DS, Trick MA (eds) (1996) *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26*. American Mathematical Society, Providence, RI: 499–520.
10. Davis L (1985) Applying adaptive algorithms to epistatic domains. In: *Proc. Intl. Joint Conf. Artificial Intelligence*, Los Angeles, CA: 162–164
11. Davis L (1991) Order based genetic algorithms and the graph coloring problem. In: *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY: 72–90.
12. Dorigo M, Maniezzo V, Colomi A (1996) The ant system: optimization by a colony of cooperating agents. *IEEE Trans. System Man Cybernetics – Part B*, (26): 29–41.
13. Eiben A, der Hauw JV, Hemert JV (1998) Graph coloring with adaptive evolutionary algorithms. *J. Heuristics*, 4: 25–46.

14. Erben W (2001) A grouping genetic algorithm for graph colouring and exam timetabling. In: *Practice and Theory of Automated Timetabling – Proc. PATAT2000*, Lecture Notes in Computer Science 2079, Springer-Verlag, Berlin: 132–156.
15. Falkenauer E (1995) Solving equal piles with the grouping genetic algorithm. In: Eshelman LJ (ed) *Proc. 6th Intl. Conf. Genetic Algorithms – ICGA*, 15–19 July, San Francisco, CA, Morgan Kaufmann: 492–497.
16. Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. *J. Heuristics*, 2: 5–30.
17. Fogel L, Owens A, Walsh M (1966) *Artificial Intelligence Through Simulated Evolution*. Wiley, New York, NY.
18. Galinier P, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. *J. Combinatorial Optimization*, 3(4): 379–397.
19. Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Boston, MA.
20. Goldberg DE, Lingle R (1985) Alleles, loci and the traveling salesman problem. In: *Proc. Intl. Conf. Genetic Algorithms and their Applications*, Lawrence Erlbaum Associates, Mahwah, NJ: 154–159.
21. Greenwood GW, Tyrrell AM (2006) *Introduction to Evolvable Hardware: a Practical Guide for Designing Self-Adaptive Systems*. Wiley-IEEE Press, Chichester, UK.
22. Holland JH (1975) *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI.
23. Hurley S, Smith D, Thiel S (1997) Fasoft: a system for discrete channel frequency assignment. *Radio Science*, 32(5): 1921–1940.
24. Johnson DS, Trick MA (eds) (1996) *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26*. American Mathematical Society, Providence, RI.
25. Jones D, Beltramo M (1991) Solving partitioning problems with genetic algorithms. In: Belew RK, Booker LB (eds) *Proc. 4th Intl. Conf. Genetic Algorithms*, July, San Diego, CA, Morgan Kaufmann: 442–449.
26. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: *Proc. IEEE Intl. Conf. Neural Networks*, IEEE Computer Society Press, Piscataway, NJ: 1942–1948.
27. Kirkpatrick S, Gelatt C, Vecchi M (1983) Optimization by simulated annealing. *Science*, 220(4598): 671–680.
28. Langton C (ed) (1995) *Artificial Life: an Overview*. MIT Press, Cambridge, MA.
29. Lewis R, Paechter B (2005) Application of the grouping genetic algorithm to university course timetabling. In: Raidl GR, Gottlieb J (eds) *Evolutionary Computation in Combinatorial Optimization (Proc. 5th European Conf. – EvoCOP 2005)*, 30 March – 1 April, Lausanne, Switzerland. Lecture Notes in Computer Science 3448, Springer-Verlag, Berlin: 144–153.
30. Martello S, Toth P (1990) *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester, UK.
31. Matula D, Marble G, Isaacson J (1972) Graph coloring algorithms. In: Read R (ed) *Graph Theory and Computing*, Academic Press, New York, NY: 104–122.
32. Michalewicz Z (1996) *Genetic Algorithms + Data Structures = Evolutionary Programs 3rd ed.* Springer-Verlag, London, UK.
33. Mitchell M (1996) *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.

34. Mumford C (2006) New order based crossovers for the graph coloring problem. In: Runarsson TP, Beyer H-G, Burke EK, Guervós JJM, Whitley LD, Yao X (eds) *Parallel Problem Solving from nature (Proc. 9th Intl. Conf. Parallel Problem Solving from Nature Conf. – PPSN IX)*, 9–13 September, Reykjavik, Iceland. Lecture Notes in Computer Science 4193, Springer-Verlag, Berlin: 880–889.
35. Oliver IM, Smith DJ, Holland JH (1987) A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette JJ (ed) *Genetic Algorithms and their Applications (Proc. 2nd Intl. Conf. Genetic Algorithms)*, July, Cambridge, MA, Lawrence Erlbaum Associates, Mahwah, NJ: 224–230.
36. Pankratz G (2005) A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27(1): 21–41.
37. Rechenberg I (1965) Cybernetic solution path of an experimental problem. *Technical Report 1122*, Ministry of Aviation, Royal Aircraft Establishment, Farnborough, Hants, UK, August.
38. Rekiek B, Lit PD, Pellichero F, Falkenauer E, Delchambre A (1999) Applying the equal piles problem to balance assembly lines. In: *Proc. 1999 IEEE Intl. Symp. Assembly and Task Planning – ISATP’99*, Portugal, IEEE Computer Society Press, Piscataway, NJ: 399–404.
39. Ross P, Hart E, Corne D (1998) Some observations about ga-based exam timetabling. In: *Selected Papers from Practice and Theory of Automated Timetabling II – PATAT 1997*, Lecture Notes in Computer Science 1408, Springer-Verlag, London, UK: 115–129.
40. Schaerf A (1999) A survey of automated timetabling. *Artificial Intelligence Review*, 13: 87–127.
41. Schwerin P, Wäscher G (1997) The bin-packing problem: a problem generator and some numerical experiments with ffd packing and mtp. *Intl. Trans. Operational Research*, 4(5-6): 377–389.
42. Valenzuela CL (2001) A study of permutation operators for minimum span frequency assignment using an order based representation. *J. Heuristics*, 7(1): 5–22. (CL Valenzuela is now known as CL Mumford).
43. Welsh D, Powell M (1967) An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10: 85–86.

Resources

1 Key Books

De Jong KA (2006) *Evolutionary Computation: a Unified Approach*. MIT Press, Cambridge, MA.

Eiben AE, Smith JE (2003) *Introduction to Evolutionary Computing*. Springer-Verlag, New York, NY.

Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Boston, MA.

Holland JH (1992) *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA.

Mitchell M (1998) *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.

Zbigniew M (1996) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, NY.

2 Key International Conferences

EvoCOP 2006: 6th European Conference on Evolutionary Computation in Combinatorial Optimization.

<http://evonet.lri.fr/eurogp2006/>

GECCO 2006: Genetic and Evolutionary Computation Conference.

<http://www.sigevo.org/gecco-2006/>

IEEE WCCI 2006: World Congress on Computational Intelligence.
<http://www.compsys.ia.ac.cn/wcci2006/>

PATAT 2006: The 6th International Conference for the Practice and Theory of Automated Timetabling.
<http://www.asap.cs.nott.ac.uk/patat/patat06/patat06.shtml>

PPSN IX 2006: Parallel Problem Solving from Nature.
<http://ppsn2006.raunvis.hi.is/>

3 Interest Groups/Web sites

SIGEVO: ACM Special Interest Group on Genetic and Evolutionary Computation.
<http://www.sigevo.org/>

IEEE CIS: IEEE Computational Intelligence Society.
<http://iee-cis.org/pubs/tec/>

4 (Open Source) Software

GAGS: A genetic algorithm C++ class library
<http://kal-el.ugr.es/GAGS/>

GAlib: A C++ Library of Genetic Algorithm Components
<http://lancet.mit.edu/ga/>

GAJIT: A Simple Java Genetic Algorithms Package
<http://www.micropraxis.com/gajit/index.html>

GA Playground (Genetic Algorithms Toolkit): Java genetic algorithms toolkit.
<http://www.aridolan.com/ga/gaa/gaa.html>

JAGA: Java API for genetic algorithms
<http://jaga.sourceforge.net/>

Java genetic algorithms package
<http://jgap.sourceforge.net/>

GATbx: Genetic Algorithm Toolbox for use with MATLAB.
<http://www.shef.ac.uk/acse/research/ecrg/gat.html>

GAOT: Genetic Algorithm Optimization Toolbox for use with MATLAB.
<http://www.ise.ncsu.edu/mirage/GAToolBox/gaot/>

5 Data Sets used in the Chapter

Timetabling:

<http://www.cs.nott.ac.uk/~rxq/data.htm>

Bin Packing – Scholl and Klein:

<http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm>

Bin Packing – Falkenauer:

<http://people.brunel.ac.uk/~mastjjb/jeb/orlib/binpackinfo.html>

DIMACS Challenge GCP:

<ftp://dimacs.rutgers.edu/pub/challenge/graph/>

Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications

William B. Langdon¹, Riccardo Poli², Nicholas F. McPhee³,
and John R. Koza⁴

¹ Departments of Biological and Mathematical Sciences, University of Essex, UK,
wlangdon@essex.ac.uk

² Department of Computing and Electronic Systems, University of Essex, UK,
rpoli@essex.ac.uk

³ Division of Science and Mathematics, University of Minnesota, Morris, USA,
mcphee@morris.umn.edu

⁴ Stanford University, Stanford, CA, USA, john@johnkoza.com

1 Introduction

The goal of having computers automatically solve problems is central to artificial intelligence, machine learning, and the broad area encompassed by what Turing called ‘machine intelligence’ [384]. Machine learning pioneer Arthur Samuel, in his 1983 talk entitled ‘AI: Where It Has Been and Where It Is Going’ [337], stated that the main goal of the fields of machine learning and artificial intelligence is:

“to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence.”

Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without having to tell the computer explicitly how to do it. At the most abstract level GP is a *systematic, domain-independent* method for getting computers to *automatically* solve problems starting from a *high-level statement* of what needs to be done.

Over the last decade, GP has attracted the interest of streams of researchers around the globe. This Chapter is intended to give an overview of the basics of GP, to summarize important work that gave direction and impetus to research in GP as well as to discuss some interesting new directions and applications. Things change fast in this field, as investigators discover new ways of doing things, and new things to do with GP. It is impossible to cover all aspects of this area, even within the generous page limits of this chapter. Thus this

Algorithm 1 Abstract GP algorithm

-
- 1: Randomly create an *initial population* of programs from the available primitives (see Sect. 2.2).
 - 2: **repeat**
 - 3: *Execute* each program and ascertain its fitness.
 - 4: *Select* one or two program(s) from the population with a probability based on fitness to participate in genetic operations (see Sect. 2.3).
 - 5: Create new individual program(s) by applying *genetic operations* with specified probabilities (see Sect. 2.4).
 - 6: **until** an acceptable solution is found or some other stopping condition is met (for example, reaching a maximum number of generations).
 - 7: **return** the best-so-far individual.
-

Chapter should be seen as a snapshot of the view we, the authors, have at the time of writing.

1.1 GP in a Nutshell

Technically, GP is a special evolutionary algorithm (EA) where the individuals in the population are *computer programs*. So, generation by generation GP *iteratively* transforms populations of programs into other populations of programs as illustrated in Fig. 1. During the process, GP constructs new programs by applying genetic operations which are specialized to act on computer programs.

Algorithmically, GP comprises the steps shown in Algorithm 1. The main genetic operations involved in GP (line 5 of Algorithm 1) are the following:

- **Crossover:** the creation of one or two offspring programs by recombining randomly chosen parts from two selected programs.
- **Mutation:** the creation of one new offspring program by randomly altering a randomly chosen part of one selected program.

Some GP systems also support structured solutions (see, for example, Sect. 5.1), and some of these then include *architecture-altering operations*

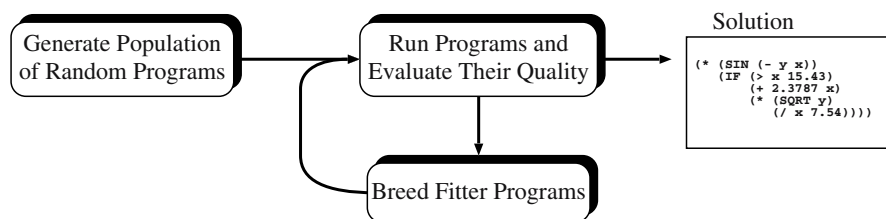


Fig. 1. GP main loop

which randomly alter the architecture (for example, the number of subroutines) of a program to create a new offspring program. Also, often, in addition of crossover, mutation and the architecture-altering operations, an operation which simply copies selected individuals in the next generation is used. This operation, called *reproduction*, is typically applied only to produce a fraction of the new generation.

1.2 Overview of the Chapter

This Chapter starts with an overview of the key representations and operations in GP (Sect. 2), a discussion of the decisions that need to be made before running GP (Sect. 3), and an example of a GP run (Sect. 4).

This is followed by descriptions of some more advanced GP techniques including: automatically defined functions (Sect. 5.1) and architecture-altering operations (Sect. 5.2), the GP problem solver (Sect. 5.3), systems that constrain the syntax of evolved programs in some way (for instance, using grammars or type systems; Sect. 5.4) and developmental GP (Sect. 5.5). Alternative program representations, namely linear GP (Sect. 6.1) and graph-based GP (Sect. 6.2) are then discussed.

After this survey of representations, we provide a review of the enormous variety of applications of GP, including curve fitting and data modeling (Sect. 7.1), human competitive results (Sect. 7.2) and much more, and a substantial collection of ‘tricks of the trade’ used by experienced GP practitioners (Sect. 8). We also give an overview of some of the considerable work that has been done on the theory of GP (Sect. 9).

After concluding the Chapter (Sect. 10), we provide a resources appendix that reviews the many sources of further information on GP, its applications, and related problem solving systems.

2 Representation, Initialization and Operators in Tree-Based GP

In this Section we will introduce the basic tools and terms used in genetic programming. In particular, we will look at how solutions are represented in most GP systems (Sect. 2.1), how one might construct the initial, random population (Sect. 2.2), and how selection (Sect. 2.3) as well as recombination and mutation (Sect. 2.4) are used to construct new individuals.

2.1 Representation

In GP programs are usually expressed as *syntax trees* rather than as lines of code. Figure 2 shows, for example, the tree representation of the program $\max(x*x, x+3*y)$. Note how the variables and constants in the program

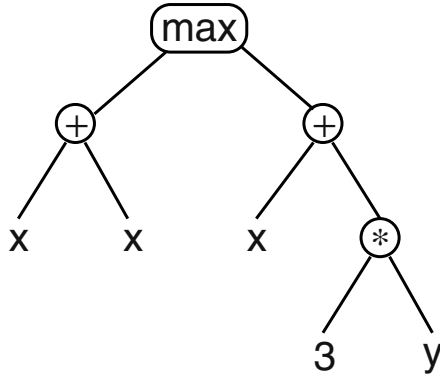


Fig. 2. GP syntax tree representing $\max(x \cdot x, x + 3 \cdot y)$

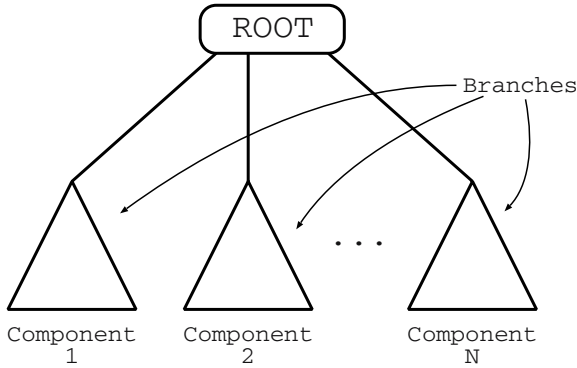


Fig. 3. Multi-component program representation

(x , y , and 3), called *terminals* in GP, are leaves of the tree, while the arithmetic operations ($+$, $*$, and \max) are internal nodes (typically called *functions* in the GP literature). The sets of allowed functions and terminals together form the *primitive set* of a GP system.

In more advanced forms of GP, programs can be composed of multiple components (say, subroutines). In this case the representation used in GP is a set of trees (one for each component) grouped together under a special root node that acts as glue, as illustrated in Fig. 3. We will call these (sub)trees *branches*. The number and type of the branches in a program, together with certain other features of the structure of the branches, form the *architecture* of the program.

It is common in the GP literature to represent expressions in a *prefix* notation similar to that used in Lisp or Scheme. For example, $\max(x \cdot x, x + 3 \cdot y)$ becomes $(\max (* x x) (+ x (* 3 y)))$. This notation often makes it easier to see the relationship between (sub)expressions and their corresponding

(sub)trees. Therefore, in the following, we will use trees and their corresponding prefix-notation expressions interchangeably.

How one implements GP trees will obviously depend a great deal on the programming languages and libraries being used. Most traditional languages used in AI research (such as Lisp and Prolog), many recent languages (say Ruby and Python), and the languages associated with several scientific programming tools (namely, MATLAB® and Mathematica®) provide automatic garbage collection and dynamic lists as fundamental data types making it easy to directly implement expression trees and the necessary GP operations. In other languages one may have to implement lists/trees or use libraries that provide such data structures.

In high performance environments, however, the tree-based representation may be too memory-inefficient since it requires the storage and management of numerous pointers. If all functions have a fixed arity (which is extremely common in GP applications) the brackets become redundant in prefix-notation expressions, and the tree can be represented as a simple linear sequence. For example, the expression $(\max (* x x) (+ x (* 3 y)))$ could be written unambiguously as the sequence $\max * x x + x * 3 y$. The choice of whether to use such a linear representation or an explicit tree representation is typically guided by questions of convenience, efficiency, the genetic operations being used (some may be more easily or more efficiently implemented in one representation), and other data one may wish to collect during runs (for instance, it is sometimes useful to attach additional information to nodes, which may require that they be explicitly represented). There are also numerous high-quality, freely available GP implementations (see the resources in the appendix at the end of this chapter for more information).

While these tree representations are the most common in GP, there are other important representations, some of which are discussed in Sect. 6.

2.2 Initializing the Population

Similar to other evolutionary algorithms, in GP the individuals in the initial population are randomly generated. There are a number of different approaches to generating this random initial population. Here we will describe two of the simplest (and earliest) methods (the *Full* and *Grow* methods), and a widely used combination of the two known as *Ramped half-and-half*.

In both the *Full* and *Grow* methods, the initial individuals are generated subject to a pre-established maximum depth. In the *Full* method (so named because it generates full trees) nodes are taken at random from the function set until this maximum tree depth is reached, and beyond that depth only terminals can be chosen. Figure 4 shows snapshots of this process in the construction of a full tree of depth 2. The children of the $*$ node, for example,

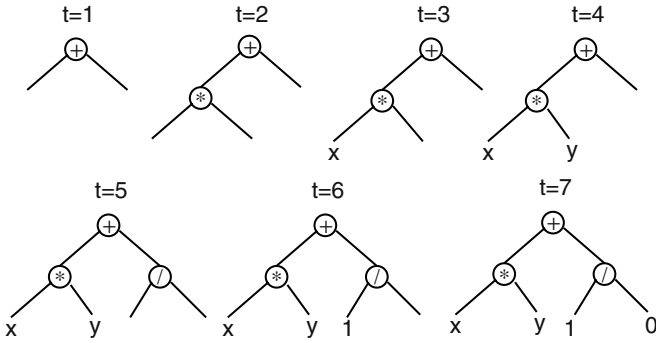


Fig. 4. Creation of a full tree having maximum depth 2 (and therefore a total of seven nodes) using the Full initialization method ($t=time$)

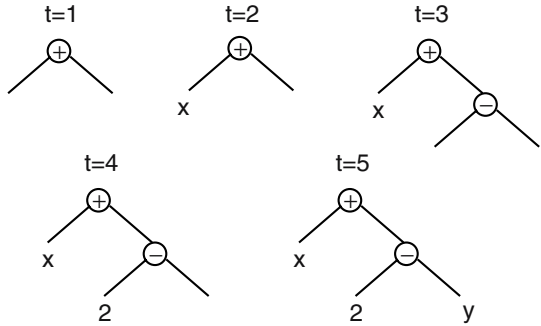


Fig. 5. Creation of a five node tree using the Grow initialization method with a maximum depth of 2 ($t=time$). A terminal is chosen at $t = 2$, causing the left branch of the root to be closed at that point even though the maximum depth had not been reached

must be leaves, or the resulting tree would be too deep; thus at time $t = 3$ and time $t = 4$ terminals must be chosen (x and y in this case).

Where the *Full* method generates trees of a specific size and shape, the *Grow* method allows for the creation of trees of varying size and shape. Here nodes are selected from the whole primitive set (functions *and* terminals) until the depth limit is reached, below which only terminals may be chosen (as is the case in the *Full* method). Figure 5 illustrates this process for the construction of a tree with depth limit 2. Here the first child of the root $+$ node happens to be a terminal, thus closing off that branch before actually reaching the depth limit. The other child, however, is a function ($-$), but its children are forced to be terminals to ensure that the resulting tree does not exceed the depth limit.

Pseudo code for a recursive implementation of both the *Full* and *Grow* methods is given in Algorithm 2.

Algorithm 2 Pseudo code for recursive program generation with the *Full* and *Grow* methods

```

procedure: gen_rnd_expr( func_set, term_set, max_d, method )
1: if max_d = 0 or ( method = grow and rand() <  $\frac{|term\_set|}{|term\_set|+|func\_set|}$  ) then
2:   expr = choose_random_element( term_set )
3: else
4:   func = choose_random_element( func_set )
5:   for i = 1 to arity(func) do
6:     arg_i = gen_rnd_expr( func_set, term_set, max_d - 1, method );
7:   expr = (func, arg_1, arg_2, ...);
8: return expr

```

Notes: **func_set** is a function set, **term_set** is a terminal set, **max_d** is the maximum allowed depth for expressions, **method** is either *Full* or *Grow* and **expr** is the generated expression in prefix notation.

Note here that the size and shapes of the trees generated via the *Grow* method are highly sensitive to the sizes of the function and terminal sets. If, for example, one has significantly more terminals than functions, the *Grow* method will almost always generate very short trees regardless of the depth limit. Similarly, if the number of functions is considerably greater than the number of terminals, then the *Grow* method will behave quite similarly to the *Full* method. While this is a particular problem for the *Grow* method, it illustrates a general issue where small (and often apparently inconsequential) changes such as the addition or removal of a few functions from the function set can in fact have significant implications for the GP system, and potentially introduce important unintended biases.

Because neither the *Grow* or *Full* method provide a very wide array of sizes or shapes on their own, Koza proposed a combination called *ramped half-and-half* [188]. Here half the initial population is constructed using *Full* and half is constructed using *Grow*. This is done using a range of depth limits (hence the term ‘ramped’) to help ensure that we generate trees having a variety of sizes and shapes.

While these methods are easy to implement and use, they often make it difficult to control the statistical distributions of important properties such as the sizes and shapes of the generated trees. Other initialization mechanisms, however, have been developed (such as [239]) that do allow for closer control of these properties in instances where such control is important.

It is also worth noting that the initial population need not be entirely random. If something is known about likely properties of the desired solution, trees having these properties can be used to seed the initial population. Such seeds might be created by humans based on knowledge of the problem domain, or they could be the results of previous GP runs. However, one needs to be careful not to create a situation where the second generation is dominated

by offspring of a single or very small number of seeds. Diversity preserving techniques, such as multi-objective GP [287, 344], demes [203] (see Sect. 8.6), fitness sharing [115] and the use of multiple seed trees, might be used. In any case, the diversity of the population should be monitored to ensure that there is significant mixing of different initial trees.

2.3 Selection

Like in most other EAs, genetic operators in GP are applied to individuals that are probabilistically selected based on fitness. That is, better individuals are more likely to have more child programs than inferior individuals. The most commonly employed method for selecting individuals in GP is tournament selection, followed by fitness-proportionate selection, but any standard EA selection mechanism can be used. Since selection has been described elsewhere in this book (in the Chapters on EAs), we will not provide any additional details.

2.4 Recombination and Mutation

Where GP departs significantly from other EAs is in the implementation of the operators of crossover and mutation. The most commonly used form of crossover is *subtree crossover*. Given two parents, subtree crossover randomly selects a crossover point in each parent tree. Then, it creates the offspring by replacing the sub-tree rooted at the crossover point in a copy of the first parent with a copy of the sub-tree rooted at the crossover point in the second parent, as illustrated in Fig. 6.

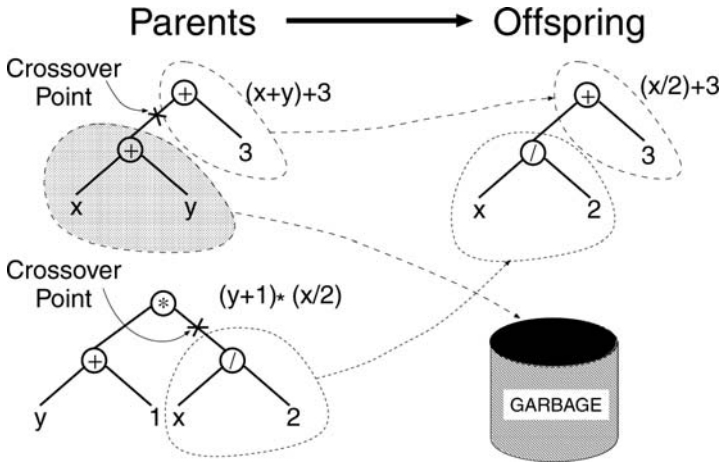


Fig. 6. Example of subtree crossover

Except in technical studies on the behavior of GP, crossover points are usually *not* selected with uniform probability. Typical GP primitive sets lead to trees with an average branching factor of at least two, so the majority of the nodes will be leaves. Consequently the uniform selection of crossover points leads to crossover operations frequently exchanging only very small amounts of genetic material (that is, small subtrees); many crossovers may in fact reduce to simply swapping two leaves. To counter this, Koza suggested the widely used approach of choosing functions 90% of the time, while leaves are selected 10% of the time.

While subtree crossover is the most common version of crossover in tree-based GP, other forms have been defined and used. For example, *one-point crossover* [222, 298, 300] works by selecting a *common* crossover point in the parent programs and then swapping the corresponding subtrees. To account for the possible structural diversity of the two parents, one-point crossover analyzes the two trees from the root nodes and considers for the selection of the crossover point only the parts of the two trees, called the *common region*, which have the same topology (that is, the same arity in the nodes encountered traversing the trees from the root node). In *context-preserving crossover* [79], the crossover points are constrained to have the same coordinates, like in one-point crossover. However, in this case no other constraint is imposed on their selection (in other words, they are not limited to the common region).

In *size-fair crossover* [205, 206] the first crossover point is selected randomly like in standard crossover. Then the size of the subtree to be excised from the first parent is calculated. This is used to constrain the choice of the second crossover point so as to guarantee that the subtree excised from the second parent will not be ‘unfairly’ big. Finally, it is worth mentioning that the notion of common region is related to the notion of homology, in the sense that the common region represents the result of a matching process between parent trees. It is then possible to imagine that within such a region transfer of homologous primitives can happen in very much like the same way as it happens in GAs operating on linear chromosomes. An example of recombination operator that implements this idea is *uniform crossover* for GP [299].

The most commonly used form of mutation in GP (which we will call *subtree mutation*) randomly selects a mutation point in a tree and substitutes the sub-tree rooted there with a randomly generated sub-tree. This is illustrated in Fig. 7. Subtree mutation is sometimes implemented as crossover between a program and a newly generated random program; this operation is also known as ‘headless chicken’ crossover [10].

Another common form of mutation is *point mutation*, which is the rough equivalent for GP of the bit-flip mutation used in GAs. In point mutation a random node is selected and the primitive stored there is replaced with a different random primitive of the same arity taken from the primitive set. If no other primitives with that arity exist, nothing happens to that node (but other

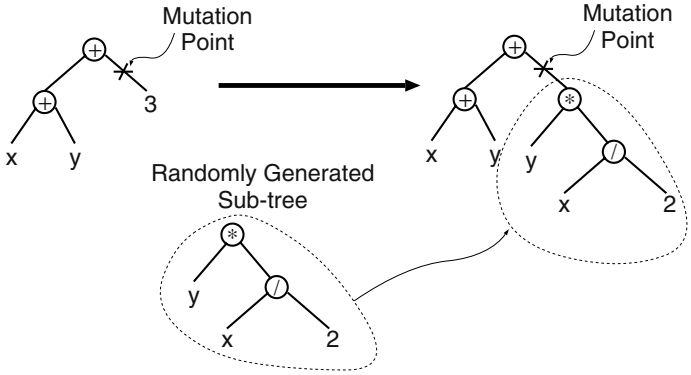


Fig. 7. Example of subtree mutation

nodes may still be mutated). Note that, when subtree mutation is applied, this involves the modification of exactly one subtree. Point mutation, on the other hand, is typically applied with a given mutation rate on a per-node basis, allowing multiple nodes to be mutated independently.

There are a number of mutation operators which treat constants in the program as special cases. [341] mutates constants by adding Gaussianly distributed random noise to them. However, others use a variety of potentially expensive optimization tools to try and fine tune an existing program by finding the 'best' value for constants within it. For instance, [340] uses 'a numerical partial gradient ascent ... to reach the nearest local optimum' to modify all constants in a program, while [348] uses simulated annealing to stochastically update numerical values within individuals.

While mutation is not necessary for GP to solve many problems, [281] argues that, in some cases, GP with mutation alone can perform as well as GP using crossover. While mutation was often used sparsely in early GP work, it is more widely used in GP today, especially in modeling applications.

3 Getting Ready to Run Genetic Programming

To run a GP system to solve a problem a small number of ingredients, often termed *preparatory steps*, need to be specified:

1. the terminal set,
2. the function set,
3. the fitness measure,
4. certain parameters for controlling the run, and
5. the termination criterion and method for designating the result of the run.

In this Section we consider these ingredients in more detail.

3.1 Step 1: Terminal Set

While it is common to describe GP as evolving *programs*, GP is not typically used to evolve programs in the familiar, Turing-complete languages humans normally use for software development. It is instead more common to evolve programs (or expressions or formulae) in a more constrained and often domain-specific language. The first two preparatory steps, the definition of the terminal and function sets, specify such a language – that is, the ingredients that are available to GP to create computer programs.

The terminal set may consist of:

- *the program's external inputs*, typically taking the form of named variables (say `x`, `y`);
- *functions with no arguments*, which are, therefore, interesting either because they return different values in different invocations (for example, the function `rand()` that returns random numbers, or a function `dist_to_wall()` that returns the distance from the robot we are controlling to an obstacle) or because they produce side effects (such as `go_left()`); and
- *constants*, which can either be pre-specified or randomly generated as part of the tree creation process.

Note that using a primitive such as `rand` can cause the behavior of an individual program to vary every time it is called, even if it is given the same inputs. What we often want instead is a set of fixed random constants that are generated as part of the process of initializing the population. This is typically accomplished by introducing a terminal that represents an *ephemeral random constant*. Every time this terminal is chosen in the construction of an initial tree (or a new subtree to use in an operation like mutation), a different random value is generated which is then used for that *particular* terminal, and which will remain fixed for the rest of the run. The use of ephemeral random constants is typically denoted by including the symbol \mathfrak{R} in the terminal set; see Sect. 4 for an example.

3.2 Step 2: Function Set

The function set used in GP is typically driven by the nature of the problem domain. In a simple numeric problem, for example, the function set may consist of merely the arithmetic functions (`+`, `-`, `*`, `/`). However, all sorts of other functions and constructs typically encountered in computer programs can be used. Table 1 shows a sample of some of the functions one sees in the GP literature. Also for many problems, the primitive set includes specialised functions and terminals which are expressly designed to solve problems in a specific domain of application. For example, if the goal is to program a robot to mop the floor, then the function set might include such actions as `move`, `turn`, and `swish-the-mop`.

Table 1. Examples of primitives allowed in the GP function and terminal sets

Function Set		Terminal Set	
<i>Kind of Primitive Example(s)</i>		<i>Kind of Primitive Example(s)</i>	
Arithmetic	<code>+, *, /</code>	Variables	<code>x, y</code>
Mathematical	<code>sin, cos, exp</code>	Constant values	<code>3, 0.45</code>
Boolean	<code>AND, OR, NOT</code>	0-arity functions	<code>rand, go_left</code>
Conditional	<code>IF-THEN-ELSE</code>		
Looping	<code>FOR, REPEAT</code>		
⋮	⋮		

Closure

For GP to work effectively, most function sets are required to have an important property known as *closure* [188], which can in turn be broken down into the properties of *type consistency* and *evaluation safety*.

Type consistency is necessitated by the fact that subtree crossover (as described in Sect. 2.4) can mix and join nodes quite arbitrarily during the evolutionary process. As a result it is necessary that *any* subtree can be used in any of the argument positions for every function in the function set, because it is always possible that sub-tree crossover will generate that combination. For functions that return a value (such as `+`, `-`, `*`, `/`), it is then common to require that all the functions be type consistent, namely that they all return values of the same type, and that all their arguments be of that type as well. In some cases this requirement can be weakened somewhat by providing an automatic conversion mechanism between types – for example, converting numbers to Booleans by treating all negative values as *false*, and non-negative values as *true*. Conversion mechanisms like this can, however, introduce unexpected biases into the search process, so they should be used thoughtfully.

The requirement of type consistency can seem quite limiting, but often simple restructuring of the functions can resolve apparent problems. An `if` function, for example, would often be defined as taking three arguments: The test, the value to return if the test evaluates to *true*, and the value to return if the test evaluates to *false*. The first of these three arguments is clearly Boolean, which would suggest that `if` can't be used with numeric functions like `+`. This can easily be worked around however by providing a mechanism to automatically convert a numeric value into a Boolean as discussed above. Alternatively, one can replace the traditional `if` with a function of four (numeric) arguments *a, b, c, d* with the semantics ‘If *a* < *b* then return value *c*, otherwise return value *d*’. These are obviously just specific examples of general techniques; the details are likely to depend on the particulars of your problem domain.

An alternative to requiring type consistency is to extend the GP system to, for example, explicitly include type information, and constrain operations like crossover so they do not perform ‘illegal’ (from the standpoint of the type system) operations. This is discussed further in Sect. 5.4.

The other component of closure is evaluation safety, necessitated by the fact that many commonly used functions can fail in various ways. An evolved expression might, for example, divide by 0, or call `MOVE_FORWARD` when facing a wall or precipice. This is typically dealt with by appropriately modifying the standard behavior of primitives. It is common, for example, to use *protected* versions of numeric functions that can throw exceptions, such as division, logarithm, and square root. The protected version of such a function first tests for potential problems with its input(s) before executing the corresponding instruction, and if a problem is spotted some pre-fixed value is returned. Protected division (often notated with `%`), for example, checks for the case that its second argument is 0, and typically returns 1 if it is (regardless of the value of the first argument).¹ Similarly, `MOVE_AHEAD` can be modified to do nothing if a forward move is illegal for some reason or, if there are no other obstacles, the edges can simply be eliminated by making the world toroidal.

An alternative to protected functions is to trap run-time exceptions and strongly reduce the fitness of programs that generate such errors. If the likelihood of generating invalid expressions is very high, however, this method can lead to all the individuals in the population having nearly the same (very poor) fitness, leaving selection with very little discriminatory power.

One type of run-time error that is somewhat more difficult to check for is numeric overflow. If the underlying implementation system throws some sort of exception, then this can be handled either by protection or by penalizing as discussed above. If, however, the implementation language quietly ignores the overflow (for instance, the common practice of wrapping around on integer overflow), and if this behavior is seen as unacceptable, then the implementation will need to include appropriate checks to catch and handle such overflows.

Sufficiency

There is one more property that, ideally, primitives sets should have: *sufficiency*. Sufficiency requires that the primitives in the primitive set are capable of expressing the solutions to the problem, in other words that the set of all the possible recursive compositions of such primitives includes at least one

¹ The decision to return 1 here provides the GP system with a simple and reliable way to generate the constant 1, via an expression of the form `(/ x x)`. This, combined with a similar mechanism for generating 0 via `(- x x)` ensures that GP can easily construct these two important constant.

solution. Unfortunately, sufficiency can be guaranteed only for some problems, when theory or experience with other methods tells us that a solution can be obtained by combining the elements of the primitive set.

As an example of a sufficient primitive set let us consider the set {AND, OR, NOT, x_1 , x_2 , ..., x_N }, which is always sufficient for Boolean function induction problems, since it can produce all Boolean functions of the variables x_1 , x_2 , ..., x_N . An example of insufficient set is the set {+, -, *, /, x , 0, 1, 2}, which is insufficient whenever, for example, the target function is transcendental – for example, $\exp(x)$ – and therefore cannot be expressed as a rational function (basically, a ratio of polynomials). When a primitive set is insufficient for a particular application, GP can only develop programs that approximate the desired one, although perhaps very closely.

Evolving Structures other than Programs

There are many problems in the real world where solutions cannot be directly cast as computer programs. For example, in many design problems the solution is an artifact of some type (a bridge, a circuit, or similar). GP has been applied to problems of this kind by using a trick: the primitive set is designed in such a way that, through their execution, programs construct solutions to the problem. This has been viewed as analogous to the development by which an egg grows into an adult. For example, if the goal is the automatic creation of an electronic controller for a plant, the function set might include common components such as **integrator**, **differentiator**, **lead**, **lag**, and **gain**, and the terminal set might contain **reference**, **signal**, and **plant output**. Each of these operations, when executed, then insert the corresponding device into the controller being built. If, on the other hand, the goal is the synthesis of analogue electrical circuits the function set might include components such as transistors, capacitors, resistors, and so on. This is further discussed in Sect. 5.5.

3.3 Step 3: Fitness Function

The first two preparatory steps define the primitive set for GP, and therefore, indirectly define the search space GP will explore. This includes all the programs that can be constructed by composing the primitives in all possible ways. However, at this stage we still do not know which elements or regions of this search space are good (that is, include programs that solve or approximately solve the problem). This is the task of the fitness measure, which effectively (albeit implicitly) specifies the desired goal of the search process. The fitness measure is our primary (and often sole) mechanism for giving a high-level statement of the problem's requirements to the GP system. For example, if the goal is to get GP to automatically synthesize an amplifier, the fitness function is the mechanism for telling GP to synthesize a circuit that

amplifies an incoming signal (as opposed to, say, a circuit that suppresses the low frequencies of an incoming signal or computes its square root).

Depending on the problem at hand, fitness can be measured in terms of the amount of *error* between its output and the desired output, the amount of *time* (fuel, money, and the like) required to bring a system to a desired *target state*, the *accuracy* of the program in recognizing patterns or classifying objects into classes, the *payoff* that a game-playing program produces, the *compliance* of a structure with user-specified design criteria, and so on.

There is something unusual about the fitness functions used in GP that differentiates them from those used in most other EAs. Because the structures being evolved in GP are computer programs, fitness evaluation normally requires executing all the programs in the population, typically multiple times. While one can compile the GP programs that make up the population, the overhead is usually substantial, so it is much more common to use an interpreter to evaluate the evolved programs.

Interpreting a program tree means executing the nodes in the tree in an order that guarantees that nodes are not executed before the value of their arguments (if any) is known. This is usually done by traversing the tree recursively starting from the root node, and postponing the evaluation of each node until the value of its children (arguments) is known. This process is illustrated in Fig. 8, where the number to the right of each internal node represents the result of evaluating the subtree root at that node. In this example, the independent variable *X* evaluates to -1 . Algorithm 3 gives a pseudo-code implementation of the interpretation procedure. The code assumes that programs are represented as prefix-notation expressions and that such expressions can be treated as lists of components.

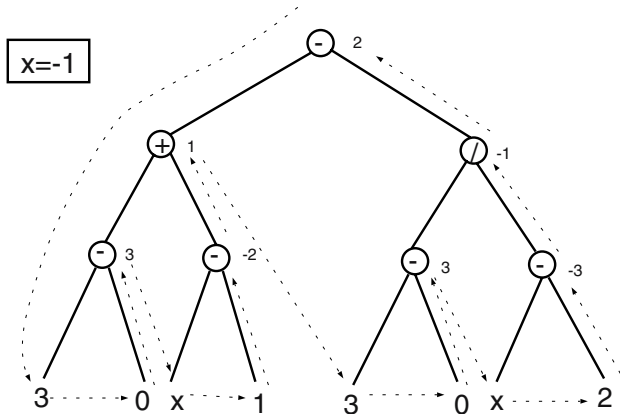


Fig. 8. Example interpretation of a syntax tree (the terminal *x* is a variable has a value of -1). The number to the right of each internal node represents the result of evaluating the subtree root at that node

Algorithm 3 Typical interpreter for GP

```

procedure: eval( expr )
1: if expr is a list then
2:   proc = expr(1) {Non-terminal: extract root}
3:   if proc is a function then
4:     value = proc( eval(expr(2)), eval(expr(3)), ... ) {Function: evaluate
       arguments}
5:   else
6:     value = proc( expr(2), expr(3), ... ) {Macro: don't evaluate arguments}
7:   else
8:     if expr is a variable or expr is a constant then
9:       value = expr {Terminal variable or constant: just read the value}
10:    else
11:      value = expr() {Terminal 0-arity function: execute}
12: return value

```

Notes: **expr** is an expression in prefix notation, **expr(1)** represents the primitive at the root of the expression, **expr(2)** represents the first argument of that primitive, **expr(3)** represents the second argument, and so forth.

In some problems we are interested in the *output* produced by a program, namely the value returned when we evaluate starting at the root node. In other problems, however, we are interested in the actions performed by a program. In this case the primitive set will include functions with side effects – that is, functions that do more than just return a value – but say change some global data structures, print or draw something on the screen or control the motors of a robot. Irrespective of whether we are interested in program outputs or side effects, quite often the fitness of a program depends on the results produced by its execution on many different inputs or under a variety of different conditions. These different test cases typically incrementally contribute to the fitness value of a program, and for this reason are called *fitness cases*.

Another common feature of GP fitness measures is that, for many practical problems, they are *multi-objective*, in other words they combine two or more different elements that are often in competition with one another. The area of multi-objective optimization is a complex and active area of research in GP and machine learning in general; see [73], for example, for more.

3.4 Steps 4 and 5: Parameters and Termination

The fourth and fifth preparatory steps are administrative. The fourth preparatory step entails specifying the control parameters for the run. The most important control parameter is the population size. Other control parameters include the probabilities of performing the genetic operations, the maximum size for programs, and other details of the run.

The fifth preparatory step consists of specifying the termination criterion and the method of designating the result of the run. The termination criterion may include a maximum number of generations to be run as well as a problem-specific success predicate. Typically the single best-so-far individual is then harvested and designated as the result of the run, although one might wish to return additional individuals and data as necessary or appropriate for your problem domain.

4 Example Genetic Programming Run

This Section provides a concrete, illustrative run of GP in which the goal is to automatically evolve an expression whose values match those of the quadratic polynomial $x^2 + x + 1$ in the range $[-1, +1]$. That is, the goal is to automatically create a computer program that matches certain numerical data. This process is sometimes called *system identification* or *symbolic regression* (see Sect. 7.1 for more).

We begin with the five preparatory steps from the previous section, and then describe in detail the events in one possible run.

4.1 Preparatory Steps

The purpose of the first two preparatory steps is to specify the ingredients the evolutionary process can use to construct potential solutions. Because the problem is to find a mathematical function of one independent variable, x , the terminal set (the inputs to the to-be-evolved programs) must include this variable. The terminal set also includes ephemeral random constants, drawn from some reasonable range, say from -5.0 to $+5.0$, as described in Sect. 3.1. Thus the terminal set, T , is

$$T = \{x, \mathfrak{R}\} \quad (1)$$

The statement of the problem is somewhat flexible in that it does not specify what functions may be employed in the to-be-evolved program. One simple choice for the function set consists of the four ordinary arithmetic functions: addition, subtraction, multiplication, and division. Most numeric regression will include at least these operations, often in conjunction with additional functions such as sin and log. In our example, however, we will restrict ourselves to the simple function set

$$F = \{+, -, *, \%\} \quad (2)$$

where $\%$ is protected division as discussed in Sect. 3.2.

The third preparatory step involves constructing the fitness measure that specifies what the human wants. The high-level goal of this problem is to

find a program whose output is equal to the values of the quadratic polynomial $x^2 + x + 1$. Therefore, the fitness assigned to a particular individual in the population for this problem must reflect how closely the output of an individual program comes to the target polynomial $x^2 + x + 1$.

The fitness measure *could* be defined as the integral of the absolute value of the differences (errors) between the individual mathematical expression and the target quadratic polynomial $x^2 + x + 1$, taken over the range $[-1, +1]$. However, for most symbolic regression problems, it is not practical or possible to analytically compute the value of the integral of the absolute error. Thus it is common to instead define the fitness to be the *sum of absolute errors* measured at different values of the independent variable x in the range $[-1.0, +1.0]$. In particular, we will measure the errors for $x = -1.0, -0.9, \dots, 0.9, 1.0$. A smaller value of fitness (error) is better; a fitness (error) of zero would indicate a perfect fit. Note that with this definition, our fitness is (approximately) proportional to the area between the parabola $x^2 + x + 1$ and the curve representing the candidate individual (see Fig. 10 for examples).

The fourth step is where we set our run parameters. The population size in this small illustrative example will be just four. In actual practice, the population size for a run of GP typically consists of thousands or millions of individuals, but we will use this tiny population size to keep the example manageable. In practice, the crossover operation is commonly used to generate about 90% of the individuals in the population; the reproduction operation (where a fit individual is simply copied from one generation to the next) is used to generate about 8% of the population; the mutation operation is used to generate about 1% of the population; and the architecture-altering operations (see Sect. 5.2) are used to generate perhaps 1% of the population. Because this example involves an abnormally small population of only four individuals, the crossover operation will be used to generate two individuals, and the mutation and reproduction operations will each be used to generate one individual. For simplicity, the architecture-altering operations are not used for this problem.

In the fifth and final step we need to specify a termination condition. A reasonable termination criterion for this problem is that the run will continue from generation to generation until the fitness (or error) of some individual is less than 0.1. In this contrived example, our example run will (atypically) yield an algebraically perfect solution (with a fitness of zero) after merely one generation.

4.2 Step-by-Step Sample Run

Now that we have performed the five preparatory steps, the run of GP can be launched.

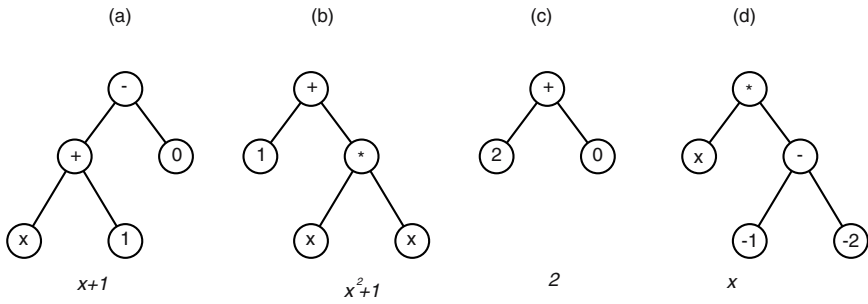


Fig. 9. Initial population of four randomly created individuals of generation 0

Initialization

GP starts by randomly creating a population of four individual computer programs. The four programs are shown in Fig. 9 in the form of trees.

The first randomly constructed program tree (Fig. 9a), and is equivalent to the expression $x + 1$. The second program (Fig. 9b) adds the constant terminal 1 to the result of multiplying x by x and is equivalent to $x^2 + 1$. The third program (Fig. 9c) adds the constant terminal 2 to the constant terminal 0 and is equivalent to the constant value 2. The fourth program (Fig. 9d) is equivalent to x .

Fitness Evaluation

Randomly created computer programs will, of course, typically be very poor at solving the problem at hand. However, even in a population of randomly created programs, some programs are better than others. Here, for example, the four random individuals from generation 0 in Fig. 9 produce outputs that deviate by different amounts from the target function $x^2 + x + 1$. Fig. 10 compares the plots of each of the four individuals in Fig. 9 and the target quadratic function $x^2 + x + 1$. The sum of absolute errors for the straight line $x + 1$ (the first individual) is 7.7 (Fig. 10a). The sum of absolute errors for the parabola $x^2 + 1$ (the second individual) is 11.0 (Fig. 10b). The sums of the absolute errors for the remaining two individuals are 17.98 (Fig. 10c) and 28.7 (Fig. 10d), respectively.

As can be seen in Fig. 10, the straight line $x + 1$ (Fig. 10a) is closer to the parabola $x^2 + x + 1$ in the range from -1 to $+1$ than any of three other programs in the population. This straight line is, of course, not equivalent to the parabola $x^2 + x + 1$; it is not even a quadratic function. It is merely the best candidate that happened to emerge from the blind (and very limited) random search of generation 0. In the valley of the blind, the one-eyed man is king.

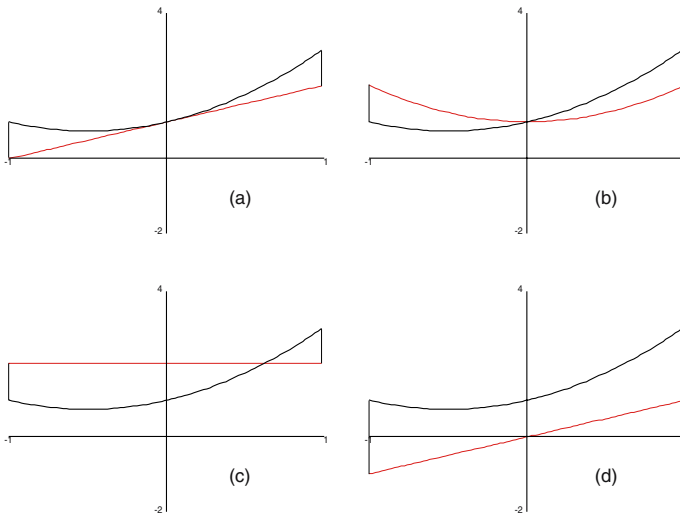


Fig. 10. Graphs of the evolved functions from generation 0. The heavy line in each plot is the target function $x^2 + x + 1$, with the other line being the evolved functions from the first generation (see Fig. 9). The fitness of each of the four randomly created individuals of generation 0 is approximately proportional to the area between two curves, with the actual fitness values being 7.7, 11.0, 17.98 and 28.7 for individuals (a) through (d), respectively

Selection, Crossover and Mutation

After the fitness of each individual in the population is ascertained, GP then probabilistically selects relatively more fit programs from the population to act as the parents of the next generation. The genetic operations are applied to the selected individuals to create offspring programs. The important point for our example is that our selection process is not greedy. Individuals that are known to be inferior will be selected to a certain degree. The best individual in the population is not guaranteed to be selected and the worst individual in the population will not necessarily be excluded.

In this example, we will start with the reproduction operation. Because the first individual (Fig. 9a) is the most fit individual in the population, it is very likely to be selected to participate in a genetic operation. Let us suppose that this particular individual is, in fact, selected for reproduction. If so, it is copied, without alteration, into the next generation (generation 1). It is shown in Fig. 11a as part of the population of the new generation.

We next perform the mutation operation. Because selection is probabilistic, it is possible that the third best individual in the population (Fig. 9c) is selected. One of the three nodes of this individual is then randomly picked as the site for the mutation. In this example, the constant terminal 2 is picked

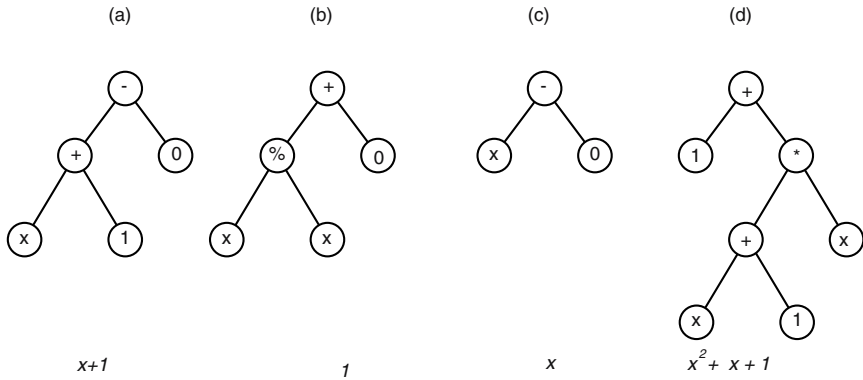


Fig. 11. Population of generation 1 (after one reproduction, one mutation, and one two-offspring crossover operation)

as the mutation site. This program is then randomly mutated by deleting the entire subtree rooted at the picked point (in this case, just the constant terminal 2) and inserting a subtree that is randomly constructed in the same way that the individuals of the initial random population were originally created. In this particular instance, the randomly grown subtree computes the quotient of x and x using the protected division operation $\%$. The resulting individual is shown in Fig. 11b. This particular mutation changes the original individual from one having a constant value of 2 into one having a constant value of 1, improving its fitness from 17.98 to 11.0.

Finally, we use the crossover operation to generate our final two individuals for the next generation. Because the first and second individuals in generation 0 are both relatively fit, they are likely to be selected to participate in crossover. However, selection can always pick suboptimal individuals. So, let us assume that in our first application of crossover the pair of selected parents is composed of the above-average tree in Figs. 9a and the below-average tree in Fig. 9d. One point of the first parent, namely the $+$ function in Fig. 9a, is randomly picked as the crossover point for the first parent. One point of the second parent, namely the leftmost terminal x in Fig. 9d, is randomly picked as the crossover point for the second parent. The crossover operation is then performed on the two parents. The offspring (Fig. 11c) is equivalent to x and is not particularly noteworthy.

Let us now assume, that in our second application of crossover, selection chooses the two most fit individuals as parents: the individual in Fig. 9b as the first parent, and the individual in Fig. 9a as the second. Let us further imagine that crossover picks the leftmost terminal x in Fig. 9b as a crossover point for the first parent, and the $+$ function in Fig. 9a as the crossover point for the second parent. Now the offspring (Fig. 11d) is equivalent to $x^2 + x + 1$ and has a fitness (sum of absolute errors) of zero. Because the fitness of this

individual is below 0.1, the termination criterion for the run is satisfied and the run is automatically terminated. This best-so-far individual (Fig. 11d) is then designated as the result of the run.

Note that the best-of-run individual (Fig. 11d) incorporates a good trait (the quadratic term x^2) from the first parent (Fig. 9b) with two other good traits (the linear term x and constant term of 1) from the second parent (Fig. 9a). The crossover operation thus produced a solution to this problem by recombining good traits from these two relatively fit parents into a superior (indeed, perfect) offspring.

This is, obviously, a highly simplified example, and the dynamics of a real GP run are typically far more complex than what is presented here. Also, in general there is no guarantee that an exact solution like this will be found by GP.

5 Advanced Tree-Based GP Techniques

5.1 Automatically Defined Functions

Human programmers organize sequences of repeated steps into reusable components such as subroutines, functions, and classes. They then repeatedly invoke these components — typically with different inputs. Reuse eliminates the need to ‘reinvent the wheel’ every time a particular sequence of steps is needed. Reuse makes it possible to exploit a problem’s modularities, symmetries, and regularities (and thereby potentially accelerate the problem-solving process). This can be taken further, as programmers typically organize these components into hierarchies in which top level components call lower level ones, which call still lower levels, and so forth.

While several different mechanisms for evolving reusable components have been proposed (for instance, [13, 331]), Koza’s *Automatically Defined Functions* (ADFs) [189] have been the most successful way of evolving reusable components.

When ADFs are used, a program consists of one (or more) function-defining trees (that is, ADFs) as well as one or more main result-producing trees (see Fig. 3). An ADF may have none, one, or more inputs. The body of an ADF contains its work-performing steps. Each ADF belongs to a particular program in the population. An ADF may be called by the program’s main result-producing tree, by another ADF, or by another type of tree (such as the other types of automatically evolved program components described below). Recursion is sometimes allowed. Typically, the ADFs are called with different inputs. The work-performing steps of the program’s main result-producing tree and the work-performing steps of each ADF are automatically and simultaneously created by GP. The program’s main result-producing tree and its

ADFs typically have different function and terminal sets. ADFs are the focus of [189] and [190].

Koza also proposed other types of automatically evolved program components. Automatically defined iterations (ADIs), automatically defined loops (ADLs) and automatically defined recursions (ADRs) provide means (in addition to ADFs) to reuse code. Automatically defined stores (ADSs) provide means to reuse the result of executing code. These automatically defined components are described in [195].

5.2 Program Architecture and Architecture-Altering Operations

The architecture of a program consists of the total number of trees, the type of each tree (for example, result-producing tree, ADF, ADI, ADL, ADR, or ADS), the number of arguments (if any) possessed by each tree, and, finally, if there is more than one tree, the nature of the hierarchical references (if any) allowed among the tree.

There are three ways to determine the architecture of the computer programs that will be evolved:

1. The human user may specify in advance the architecture of the overall program, in other words perform an architecture-defining preparatory step in addition to the five itemized in Sect. 2.
2. The run may employ evolutionary selection of the architecture (as described in [189]), thereby enabling the architecture of the overall program to emerge from a competitive process during the run of GP.
3. The run may employ a set of *architecture-altering operations* which can create new ADFs, remove ADFs, and increase or decrease the number of inputs an ADF has. Note initially, many architecture changes (such as those define in [189]) are designed not to change the semantics of the program and, so, the altered program often has exactly the same fitness as its parent. However, the new arrangement of ADFs may make it easier for subsequent changes to evolve better programs later.

5.3 Genetic Programming Problem Solver

The Genetic Programming Problem Solver (GPPS) is described in part 4 of [195]. It is a very powerful AI approach, but typically it requires considerable computational time.

When GPPS is used, the user does not need to chose either the terminal set or the function set (the first and second preparatory steps – Sect. 2). The function set for GPPS is the four basic arithmetic functions (addition, subtraction, multiplication, and division) and a conditional operator IF. The terminal set for GPPS consists of numerical constants and a set of input terminals that are presented in the form of a vector. By employing this generic

function set and terminal set, GPPS reduces the number of preparatory steps from five to three.

GPPS relies on the architecture-altering operations described in Sect. 5.2 to dynamically create, duplicate, and delete subroutines and loops during the run of GP. Additionally, in version 2.0 of GPPS [195, Chapter 22], the architecture-altering operations are used to dynamically create, duplicate, and delete recursions and internal storage. Because the architecture of the evolving program is automatically determined during the run, GPPS eliminates the need for the user to specify in advance whether to employ subroutines, loops, recursions and internal storage in solving a given problem. It similarly eliminates the need for the user to specify the number of arguments possessed by each subroutine. Further, GPPS eliminates the need for the user to specify the hierarchical arrangement of the invocations of the subroutines, loops, and recursions.

5.4 Constraining Syntactic Structures

As discussed in Sect. 3, most GP systems require type consistency where all sub-trees return data of the same type, ensuring that the output of any subtree can be used as one of the inputs to any other node. This ensures that the shuffling caused by sub-tree crossover, and so on, doesn't lead to incompatible connections between nodes. Many problem domains, however, have multiple types and do not naturally satisfy the type consistency requirement. This can often be addressed through creative definitions of functions and implicit type conversion, but this may not always be desirable. For example, if a key goal is that the evolved solutions should be easily understood or analyzed, then removing type concepts and other common constraints may lead to solutions that are unacceptable because they are quite difficult to interpret. GP systems that are constrained structurally or via a type system often generate results that are easier for humans to understand and analyze [137], [203, p. 126].

In this Section we will look at three different approaches to constraining the syntax of the evolved expression trees in GP: simple structure enforcement, strongly typed GP and grammar-based constraints.

Enforcing Particular Structures

If a particular structure is believed or known to be important then one can modify the GP system to require all individuals to have that structure [188]. A periodic function, for example, might be believed to be of the form $a \sin(bt)$ and so the GP is restricted to evolving expressions having that structure. (in other words, a and b are allowed to evolve freely, but the rest of the structure is fixed). Syntax restriction can also be used to make GP follow sensible engineering practices. For example, we might want to ensure that loop control variables appear in the correct parts of FOR loops and nowhere else [203, p.126].

This can be implemented in a number of ways. One could, for example, ensure that all the initial individuals have that structure (for example, generating random sub-trees for a and b while fixing the rest), and then constrain operations like crossover so that they do not alter any of the fixed regions. An alternative approach would be to evolve the various (sub)components separately in any of several ways. One could, for example, evolve pairs of trees (a, b), or one could have two separate populations, one of which is being used to evolve candidates for a while the other is evolving candidates for b .

Strongly Typed GP

Since constraints are often driven by or expressed using a type system, a natural approach is to incorporate types and their constraints into the GP system [259]. In strongly typed GP, every terminal has a type, and every function has types for each of its arguments and a type for its return value. The process that generates the initial, random expressions, and all the genetic operators are then constrained to not violate the type system's constraints.

Returning to the `if` example from Sect. 3, we might have a domain with both numeric and Boolean terminals (for instance, `get_speed` and `is_food_ahead`). We might then have an `if` function that takes three arguments: A test (Boolean), the value to return if the test is *true*, and the value to return if the test is *false*. Assuming that the second and third values are constrained to be numeric, then the output of the `if` is also going to be numeric. If we choose the test argument as a root parent crossover point, then the sub-tree to insert must have a Boolean output; if we choose either the second or third argument as a root parent crossover point, then the inserted sub-tree must be numeric.

This basic approach to types can be extended to more complex type systems including simple generics [259], multi-level type systems [138], and fully polymorphic, higher-order type systems with generics [413].

Grammar-Based Constraints

Another natural way to express constraints is via grammars, and these have been used in GP in a variety of ways [123, 143, 279, 395, 404]. Many of these simply use a grammar as a means of expressing the kinds of constraints discussed above in Sect. 5.4. One could enforce the structure for the period function using a grammar such as the following:

$$\begin{aligned}
 \textit{tree} &::= E \times \sin(E \times t) \\
 E &::= \textit{var} \mid E \textit{ op } E \\
 \textit{op} &::= + \mid - \mid \times \mid \div \\
 \textit{var} &::= x \mid y \mid z
 \end{aligned} \tag{3}$$

Genetic operators are restricted to only swapping sub-trees deriving from a common non-terminal symbol in the grammar. So, for example, an *E* could be replaced by another *E*, but an *E* could not be replaced by an *op*. This can be extended to, for example, context-sensitive grammars by incorporating various related concepts from computational linguistics. The TAG3P+ system [143, 144], for example, uses tree-adjointing grammars (TAGs) to constrain the evolved structures.

Another major area is grammatical evolution (GE) [279, 336]. In GE a grammar is used as in the example above. However instead of representing individuals directly using either expression or derivation trees, grammatical evolution represents individuals using a variable length sequence of integers. For each production rule, the set of options on the right hand side are numbered from 0 upwards. In the example above the first rule only has one option on the right hand side; this would both be numbered 0. The second rule has two options, which would be numbered 0 and 1, the third rule has four options which would be numbered 0 to 3, and the fourth rule has three options numbered 0 to 2. An expression tree is then generated by using the values in the individual to ‘choose’ which option to take in the production rules, rewriting the left-most non-terminal is the current expression.

If, for example, an individual is represented by the sequence:

$$39, 7, 2, 83, 66, 92, 57, 80, 47, 94$$

then the translation process would proceed as follows (with the non-terminal to be rewritten underlined in each case):

$$\begin{aligned}
 & \underline{tree} \\
 \rightarrow & \langle 39 \bmod 1 = 0, \text{that is, there is only one option} \rangle \\
 & \underline{E} \times \sin(E \times t) \\
 \rightarrow & \langle 7 \bmod 2 = 1, \text{in other words, choose second option} \rangle \\
 & (\underline{E} \text{ op } E) \times \sin(E \times t) \\
 \rightarrow & \langle 2 \bmod 2 = 0, \text{namely, take the first option} \rangle \\
 & (\underline{\text{const}} \text{ op } E) \times \sin(E \times t) \\
 \rightarrow & \langle 83 \bmod 3 = 2, \text{again, only one option, generate an ephemeral constant} \rangle \\
 & (z \text{ op } \underline{E}) \times \sin(E \times t) \\
 \rightarrow & \langle 66 \bmod 4 = 2, \text{take the third option} \rangle \\
 & (z \times \underline{E}) \times \sin(E \times t) \\
 \dots & \\
 & (z \times x) \times \sin(z \times t)
 \end{aligned}$$

In this example we didn’t need to use all the numbers in the sequence to generate a complete expression free of non-terminals; 94 was in fact never used. In general ‘extra’ genetic material is simply ignored. Alternatively, sometimes a sequence can be ‘too short’ in the sense that the end of the sequence

is reached before the translation process is complete. There are a variety of options in this case, including failure (assigning this individual the worst possible fitness) and wrapping (continuing the translation process, moving back to the front of the numeric sequence). See [279] for further details on this and other aspects of grammatical evolution.

Constraints and Bias

While increasing the expressive power of a type system or other constraint mechanism may indeed limit the search space by restricting the kinds of structures that can be constructed, this often comes at a price. An expressive type system typically requires more complex machinery to support it. It often makes it more difficult to generate type-correct individuals in the initial population, and more difficult to find genetic operations that do not violate the type system. In an extreme case like constructive type theory, the type system is so powerful that it can completely express the formal specification of the program, so any program/expression having this type is guaranteed to meet that specification. In the GP context this would mean that all the members of the initial population (assuming that they are required to have the desired type) would in fact be solutions to the problem, thus removing the need for any evolution at all! Even without such extreme constraints, it has often been found necessary to develop additional machinery in order to efficiently generate an initial population that satisfies the necessary constraints [259, 318, 339, 413].

As a rule, systems that focus on *syntactic* constraints (such as grammar based systems) require less machinery than those that focus on *semantic* constraints (such as type systems), since it's typically easier to satisfy the syntactic constraints in a mechanistic fashion. Grammar based systems such as GE and TAG, for example, are typically simple to initialize, and require few if any constraints to be honored by the mutation and recombination operators. The work (and the bias) in these systems is much more in the design of the grammar; once that is in place there is often little additional work required of either the practitioner or the GP system to enforce the constraints implied by the grammar.

While a constraint system may limit the search space in valuable ways [318] and can improve performance on interesting problems [144], there is no general guarantee that constraint systems will make the evolutionary search process easier. There is no broad assurance, for example, that constraint systems will significantly increase the density of solutions or (perhaps more importantly) approximate solutions. While there are cases where constraint systems smooth the search landscape [144], it is also possible for constraint systems to make the search landscape more rugged by preventing genetic operations from creating intermediate forms on potentially valuable evolutionary paths. It might be useful to extend solution density studies such as those summarised in [222] to

the landscapes generated by constraint systems in order to better understand the impact of these constraints on the underlying search spaces.

In summary, while types, grammars, and other constraint systems can be powerful tools, all such systems carry biases. One, therefore, needs to be careful to explore the biases introduced by the constraints and not simply assume that they are beneficial to the search process.

5.5 Developmental Genetic Programming

When appropriate terminals, functions and/or interpreters are defined, standard GP can go beyond the production of computer programs. For example, in a technique called *cellular encoding*, programs are interpreted as sequences of instructions which modify (grow) a simple initial structure (embryo). Once the program terminates, the quality of the resulting structure is taken to be the fitness of the program. Naturally, the primitives of the language must be appropriate to grow structures in the domain of interest. Typical instructions involve the insertion and/or sizing of components, topological modifications of the structure, etc. Cellular encoding GP has successfully been used to evolve neural networks [121, 122, 124] and electronic circuits [193–195], as well as in numerous other domains.

One of the advantages of indirect representations such as cellular encoding is that the standard GP operators can be used to manipulate structures (such as circuits) which may have nothing in common with standard GP trees. A disadvantage is that they require an additional genotype-to-phenotype decoding step. However, when the fitness function involves complex calculations with many fitness cases the relative cost of the decoding step is often small.

5.6 Strongly Typed Autoconstructive GP – PushGP

In some ways Spector’s PushGP [183, 328, 357, 364] is also a move away from constraining evolution. Push is a strongly typed tree based language which does not enforce syntactic constraints. Essentially PushGP uses evolution (i.e. genetic programming) to automatically create programs written in the Push programming language. Each of Push’s types has its own stack. In addition to stacks for integers, floats, Booleans and so on, there is a stack for objects of type program. Using this code stack, Push naturally supports both recursion and program modules (see Sect. 5.1) without human pre-specification. The code stack allows an evolved program to push itself or fragments of itself onto the stack for subsequent manipulation.

Somewhat like ‘core wars’, PushGP can use the code stack and other operations to allow programs to construct their own crossover and other genetic operations and create their own offspring. (Spector prevents programs from simply duplicating themselves to prevent catastrophic loss of population diversity.)

6 Linear and Graph-Based GP

Until now we have been talking about the evolution of programs expressed as one or more trees which are evaluated by a suitable interpreter. This is the original and most widespread type of GP, but there are other types of GP where programs are represented in different ways. This Section will look at linear programs and graph-like (parallel) programs.

6.1 Linear Genetic Programming

There are two different reasons for trying linear GP. Basic computer architectures are fundamentally the same now as they were twenty years ago, when GP began. Almost all architectures represent computer programs in a linear fashion (albeit with control structures, jumps and loops). So, why not evolve linear programs [24, 280, 288]. Also, computers do not naturally run tree-shaped programs. So, slow interpreters have to be used as part of tree-based GP. On the contrary, by evolving the binary bit patterns actually obeyed by the computer, the use of an expensive interpreter (or compiler) is avoided and GP can run several orders of magnitude faster [65, 88, 272, 275].

The typical crossover and mutation operators for linear GP ignore the details of the machine code of the computer being used. For example, crossover typically chooses randomly two crossover points in each parent and swaps the code lying between them. Since the crossed over fragments are typically of different lengths, such a crossover may change the programs' lengths (see Fig. 12). Since computer machine code is organised into 32- or 64-bit words, the crossover points occur only at the boundaries between words. Therefore, a whole number of words, containing a whole number of instructions are typically swapped over. Similarly, mutation operations normally respect word boundaries and generate legal machine code. However, linear GP lends itself to a variety of other genetic operations. For example, Fig. 13 shows homologous crossover. Many other crossover and mutation operations are possible [215].

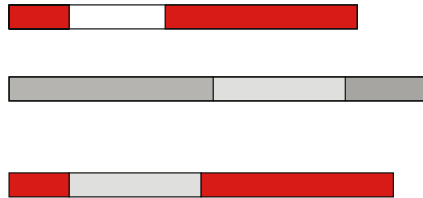


Fig. 12. Typical linear GP crossover. Two instructions are randomly chosen in each parent (*top two genomes*) as cut points. If the code fragment excised from the first parent is replaced with the code fragment excised from the second to give the child (*lower chromosome*)

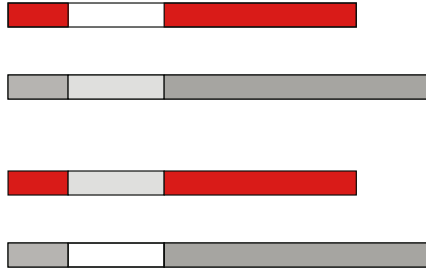


Fig. 13. Discipulus’ ‘homologous’ crossover [99,101,275]. Two parents (*top two programs*) crossover to yield two child programs (*bottom*). The two crossover cut points are the same in both parents. Note code does not change its position relative to the start of the program (*left edge*) and the child programs are the same lengths as their parents. Homologous crossover is often combined with a small amount of normal two point crossover (Fig. 12) to introduce length changes into the GP population

If the goal is execution speed, then the evolved code should be machine code for a real computer rather than some higher level language or virtual-machine code. For example, [272] started by evolving machine code for SUN computers; [65] targeted the Z80. The linear GP of [226] was firmly targeted at novel hardware but much of the GP development had to be run in simulation whilst the hardware itself was under development.

The Sun SPARC has a simple 32-bit RISC architecture which eases designing genetic operation which manipulate its machine code. Nordin wrapped each machine code GP individual inside a C function [273]. Each of the GP program’s inputs were copied from one of the C function’s arguments into one of the machine registers. Note that typically there are only a small number of inputs. Linear GP should be set up to write-protect these registers, so that inputs cannot be overwritten, since if an input is overwritten and its value is lost, the evolved code cannot be a function of it. As well as the registers used for inputs, a small number (say 2–4) of other registers are used for scratch memory to store partial results of intermediate calculations. Finally, the GP simply leaves its answer in one of the registers. The external framework uses this as the C function’s *return* value.

Note that execution speed is not the only reason for using linear GP. Linear programs can be interpreted, just as trees can be. Indeed a linear interpreter can be readily implemented. A simple linear structure lends itself to rapid analysis, which can be used for ‘dead code’ removal [33]. In some ways the search space of linear GP is easier to analyse than that of trees [204,207–209,215]. For example, we have used the T7 and T8 architectures (in simulation) for several large scale experimental and mathematical analysis of Turing complete GP [214,220,223,224]. For these reasons, it makes sense to consider linear ‘machine’ code GP, for example, in Java. Since Java is usually

Output R0..R7	Arg 1 R0..R7	Opcode + - * /	Arg 2 0...127 or R0..R7
------------------	-----------------	-------------------	----------------------------------

Fig. 14. Format of a linear GP engine instruction. To avoid the overhead of packing and unpacking data in the interpreter (written in a high level language such as C++), virtual machine instructions, unlike real machine instructions, are not packed into bit fields. In linear GP, instructions are laid from the start of the program to its end. In machine code GP, these are real machine code instructions. In interpreted linear GP, machine code is replaced with virtual machine code

run on a virtual machine, almost by definition this requires a virtual machine (like [226]) to interpret the evolved byte code [133, 240].

Since Unix was ported onto the x86, Intel's complex instruction set has had almost complete dominance. Seeing this, Nordin ported his Sun RISC linear GP onto Intel's CISC. Various changes were made to the genetic operations which ensure that the initial random programs are made only of legal Intel machine code and that mutation operations, which act inside the x86's 32-bit word, respect the x86's complex sub-fields. Since the x86 has instructions of different lengths, special care was taken when altering them. Typically several short instructions are packed into the 4-byte words. If there are any bytes left over, they are filled with no-operation codes. In this way best use is made of the available space, without instructions crossing 32-bit boundaries. Nordin's work led to *Discipulus* [99], which has been used from applications ranging from Bioinformatics [390] to robotics [219] and bomb disposal [78]. Generally, in linear GP instructions take the form shown in Fig. 14.

6.2 Graph-Based Genetic Programming

Trees are special types of graphs. So, it is natural to ask what would happen if one extended GP so as to be able to evolve graph-like programs. Starting from the mid 1990s researchers have proposed several extensions of GP that do just that, albeit in different ways.

For example, Poli proposed Parallel Distributed GP (PDGP) [290, 292]. PDGP is a form of GP which is suitable for the evolution of efficient highly parallel programs which effectively reuse partial results. Programs are represented in PDGP as graphs with nodes representing functions and terminals. Edges represent the flow of control and results. In the simplest form of PDGP edges are directed and unlabeled, in which case PDGP can be considered a generalization of standard GP. However, more complex representations can be used, which allow the exploration of a large space of possible programs including standard tree-like programs, logic networks, neural networks, recurrent transition networks and finite state automata. In PDGP, programs are

manipulated by special crossover and mutation operators which guarantee the syntactic correctness of the offspring. For this reason PDGP search is very efficient. PDGP programs can be executed in different ways, depending on whether nodes with side effects are used or not.

In a system called PADO (Parallel Algorithm Discovery and Orchestration), Teller and Veloso used a combination of GP and linear discrimination to obtain parallel classification programs for signals and images [375]. The programs in PADO are represented as graphs, although their semantics and execution strategy are very different from those of PDGP.

In Miller's Cartesian GP [256, 257], programs are represented by linear chromosomes containing integers. These are divided into groups of three or four. Each group is associated to a position in a 2-D array. An element of the group prescribes which primitive is stored at that location in the array, while the remaining elements indicate from which other locations the inputs for that primitive should be read. So, the chromosome represents a graph-like program, which is very similar to PDGP. The main difference between the two systems is that Cartesian GP operators (mainly mutation) act at the level of the linear chromosome, while in PDGP they act directly on the graph.

It is also possible to use non-graph-based GP to evolve parallel programs. For example, Bennett used a parallel virtual machine in which several standard tree-like programs (called 'agents') would have their nodes executed in parallel with a two stage mechanism simulating parallelism of sensing actions and simple conflict resolution (prioritization) for actions with side effects [28]. [8] used GP to discover rules for cellular automata, a highly parallel computational architecture, which could solve large majority classification problems. In conjunction with an interpreter implementing a parallel virtual machine, GP can also be used to translate sequential programs into parallel ones [392], or to develop parallel programs.

7 Applications

Since its early beginnings, GP has produced a cornucopia of results. The literature, which covers more than 5000 recorded uses of GP, reports an enormous number of applications where GP has been successfully used as an automatic programming tool, a machine learner or an automatic problem-solving machine. It is impossible to list all such applications here. In the following Sections we mention a representative subset for each of the main application areas of GP (Sects. 7.1–7.10), devoting particular attention to the important areas of symbolic regression (Sect. 7.1) and human-competitive results (Sect. 7.2). We conclude the section with guidelines for the choice of application areas (Sect. 7.11).

7.1 Curve Fitting, Data Modeling, and Symbolic Regression

In principle, the possible applications of GP are as many as the applications for programs (virtually infinite). However, before one can try to solve a new problem with GP, one needs to define an appropriate fitness function. In problems where only the *side effects* of the program are of interest, the fitness function usually compares the effects of the execution of a program in some suitable environments with a desired behavior, often in a very application-dependent manner. In many problems, however, the goal is *finding a function* whose output has some desired property – for example, it matches some target values (as in the example given in Sect. 4), or it is optimum against some other criteria. This type of problem is generally known as a *symbolic regression problem*.

Many people are familiar with the notion of *regression*, which is a technique used to interpret experimental data. It consists in finding the *coefficients* of a *predefined function* such that the function best fits the data. A problem with regression analysis is that, if the fit is not good, the experimenter has to keep trying different functions until a good model for the data is found. Also, in many domains the strong tradition of only using linear or quadratic models, even though it is possible that the data would be better fit by some other model. The problem of *symbolic regression*, instead, consists in finding a *general function* (with its coefficients) that fits the given data points. Since GP does not assume *a priori* a particular structure for the resulting function, it is well suited to this sort of discovery task. Symbolic regression was one of the earliest applications of GP [188], and continues to be a widely studied domain [45, 126, 167, 227].

The steps necessary to solve symbolic regression problems include the five preparatory steps mentioned in Sect. 2. However, while in the example in Sect. 4 the data points were computed using a simple formula, in most realistic situations the collection of an appropriate set of data points is an important and sometimes complex task. Often, for example, each point represents the (measured) values taken by some variables at a certain time in some dynamic process or in a certain repetition of an experiment.

Consider, for example, the case of using GP to evolve a *soft sensor* [161]. The intent is to evolve a function that will provide a reasonable estimate of what a sensor (in, say, a production facility) *would* report, based on data from other actual sensors in the system. This is typically done in cases where placing an actual sensor in that location would be difficult or expensive for some reason. It is necessary, however, to place at least one instance of such a sensor in a working system in order to collect the data needed to train and test the GP system. Once such a sensor is placed, one would collect the values reported by that sensor, and by all the other hard sensors that are available to the evolved function, at various times, presumably covering the various conditions the evolved system will be expected to act under.

Such experimental data typically come in large tables where numerous quantities are reported. In many case which quantity is the dependent variable, that is the thing that we want to predict (for example, the soft sensor value), and which other quantities are the independent variables – in other words, the information we want to use to make the prediction (say the hard sensor values), is pre-determined. If it is not, then the experimenter needs to make this decision before GP can be applied. Finally, in some practical situations, the data tables include hundreds or even thousands of variables. It is well-known, that in these cases the efficiency and effectiveness of any machine learning or program induction method, including GP, can dramatically drop as most of the variables are typically redundant or irrelevant, forcing the system to focus considerable energy on isolating the key features. It is then necessary to perform some form of feature selection – that is, we need to decide which independent variables to keep and which to leave out.

There are problems where more than one output (prediction) is required. For example, Table 2 shows a data set with four independent variables (left) and six dependent variables (right). The data were collected for the purpose of solving an inverse kinematics problem in the *Elvis* robot [219] (the robot is shown in Fig. 15 during the acquisition of a data sample). In situations like this, one can use GP individuals including multiple trees (as in Fig. 3), graph-based GP with multiple output nodes (see Sect. 6.2), linear GP with multiple

Table 2. Samples showing apparent size and location to both of *Elvis*’ eyes of his finger tip, given various right arm actuator set points (4 degrees of freedom) – see Fig. 15. When the data are used for training, GP is asked to invert the mapping and evolve functions from data collected by both cameras showing a target location to instructions to give to *Elvis*’ four arm motors so that his arm moves to the target

Arm actuator				Left eye			Right eye		
				x	y	size	x	y	size
-376	-626	1000	-360	44	10	29	-9	12	25
-372	-622	1000	-380	43	7	29	-9	12	29
-377	-627	899	-359	43	9	33	-20	14	26
-385	-635	799	-319	38	16	27	-17	22	30
-393	-643	699	-279	36	24	26	-21	25	20
-401	-651	599	-239	32	32	25	-26	28	18
-409	-659	500	-200	32	35	24	-27	31	19
-417	-667	399	-159	31	41	17	-28	36	13
-425	-675	299	-119	30	45	25	-27	39	8
-433	-683	199	-79	31	47	20	-27	43	9
-441	-691	99	-39	31	49	16	-26	45	13
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

continues for a total of 691 lines

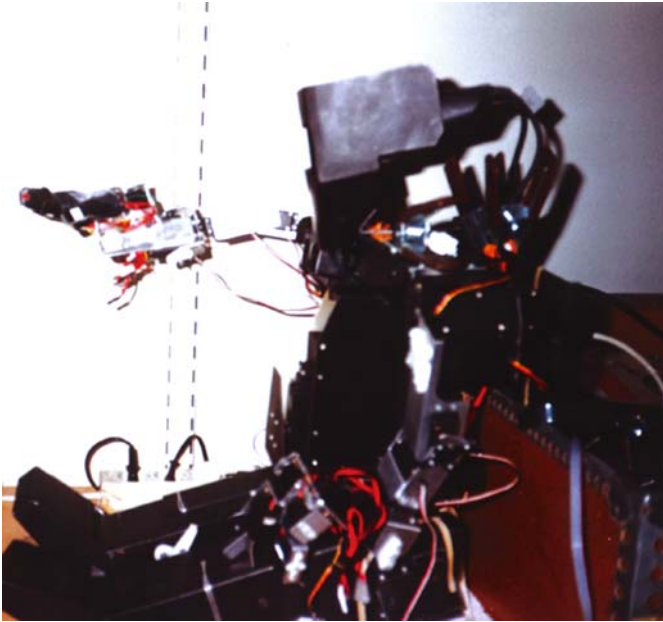


Fig. 15. Elvis sitting with right hand outstretched. The apparent position and size of the bright red laser attached to his finger tip is recorded (see Table 2). The data are then used to train a GP to move the robot's arm to a spot in three dimensions using only its eyes

output registers (see Sect. 6.1), a single GP tree with primitives operating on vectors, and so on.

Once a suitable data set is available, its dependent variables must all be represented in the primitive set. What other terminals and functions this will include depends very much on the type of the data being processed (are they numeric? strings?) and is often guided by information available to the experimenter on the process that generated the data. If something is known (or strongly suspected) about the desired structure of the evolved function (for example, the data is known to be periodic, so the function should probably be based on a something like sine), then applying some sort of constraint, like those discussed in Sect. 5.4, may be beneficial.

What is common to virtually all symbolic regression problems is that the fitness function must measure the ability of each program to predict the value of the dependent variable given the values of the independent ones (for each data point). So, most symbolic regression fitness functions tend to include sums over the (usually absolute or squared) errors measured for each record in the data set, as we did in Sect. 4.2.

The fourth preparatory step typically involves choosing a size for the population (which is often done initially based on the perceived difficulty of the problem, and is then refined based on the actual results of preliminary runs) and the balance between the selection strength (normally tuned via the tournament size) and the intensity of variation (which can be varied by varying the mutation and crossover rates, but many researchers tend to keep these fixed to some standard values).

7.2 Human Competitive Results – *The Humies*

Getting machines to produce human-like results is *the* reason for the existence of the fields of artificial intelligence and machine learning. However, it has always been very difficult to assess how much progress these fields have made towards their ultimate goal. Alan Turing understood that, to avoid human biases when assessing machines' intelligence, there is a need to evaluate their behavior objectively. This led him to propose an imitation game, now known as the Turing test [385]. Unfortunately, the Turing test is not usable in practice, and so, there is a need for more workable objective tests of machine intelligence.

Koza recently proposed to shift the attention from the notion of intelligence to the notion of *human competitiveness* [196]. A result cannot acquire the rating of 'human competitive' merely because it is endorsed by researchers *inside* the specialized fields that are attempting to create machine intelligence. A result produced by an automated method must earn the rating of 'human competitive' independently of the fact that it was generated by an automated method.

Koza proposed that an automatically-created result should be considered 'human-competitive' if it satisfies at least one of these eight criteria:

1. The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
2. The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.
3. The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
4. The result is publishable in its own right as a new scientific result, independent of the fact that the result was mechanically created.
5. The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
6. The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.

7. The result solves a problem of indisputable difficulty in its field.
8. The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

These criteria are independent of and at arms-length from the fields of artificial intelligence, machine learning, and GP.

Over the years, tens of results have passed the human-competitiveness test. Some pre-2004 human-competitive results include (see [192] for a complete list):

- Creation of quantum algorithms including: a better-than-classical algorithm for a database search problem and a solution to an AND/OR query problem [361, 362].
- Creation of a competitive soccer-playing program for the RoboCup 1997 competition [238].
- Creation of algorithms for the transmembrane segment identification problem for proteins [189, Sects. 18.8 and 18.10] and [195, Sects. 16.5 and 17.2].
- Creation of a sorting network for seven items using only 16 steps [195, Sects. 21.4.4, 23.6, and 57.8.1].
- Synthesis of analogue circuits (with placement and routing, in some cases), including: 60- and 96-decibel amplifiers [195, Sect. 45.3]; circuits for squaring, cubing, square root, cube root, logarithm, and Gaussian functions [195, Sect. 47.5.3]; a circuit for time-optimal control of a robot [195, Sect. 48.3]; an electronic thermometer [195, Sect. 49.3]; a voltage-current conversion circuit [197, Sect. 15.4.4].
- Creation of a cellular automata rule for the majority classification problem that is better than all known rules written by humans [8].
- Synthesis of topology for controllers, including: a PID (proportional, integrative, and derivative) [197, Sect. 9.2] and a PID-D2 (proportional, integrative, derivative, and second derivative) [197, Sect. 3.7] controllers; PID tuning rules that outperform the Ziegler-Nichols and Astrom-Hagglund tuning rules [197, Chapter 12]; three non-PID controllers that outperform a PID controller that uses the Ziegler-Nichols or Astrom-Hagglund tuning rules [197, Chapter 13].

In total [192] lists 36 human-competitive results. These include 23 cases where GP has duplicated the functionality of a previously patented invention, infringed a previously patented invention, or created a patentable new invention. Specifically, there are fifteen examples where GP has created an entity that either infringes or duplicates the functionality of a previously patented 20th Century invention, six instances where GP has done the same with respect to an invention patented after January 1, 2000, and two cases where GP has created a patentable new invention. The two new inventions are

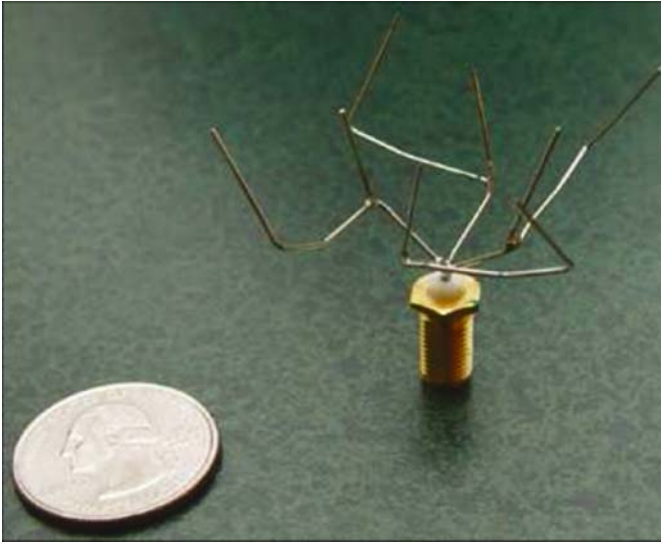


Fig. 16. Award winning human-competitive antenna design produced by GP

general-purpose controllers that outperform controllers employing tuning rules that have been in widespread use in industry for most of the 20th Century.

Many of the pre-2004 results were obtained by Koza. However, since 2004, a competition is held annually at ACM's *Genetic and Evolutionary Computation Conference* (termed the 'Human-Competitive awards – the 'Humies'). The \$10,000 prize is awarded to applications that have produced automatically-created results which are equivalent to human achievements or, better.

The Humies Prizes have typically been awarded to applications of EC to high-tech fields. Many used GP. For example, the 2004 gold medals were given for the design, via GP, of an antenna for deployment on NASA's Space Technology 5 Mission (see Fig. 16) [233] and for evolutionary quantum computer programming [358]. There were three silver medals in 2004: one for evolving local search heuristics for SAT using GP [104], one for the application of GP to the synthesis of complex kinematic mechanisms [231], and one for organization design optimization using GP [175, 176]. Also, four of the 2005 medals were awarded for GP applications: the invention of optical lens systems [1, 198], the evolution of quantum Fourier transform algorithm [247], evolving assembly programs for *Core War* [61], and various high-performance game players for Backgammon, Robocode and Chess endgame [16, 17, 135, 350]. In 2006 GP again scored a gold medal with the synthesis of interest point detectors for image analysis [381, 382], while it scored a silver medal in 2007 with

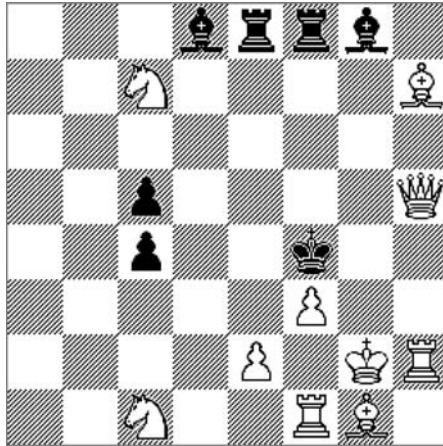


Fig. 17. Example mate-in-2 problem

the evolution of an efficient search algorithm for the Mate-in-N problem in Chess [136] (see Fig. 17).

Note that many human competitive results were presented at the Humies 2004–2007 competitions (for instance, 11 of the 2004 entries were judged to be human competitive). However, only the very best were awarded medals. So, at the time of writing we estimate that there are at least something of the order of 60 human competitive results obtained by GP. This shows GP’s potential as a powerful invention machine.

7.3 Image and Signal Processing

Hampo was one of the first people from industry to consider using GP for signal processing. He evolved algorithms for preprocessing electronic motor vehicle signals for possible use in engine monitoring and control [127]. Several applications of GP for image processing have been for military uses – for example, Tackett evolved algorithms to find tanks in infrared images [370]. Howard evolved program to pick out ships from SAR radar mounted on satellites in space [149] and to locate ground vehicles from airborne photo reconnaissance [150]. He also used GP to process surveillance data for civilian purposes, such as predicting motorway traffic jams from subsurface traffic speed measurements [148]. Using satellite SAR radar, [67] evolved algorithms to find features in polar sea ice. Optical satellite images can also be used for environmental studies [47], and for prospecting for valuable minerals [332].

Alcazar used GP to find recurrent filters (including artificial neural networks – ANN [92]) for one dimensional electronic signals [347]. Local search (simulated annealing or gradient descent) can be used to adjust or fine-tune ‘constant’ values within the structure created by genetic search [354]. [411]

have used GP to preprocess images, particularly of human faces, to find regions of interest, for subsequent analysis. (See also [382].) A particular strength of GP is its ability to take data from disparate sources [43, 369].

Zhang has been particularly active at evolving programs with GP to visually classify objects (typically coins) [419]. He has also applied GP to human speech [409]. ‘Parisian GP’ is a system in which the image processing task is split across a swarm of evolving agents (‘flies’). In [235, 236] the flies reconstruct three-dimensions from pairs of stereo images. In [235] as the flies buzz around in three-dimensions their position is projected onto the left and right of a pair of stereo images. The fitness function tries to minimize the discrepancy between the two images, thus encouraging the flies to settle on visible surfaces in the 3-D space. So, the true 3-D space is inferred from pairs of 2-D image taken from slightly different positions.

While the likes of Google have effectively indexed the written word. For speech and in particular pictures, it has been much less effective. One area where GP might be applied is in automatically indexing images. Some initial steps in this direction are given in [378].

To some extent extracting text from images (OCR) is almost a solved problem. With well formed letters and digits this is now done with near 100% accuracy as a matter of routine. However, many interesting cases remain [58] such as Arabic [182] and oriental languages, handwriting [72, 107, 200, 377] (such as the MNIST examples), other texts [327], and musical scores [317].

The scope for applications of GP to image and signal processing is almost unbounded. A promising area is medical imaging [291]. GP image techniques can also be used with sonar signals [245]. Off-line work on images, includes security and verification. For example, [386] have used GP to detect image watermarks which have been tampered with. Whilst recent work by Zhang has incorporated multi-objective fitness into GP image processing [420].

In 1999 Poli, Cagnoni and others founded the annual *European Workshop on Evolutionary Computation in Image Analysis and Signal Processing* (EvoIASP). *EvoIASP* is held every year along with the *EuroGP*. Whilst not solely dedicated to GP, many GP applications have been presented at *EvoIASP*.

7.4 Financial Trading, Time Series Prediction and Economic Modeling

GP is very widely used in these areas and it is impossible to describe all its applications. In this Section we will hint at just a few areas.

Chen has written more than 60 papers on using GP in finance and economics. Recent papers include modeling of agents in stock markets [52], game

theory [54], evolving trading rules for the S&P 500 [414] and forecasting the Heng-Sheng index [53] (see Chapter 13 of this Compendium).

The *efficient markets hypothesis* is a tenet of economics. It is founded on the idea that everyone in a market has ‘perfect information’ and acts ‘rationally’. If the efficient markets hypothesis held, then everyone would see the same value for items in the market and so agree the same price. Without price differentials, there would be no money to be made from the market itself. Whether it is trading potatoes in northern France or dollars for yen it is clear that traders are not all equal and considerable doubt has been cast on the efficient markets hypothesis. So, people continue to play the stock market. Game theory has been a standard tool used by economists to try to understand markets but is increasingly supplemented by simulations with both human and computerized agents. GP is increasingly being used as part of these simulations of social systems.

Neely and Weller of the US Federal Reserve Bank used GP to study intraday technical trading of foreign exchange to suggest the market is ‘efficient’ and found no evidence of excess returns [263, 264, 266, 267]. This negative result was criticized by [244]. Later work by [268] suggested that data after 1995 are consistent with Lo’s *adaptive markets hypothesis* rather than the *efficient markets hypothesis*. Note that here GP and computer tools are being used in a novel data-driven approach to try and resolve issues which were previously a matter of dogma.

From a more pragmatic viewpoint, Kaboudan shows GP can forecast international currency exchange rates [164], stocks [165] and stock returns [163]. [383] continue to apply GP to a variety of financial arenas, including: betting, forecasting stock prices [109], studying markets [158] and arbitrage [243]. [15, 74, 75] and HSBC also use GP in foreign exchange trading. Pillay has used GP in social studies and teaching aids in education, for instance, [289]. As well as trees [187], other types of GP have been used in finance, for example [270].

The *Intl. Conf. on Computing in Economics and Finance* (CEF) has been held every year since 1995. It regularly attracts GP papers, many of which are on-line. In 2007 Brabazon and O’Neill established the *European Workshop on Evolutionary Computation in Finance and Economics* (EvoFIN); *EvoFIN* is held with *EuroGP*.

7.5 Industrial Process Control

Of course most industrialists have little time to spend on academic reporting. A notable exception is Dow Chemical, where Kordon’s group has been very active [46, 185, 254]. [184] describes where industrial GP stands now and how it will progress. Another active collaboration is that between Kovacic and Balic, who have used GP in the computer numerical control of industrial milling and cutting machinery [186]. The partnership of Deschaine and Francone [100] is

most famous for their use of *Discipulus* [99] for detecting bomb fragments and unexploded ordnance UXO [76]. *Discipulus* has been used as an aid in the development of control systems for rubbish incinerators [77].

One of the earliest users of GP in control was Willis' Chemical Engineering group at Newcastle, which used GP to model flow in a plasticating extruder [399]. They also modelled extruding food [251] and control of chemical reactions in continuous stirred tank reactors [342]. Marenbach investigated GP in the control of biotech reactors [242]. [398] surveyed GP applications, including to control. Other GP applications to plastic include [38]. Lewin has applied GP to the control of an integrated circuit fabrication plant [68, 228]. Domingos worked on simulations of nuclear reactors (PWRs to be exact) to devise better ways of preventing xenon oscillations [83]. GP has also been used to identify which state a plant to be controlled is in (in order to decide which of various alternative control laws to apply). For example, Fleming's group in Sheffield used multi-objective GP [141, 329] to reduce the cost of running aircraft jet engines [14, 93]. [4] surveys GP and other AI techniques applied in the electrical power industry.

7.6 Medicine, Biology and Bioinformatics

GP has long been applied to medicine, biology and bioinformatics. Early work by Handley [128] and Koza [191] used GP to make predictions about the behavior and properties of biological systems, principally proteins. Oakley, a practising medical doctor, used GP to model blood flow in toes [276] as part of his long term interests in frostbite.

In 2002 Banzhaf and Foster organized *BioGEC*, the first *GECCO Workshop on Biological Applications of Genetic and Evolutionary Computation*. *BioGEC* has become a bi-annual feature of the annual *GECCO* conference. Half a year later Marchiori and Corne organized *EvoBio*, the *European Conf. on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. *EvoBio* is held every year alongside *EuroGP*; GP figures heavily in both *BioGEC* and *EvoBIO*.

GP is often used in data mining. Of particular medical interest are very wide data sets, with many inputs per sample. Examples include infrared spectra [89, 90, 117, 119, 131, 159, 250, 373, 387], single nuclear polymorphisms [26, 320, 345] and Affymetrix GeneChip microarray data [71, 91, 139, 142, 147, 217, 229, 230, 412].

Kell and his colleagues in Aberystwyth have had great success in applying GP widely in bioinformatics (see infrared spectra above and [2, 70, 113, 118, 160, 168–171, 349, 407]). Another very active group is that of Moore and his colleagues at Vanderbilt [261, 262, 325, 326].

Computational chemistry is widely used in the drug industry. The properties of simple molecules can be calculated. However, the interactions between

chemicals which might be used as drugs and medicinal targets within the body are beyond exact calculation. Therefore, there is great interest in the pharmaceutical industry in approximate *in silico* models which attempt to predict either favourable or adverse interactions between proto-drugs and biochemical molecules. Since these are computational models, they can be applied very cheaply in advance of manufacture of chemicals, to decide which of the myriad of chemicals might be worth further study. Potentially such models can make a huge impact both in terms of money and time without being anywhere near 100% correct. Machine learning and GP have both been tried. GP approaches include [21, 27, 43, 95, 114, 120, 131, 134, 199, 351, 388, 393].

7.7 Mixing GP with Other Techniques

GP can be hybridised with other techniques. Iba [152], Nikolaev [269], and Zhang [417] have incorporated information theoretic and minimum description length ideas into GP fitness functions to provide a degree of regularization and so avoid over-fitting (and bloat, see Sect. 9.3). As mentioned in Sect. 5.4 computer language grammars can be incorporated into GP. Indeed Wong [401–403, 405] has had success integrating these with GP. The use of simulated annealing and hill climbing to locally fine tune parts of solutions found by GP was described in Sect. 2.

7.8 GP to Create Searchers and Solvers – Hyper-Heuristics

Hyper-heuristics could simply be defined as ‘heuristics to choose other heuristics’ [40]. A heuristic is considered as a rule-of-thumb or ‘educated guess’ that reduces the search required to find a solution. The difference between meta-heuristics and hyper-heuristics is that the former operate directly on the problem search space with the goal of finding optimal or near-optimal solutions. The latter, instead, operate on the heuristics search space (which consists of the heuristics used to solve the target problem). The goal then is finding or generating high-quality heuristics for a problem, for a certain class of instances of a problem, or even for a particular instance.

GP has been very successfully used as a hyper-heuristic. For example, GP has evolved competitive SAT solvers [19, 20, 103, 177], state-of-the-art or better than state-of-the-art bin packing algorithms [41, 42, 313], particle swarm optimizers [310, 311], evolutionary algorithms [277], and travelling salesman problem solvers [172–174, 278].

7.9 Artistic

Computers have long been used to create purely aesthetic artifacts. Much of today’s computer art tends to ape traditional drawing and painting, producing static pictures on a computer monitor. However, the immediate advantage

of the computer screen – movement – can also be exploited. In both cases EC can and has been exploited. Indeed with evolution's capacity for unlimited variation, EC offers the artist the scope to produce ever changing works. Some artists have also worked with sound.

The use of GP in computer art can be traced back at least to the work of Sims [353] and Latham. Jacob provides many examples [155, 156]. Since 2003, *EvoMUSART* has been held every year with *EuroGP*. McCormack considers the recent state of play in evolutionary art and music [249]. Many recent techniques are described in [241].

Evolutionary music [379] has been dominated by Jazz [359], which is not to everyone's taste; an exception is Bach [94]. Most approaches to evolving music have made at least some use of interactive evolution [371] in which the fitness of programs is provided by users, often via the Internet [5, 49]. The limitation is almost always finding enough people willing to participate [211]. Funes reports experiments which attracted thousands of people via the Internet who were entertained by evolved *Tron* players [105]. Costelloe tried to reduce the human burden in [62]; algorithmic approaches are also possible [59, 153].

One of the sorrows of AI is that as soon as it works it stops being AI (and celebrated as such) and becomes computer engineering. For example, the use of computer generated images has recently become cost effective and is widely used in Hollywood. One of the standard state-of-the-art techniques is the use of Reynold's swarming 'boids' [321] to create animations of large numbers of rapidly moving animals. This was first used in *Cliffhanger* (1993) to animate a cloud of bats. Its use is now common place (herds of wildebeest, schooling fish, and the like); in 1997 Craig was awarded an Oscar.

7.10 Entertainment and Computer Games

Today the major usage of computers is interactive games [315]. There has been a little work on incorporating artificial intelligence into mainstream commercial games. The software owners are not keen on explaining exactly how much AI they use or giving away sensitive information on how they use AI. Work on GP and games includes [16, 389]. Since 2004 the annual *CEC Conference* has included sessions on EC in games. After chairing the *IEEE Symposium on Computational Intelligence and Games 2005* at Essex University, Lucas founded the IEEE Computational Intelligence Society's Technical Committee on Games. GP features heavily in the Games TC's activities, for example Othello, Poker, Backgammon, Draughts, Chess, Ms Pac-Man, robotic football and radio controlled model car racing.

7.11 Where can we Expect GP to Do Well?

GP and other EC methods have been especially productive in areas having some or all of the following properties:

- The interrelationships among the relevant variables is unknown or poorly understood (or where it is suspected that the current understanding may possibly be wrong).
- Finding the size and shape of the ultimate solution to the problem is a major part of the problem.
- Large amounts of primary data requiring examination, classification, and integration is available in computer-readable form.
- There are good simulators to test the performance of tentative solutions to a problem, but poor methods to directly obtain good solutions.
- Conventional mathematical analysis does not, or cannot, provide analytic solutions.
- An approximate solution is acceptable (or is the only result that is ever likely to be obtained).
- Small improvements in performance are routinely measured (or easily measurable) and highly prized.

The best predictor of future performance is the past. So, we should expect GP to continue to be successful in application domains with these features.

8 Tricks of the Trade

8.1 Getting Started

Newcomers to the field of GP often ask themselves (and/or other more experienced genetic programmers) questions such as:

1. What is the best way to get started with GP? Which papers should I read?
2. Should I implement my own GP system or should I use an existing package? If so, what package should I use?

Let us start from question 1. A variety of sources of information about GP are available (many of which are listed in the Resources Appendix). Consulting information available on the Web is certainly a good way to get quick answers for a ‘newbie’ who wants to know what GP is. These answers, however, will often be too shallow for someone who really wants to then apply GP to solve practical problems. People in this position should probably invest some time going through more detailed accounts, such [25, 188, 222] or some of the other books in the Resources Appendix. Technical papers may be the next stage. The literature on GP is now quite extensive. So, although this is easily accessible thanks to the complete online bibliography (<http://www.cs.bham.ac.uk/~wbl/biblio/>), newcomers will often need to be selective in what they read. The objective here may be different for different types of readers. Practitioners should probably identify and read only papers which deal with the same problem they are interested in. Researchers and PhD students interested in developing a deeper understanding of GP should also make sure they identify and read as many seminal papers as possible,

including papers or books on empirical and theoretical studies on the inner mechanisms and behavior of GP. These are frequently cited in other papers and so can easily be identified.

The answer to question 2 depends on the particular experience and background of the questioner. Implementing a simple GP system from scratch is certainly an excellent way to make sure one really understands the mechanics of GP. In addition to being an exceptionally useful exercise, this will always result in programmers knowing their systems so well that they will have no problems customizing them for specific purposes (for example, adding new, application specific genetic operators or implementing unusual, knowledge-based initialization strategies). All of this, however, requires reasonable programming skills and the will to thoroughly test the resulting system until it fully behaves as expected. If the skills or the time are not available, then the best way to get a working GP application is to retrieve one of the many public-domain GP implementations and adapt this for the user's purposes. This process is faster, and good implementations are often quite robust, efficient, well-documented and comprehensive. The small price to pay is the need to study the available documentation and examples. These often explain also how to modify the GP system to some extent. However, deeper modifications (such as the introduction of new or unusual operators) will often require studying the actual source code of the system and a substantial amount of trial and error. Good, publicly-available GP implementations include: `Lil-GP`, `ECJ`, and `DGPC`.

While perhaps to some not as exciting as coding or running GP, a through search of the literature can avoid 're-inventing the wheel'.

8.2 Presenting Results

It is so obvious that it is easy to forget one major advantage of GP: we create *visible* programs. That is, the way they work is accessible. This need not be the case with other approaches. So, when presenting GP results, as a matter of routine one should perhaps always make a comprehensible slide or figure which contains the whole evolved program,² trimming unneeded details (such as removing excess significant digits) and combining constant terms. Naturally, after cleaning up the answer, one should make sure the program still works.

If one's goal is to find a comprehensible model, in practice it must be small. A large model will not only be difficult to understand but also may over-fit the training data [112]. For this reason (and possibly others), one should use one of the anti-bloat mechanisms described in Sect. 9.3.

There are methods to automatically simplify expressions (for example, in `Mathematica` and `Emacs`). However, since in general there is an exponentially

² The program `Lisp2dot` can be of help in this.

large number of equivalent expressions, automatic simplification is hard. Another way is to use GP. After GP has found a suitable but large model, one can continue evolution changing the fitness function to include a second objective: that the model be as small as possible [203]. GP can then trim the trees but ensure the evolved program still fits the training data.

It is important to use the language that one's customers, audience or readers use. For example, if the fact that GP discovers a particular chemical is important, one should make this fact stand out, say by using colours. Also, GP's answer may have evolved as a tree but, if the customers use Microsoft Excel, it may be worthwhile translating the tree into a spreadsheet formula.

Also, one should try to discover how the customers intend to validate GP's answer. Do not let them invent some totally new data which has nothing to do with the data they supplied for training ('just to see how well it does...'). Avoid customers with contrived data. GP is not god, it knows nothing about things it has not seen. At the same time users should be scrupulous about their own use of holdout data. GP is a very powerful machine learning technique. With this comes the ever present danger of over-fitting. One should never allow performance on data reserved for validation to be used to choose which answer to present to the customer.

8.3 Reducing Fitness Evaluations/Increasing their Effectiveness

While admirers of linear GP will suggest that machine code GP is the ultimate in speed, tree GP can be made faster in a number of ways. The first is to reduce the number of times a tree is evaluated. Many applications find the fitness of trees by running them on multiple training examples. However, ultimately the point of fitness evaluation is to make a binary decision: does this individual get a child or not. Indeed usually a noisy selection technique is used such as roulette wheel, SUS [22], or tournament selection. Stochastic selection is an essential part of genetic search but it necessarily injects noise into the vital decision of which points in the search to proceed from and which to abandon. The overwhelming proportion of GP effort (or indeed any EC technique) goes into adjusting the probability of the binary decision as to whether each individual in the population should be allowed to reproduce or not. If a program has already demonstrated it works very badly compared to the rest of the population on a fraction of the available training data, it is likely not to have children. Conversely, if it has already exceeded many programs in the population after being tested on only a fraction of the training set, it is likely to have a child [203]. In either case, it is apparent that we do not need to run it on the remaining training examples. Teller and Andre developed this idea into an effective algorithm [376].

As well as the computational cost, there are other aspects of using all the training data all the time. It gives rise to a static fitness function. Arguably

this tends to evolve the population into a cul-de-sac where the population is dominated by offspring of a single initial program which did well of some fraction of the training data but was unable to fit others. A static fitness function can easily have the effect that the other good programs which perhaps did well on other parts of the training data get lower fitness scores and fewer children.

With high selection pressure, it takes surprisingly little time for the best individual to dominate the whole population. Goldberg calls this the ‘take over time’ [115]. This can be made quite formal [31, 85]. However, for tournament selection, a simple rule of thumb is often sufficient. If T is the tournament size, about $\log_T(\text{Pop size})$ generations are needed for the whole population to become descendants of a single individual. For example, if we use binary tournaments ($T = 2$), then ‘take over’ will require about ten generations for a population of 1,024. Alternatively, if we have a population of one million (10^6) and use ten individuals in each tournament ($T = 10$), then after about six generations more or less everyone will have the same great₆ great₅ great₄ great₃ grand₂ mother₁.

Gathercole investigated a number of ways of changing which training examples to use as the GP progressed [110, 111]. (Siegel proposed a rather different implementation in [352].) This juggles a number of interacting effects. Firstly, by using only a subset of the available data, the GP fitness evaluation takes less time. Secondly, by changing which examples are being used, the evolving population sees more of the training data and, so, is less liable to over fit a fraction of it. Thirdly, by randomly changing the fitness function, it becomes more difficult for evolution to produce an over specialized individual which takes over the population at the expense of solutions which are viable on other parts of the training data. Dynamic Subset Selection (DSS) appears to have been the most successful of Gathercole’s suggested algorithms. It has been incorporated into *Discipulus*. Indeed a huge data mining application [66] recently used DSS.

Where each fitness evaluation may take a long time, it may be attractive to interrupt a long running program in order to let others run. In GP systems which allow recursion or contain iterative elements [36, 203, 400, 404] it is common to enforce a time limit, a limit on the number of instructions executed, or a bound on the number of times a loop is executed. Maxwell proposed [248] a solution to the question of what fitness to we give to a program we have interrupted. He allowed each program in the population a quantum of CPU time. When the program uses up its quantum it is check-pointed. When the program is check-pointed, sufficient information (principally the program counter and stack) is saved so that it can be restarted from where it got to later. (Many multi-tasking operating systems do something similar.) In Maxwell’s system, he assumed the program gained fitness as it ran. For example, each time it correctly processes a fitness case, its fitness is incremented. So the fitness of

a program is defined while it is running. Tournament selection is then performed. If all members of the tournament have used the same number of CPU quanta, then the program which is fitter is the winner. However, if a program has used less CPU than the others (and has a lower fitness) then it is restarted from where it was and is run until it has used as much CPU as the others. Then fitnesses are compared in the normal way.

Teller had a similar but slightly simpler approach: everyone in the population was run for the same amount of time. When the allotted time elapses the program is aborted and an answer extracted from it, regardless of whether it was ready or not; he called this an ‘any time’ approach [374]. This suits graph or linear GP, where it is easy to designate a register as the output register. The answer can be extracted from this register or from an indexed memory cell at any point (including whilst the programming is running). Other any time approaches include [220, 360].

A simple technique to speed up the evaluation of complex fitness functions is to organize the fitness function into stages of progressively increasing computational cost. Individuals are evaluated stage by stage. Each stage contributes to the overall fitness of a program. However, individuals need to reach a minimum fitness value in each stage in order for them to be allowed to progress to the next stage and acquire further fitness. Often different stages represent different requirements and constraints imposed on solution.

Recently, a sophisticated technique, called *backward chaining GP*, has been proposed [297, 301–303] that can radically reduce the number of fitness evaluations in runs of GP (and other EAs) using tournament selection with small tournament sizes. Tournament selection randomly draws programs from the population to construct tournaments, the winners of which are then selected. Although this process is repeated many times in each generation, when the tournaments are small there is a significant probability that an individual in the current generation is never chosen to become a member of any tournament. By reordering the way operations are performed in GP, backward chaining GP exploits this not only to avoid the calculation of individuals that are never sampled, but also to achieve higher fitness sooner.

8.4 Co-Evolution

One way of viewing DSS is as automated co-evolution. In co-evolution there are multiple evolving species (typically two) whose fitness depends upon the other species. (Of course, like DSS, co-evolution can be applied to linear and other types of GP as well as tree GP.) One attraction of co-evolution is that it effectively produces the fitness function for us. There have been many successful applications of co-evolution [16, 35, 39, 48, 82, 108, 140, 338, 346, 400], however it complicates the already complex phenomena taking place in the presence of dynamic fitness functions still further. Therefore, somewhat reluctantly, at present it appears to be beneficial to use co-evolution only if an

application really requires it. Co-evolution may suffer from unstable populations. This can occur in nature, oscillations in Canadian Lynx and Snowshoe Hare populations being a famous example. There are various ‘hall of fame’ techniques [106], which try to damp down oscillations and prevent evolution driving competing species in circles.

8.5 Reducing Cost of Fitness with Caches

In computer hardware it is common to use data caches which automatically hold copies of data locally in order to avoid the delays associated with fetching it from disk or over a network every time it is needed. This can work well where a small amount of data is needed many times over a short interval. Caches can also be used to store results of calculations, thereby avoiding the re-calculation of data [129]. GP populations have enormous amounts of common code [203, 215, 220]. This is after all how genetic search works: it promotes the genetic material of fit individuals. So, typically in each generation we see many copies of successful code. In a typical GP system, but by no means all GP systems, each subtree has no side-effects. This means its results pass through its root node in a well organized and easy to understand fashion. Thus, if we remember a subtree’s inputs and output when it was run before, we can avoid re-executing code whenever we are required to run the subtree again. Note that this is true irrespective of whether we need to run the same subtree inside a different individual or at a different time (namely, a later generation). Thus, if we stored the output with the root node, we need only run the subtree once, for a given set of inputs. Whenever the interpreter comes to evaluate the subtree, it needs only to check if the root contains a cache of the values the interpreter calculated last time, thus saving considerable computation time. However, there is a problem: not only must the answer be stored, but the interpreter needs to know that the subtree’s inputs are the same too.

The common practices of GP come to our aid here. Usually every tree in the population is run on exactly the same inputs for each of the fitness cases. Thus, for a cache to work, the interpreter does not need to know in detail which inputs the subtree has or their exact values corresponding to every value calculated by the subtree. It need only know which of the fixed set of test cases was used.

A simple cache implementation is to store a vector of values returned by each subtree. The vector is as long as the number of test cases. Whenever a subtree is created (namely, in the initial generation, by crossover or by mutations) the interpreter is run and the cache of values for its root node is set. Note this is recursive, so caches can also be calculated for subtrees within it at the same time. Now when the interpreter is run and comes to a subtree’s root node, it will know which test case it is running and instead of interpreting the subtree it simply retrieves the value it calculated using the

test case's number as an index into the cache vector. This could be many generations after the subtree was originally created.

If a subtree is created by mutation, then its cache of values will be initially empty and will have to be calculated. However, this costs no more than without caches.

When subtrees are crossed over the subtree's cache remains valid and so cache values can be crossed over like the code.

When code is inserted into an existing tree, be it by mutation or crossover, the chance that the new code behaves identically to the old code is normally very small. This means the caches of every node between the new code and the root node may be invalid. The simplest thing is to re-evaluate them all. This sounds expensive, but remember the caches in all the other parts of the individual remain valid and so can be used when the cache above them is re-evaluated. Thus, in effect, if the crossed over code is inserted at level- d , only d nodes need to be evaluated. Recent analysis [57, 81, 222, 312] has shown that GP trees tend not to have symmetric shapes, and many leaves are very close to the root. Thus in theory (and in practice) considerable computational saving can be made by using fitness caches. Sutherland is perhaps the best known GP system which has implemented fitness caches [253]. As well as the original DAG implementation [129]; other work includes [57, 166, 410].

In [203] we used fitness caches in evolved trees with side effects by exploiting syntax rules about where in the code the side-effects could lie. The whole question of monitoring how effective individual caches are, what their hit-rates are, and so on, has been little explored. In practice, in many common GP systems, impressive savings have been made by simple implementations, with little monitoring and rudimentary garbage collection. While it is possible to use hashing schemes to efficiently find common code, in practice assuming that common code only arises because it was inherited from the same location (for instance, by crossing over) is sufficient.

8.6 GP Running in Parallel

In contrast to much of computer science, EC can be readily run on parallel computer hardware; indeed it is 'embarrassingly parallel' [7]. For example, when Turton ran GP on a Cray supercomputer he obtained about 30% of its theoretical peak performance, embarrassing his 'supercomputer savvy' colleagues who rarely got better than a few percent out of it [280].

There are two important aspects of parallel evolutionary algorithms. These are equally important but often confused. The first is the traditional aspect of parallel computing. We port an existing algorithm onto a supercomputer so that it runs faster. The second aspect comes from the biological inspiration for EC.

In Nature everything happens in parallel. Individuals succeed or not in producing and raising children at the same time as other members of their species. The individuals are spread across oceans, lakes, rivers, plains, forests, mountain chains, and the like. It was this geographic spread that led Wright to propose that geography and changes to it are of great importance to the formation of new species and so to natural evolution as a whole [408].

While in Nature geographically distributed populations are a necessity, in EC we have a choice. We can run GP on parallel hardware so as to speed up runs, or we can distribute GP populations over geographies so as obtain some of the benefits it brings to natural evolution. In the following we will discuss both ideas. It is important to note, however, that one does not need to use parallel hardware to use geographically distributed GP populations. Although parallel hardware naturally lends itself to realize *physically-distributed* populations, one can obtain similar benefits by using *logically-distributed* populations in a single machine.

Master-Slave GP

If the objective is purely to speed up runs, we may want our GP to work exactly the same as it did on a single computer. This is possible, but to achieve it we have to be very careful to ensure that even if some parts of the population are evaluated quicker, that parallelization does not change how we do selection and which GP individual crosses over with the other. Probably the easiest way to implement this is the master-slave model.

In the master-slave model [285], breeding, selection, crossover, mutation and so on are exactly as on a single computer, and only fitness evaluation is spread across a network of computers. Each GP individual and its fitness cases are sent across the network to a compute node. The central node waits for it to return the individual's fitness. Since individuals and fitness values are small, this can be quite efficient. The central node is an obvious bottleneck. Also, a slow compute node or a lengthy fitness case will slow down the whole GP population, since eventually its result will be needed before moving onto the next generation.

Geographically Distributed GP

As we have seen, unless some type of synchronization or check pointing is imposed, say at the end of each generation, the parallel GP will not be running the same algorithm as the single node version, and, so, it will almost certainly produce different answers. If the population is divided up into sub-populations (known as *demes* [60, 80, 203]) and the exchange of individuals among populations is limited (both in terms of how many individuals are allowed to migrate per generation and a geography that constraints which populations can communicate with which), then parallelization can bring benefits similar

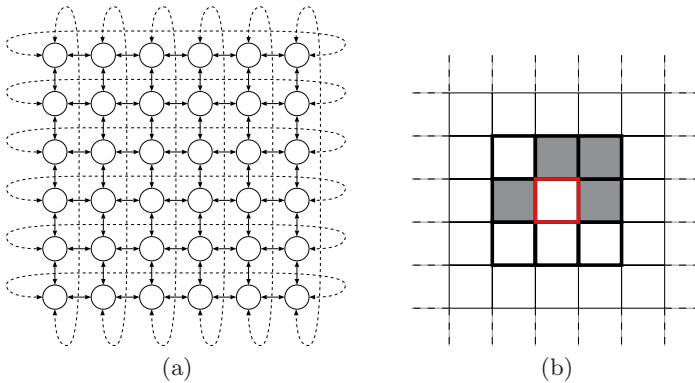


Fig. 18. Spatially structured GP populations. (a) Toroidal grid of demes, where each deme (node) contains a sub-population, and demes periodically exchange a small group of high-fitness individuals using a grid of communication channels. (b) Fine-grained distributed GP, where each grid cell contains one individual and where the selection of a mating partner for the individual in the centre cell is performed by executing a tournament among randomly selected individuals (for instance, the individuals shaded) in its 3×3 neighbourhood

to those found in Nature by [408]. For example, it may be that with limited migration between compute nodes, the evolved populations on adjacent nodes will diverge and that this increased diversity may lead to better solutions.

When Koza first started using GP on a network of Transputers [6], Andre experimentally determined the best immigration rate for their problem. He suggested Transputers arranged in an asynchronous 2-D toroidal square grid (such as the one in Fig. 18a) should exchange 2% of their population with their four neighbours.

Densely connected grids have been widely adopted in parallel GP. Usually they allow innovative partial solutions to quickly spread. However, the GA community reported better results from less connected topologies, such as arranging the compute node's populations in a ring, so that they could transport genes only between themselves and their two neighbours [365]. Potter argues in favour of spatial separation in populations (see Fig. 18b) [314]. Goldberg also suggests low migration rates [116]. In [396], Whitley includes some guidance on parallel GAs.

While many have glanced enviously at Koza's 1000 node Beowulf [368], a supercomputer [29, 162] is often not necessary. Many businesses and research centres leave computers permanently switched on. During the night their computational resources tend to be wasted. This computing power can easily and efficiently be used to execute distributed GP runs overnight. Typically GP does not demand a high performance bus to interconnect the compute nodes, and, so, existing office Ethernet LANs are often sufficient. Whilst parallel



Fig. 19. A global population [213]; the straight lines show connections between major sites in a continuously evolving L-System

GP systems can be implemented using MPI [391] or PVM [96], the use of such tools is not necessary: simple Unix commands and port-to-port HTTP is sufficient [307]. The population can be split and stored on modest computers. With only infrequent interchange of parts of the population or fitness values little bandwidth is needed. Indeed a global population spread via the Internet [213], *à la seti@home*, is perfectly feasible [56]. (See Fig. 19). Other parallel GPs include [6, 44, 50, 63, 97, 98, 125, 183, 232, 246, 334, 335, 372].

GP Running on GPUs

Modern PC graphics cards contain powerful Graphics Processing Units (GPUs) including a large number of computing components. For example, it is not atypical to have 128 streaming processors on a single PCI graphics card. In the last few years there has been an explosion of interest in porting scientific or general purpose computation to mass market graphics cards [286].

Indeed, the principal manufactures (nVidia and ATI) claim faster than Moore's Law increase in performance, suggesting that GPU floating point performance will continue to double every twelve months, rather than the 18–24 months observed [260] for electronic circuits in general and personal computer CPUs in particular. In fact, the apparent failure of PC CPUs to keep up with Moore's law in the last few years makes GPU computing even more attractive. Even today's bottom of the range GPUs greatly exceed the floating point performance of their hosts' CPU. However, this speed comes at a price, since GPUs provide a restricted type of parallel processing, often referred to a single instruction multiple data (SIMD) or single program multiple data

(SPMD). Each of the many processors simultaneously runs the same program on different data items.

There have been a few GP experiments with GPUs [55, 86, 130, 216, 218, 237, 255, 319]. So far, in GP, GPUs have just been used for fitness evaluation. Harding used the Microsoft research GPU development `Direct X` tools to allow him to compile a whole population of Cartesian GP network programs into a GPU program [132] which was loaded onto his Laptop's GPU in order to run fitness cases. We used [216, 218] a SIMD interpreter [162] written in C++ using RapidMind's `GCC OpenGL` framework to simultaneously run up to a quarter of a million GP trees on an `nVidia` GPU. A conventional tree GP S-expression can be linearized. We used reverse polish notation (RPN) – that is, post fix notation – rather than pre-fix notation. RPN avoids recursive calls in the interpreter [216]. Only small modifications are needed to do crossover and mutation so that they act directly on the RPN expressions. This means the same representation is used on both the host and the GPU. In both Cartesian and tree GP the genetic operations are done by the host CPU. Wong showed, for a genetic algorithm, these too can be done by the GPU [406].

Although each of the GPU's processors may be individually quite fast and the manufacturers claim huge aggregate FLOP ratings, the GPUs are optimized for graphics work. In practice it is hard to keep all the processors fully loaded. Nevertheless 30 GFLOPS has been achieved [218]. Given the differences in CPU and GPU architectures and clock speeds, often the speedup from using a GPU rather than the host CPU is the most useful statistic. This is obviously determined by many factors, including the relative importance of amount of computation and size of data. The measured RPN tree speedups were 7.6 [218] and 12.6 [216].

8.7 GP Trouble-Shooting

A number of practical recommendations for GP work can be made. To a large extent the advice in [181] and [188] remains sound. However, we also suggest:

- GP populations should be closely studied as they evolve. There are several properties that can be easily measured which give indications of problems:
 - *Frequency of primitives*. Recognizing when a primitive has been completely lost from the population (or its frequency has fallen to a low level, consistent with the mutation rate) may help to diagnose problems.
 - *Population variety*. If the variety – the number of distinct individuals in the population – falls below 90% of the population size, this indicates there may be a problem. However, a high variety does not mean the reverse. GP populations often contain introns, and so programs which are not identical may behave identically. Being different, these

individuals contribute to a high variety, that is a high variety need not indicate all is well. Measuring phenotypic variation (that is, diversity of behavior) may also be useful.

- Measures should be taken to encourage population diversity. Panmictic steady-state populations with tournament selection, reproduction and crossover may converge too readily.³ The above-mentioned metrics may indicate if this is happening in a particular case. Possible solutions include:
 - Not using the reproduction operator.
 - Addition of one or more mutation operators.
 - Smaller tournament sizes and/or using uniform random selection (instead of the standard negative tournaments) to decide which individuals to remove from the population. Naturally, the latter means the selection scheme is no longer elitist; it may be worthwhile forcing it to be elitist.
 - Splitting large populations into semi-isolated demes.⁴
 - Using fitness sharing to encourage the formation of many fitness niches.
- Use of *fitness caches* (either when executing an individual or between ancestors and children) can reduce run time and may repay the additional work involved with using them.
- Where GP run time is long, it is important to periodically save the current state of the run. Should the system crash, the run can be restarted from part way through rather than at the start. Care should be taken to save the entire state, so restarting a run does not introduce any unknown variation. The bulk of the state to be saved is the current population. This can be compressed, for example by using `gzip`. While compression can add a few percent to run time, reductions in disk space to less than one bit per primitive in the population have been achieved.

9 Genetic Programming Theory

Most of this Chapter is about the mechanics of GP and its practical use for solving problems. We have looked at GP from a problem-solving and engineering point of view. However, GP is a non-deterministic searcher and, so, its behavior varies from run to run. It is also a complex adaptive system which sometimes shows complex and unexpected behaviors (such as bloat). So, it

³ In a panmictic population no mating restrictions are imposed as to which individual mates with which.

⁴ What is meant by a 'large population' has changed over time. In the early days of GP populations of 1,000 or more could be considered large. However, CPU speeds and computer memory have increased exponentially over time. So, at the time of writing it is not unusual to see populations of hundred of thousands or millions of individuals being used in the solution of hard problems. Research indicates that there are benefits in splitting populations into demes even for much smaller populations.

is only natural to be interested in GP also from the scientific point of view. That is, we want to understand why can GP solve problems, how it does it, what goes wrong when it cannot, what are the reasons for certain undesirable behaviors, what can we do to get rid of them without introducing new (and perhaps even less desirable) problems, and so on.

GP is a search technique that explores the space of computer programs. The search for solutions to a problem starts from a group of points (random programs) in this search space. Those points that are above average quality are then used to generate a new generation of points through crossover, mutation, reproduction and possibly other genetic operations. This process is repeated over and over again until a stopping criterion is satisfied. If we could *visualize* this search, we would often find that initially the population looks like a cloud of randomly scattered points, but that, generation after generation, this cloud changes shape and moves in the search space. Because GP is a stochastic search technique, in different runs we would observe different trajectories. These, however, would show clear regularities which would provide us with a deep understanding of how the algorithm is searching the program space for the solutions. We would probably readily see, for example, why GP is successful in finding solutions in certain runs, and unsuccessful in others. Unfortunately, it is normally impossible to exactly visualize the program search space due to its high dimensionality and complexity, and so we cannot just use our senses to understand GP.

9.1 Mathematical Models

In this situation, in order to gain an understanding of the behavior of a GP system one can perform many real runs and record the variations of certain numerical descriptors (like the average fitness or the average size of the programs in the population at each generation, the average number of inactive nodes, the average difference between parent and offspring fitness, and so on). Then, one can try to suggest explanations about the behavior of the system which are compatible with (and could explain) the empirical observations. This exercise is very error prone, though, because a genetic programming system is a complex adaptive system with ‘zillions’ of degrees-of-freedom. So, any small number of statistical descriptors is likely to be able to capture only a tiny fraction of the complexities of such a system. This is why in order to understand and predict the behavior of GP (and indeed of most other evolutionary algorithms) in precise terms we need to define and then study *mathematical models of evolutionary search*.

Schema theories are among the oldest and the best known models of evolutionary algorithms [145, 397]. Schema theories are based on the idea of partitioning the search space into subsets, called *schemata*. They are concerned with modeling and explaining the dynamics of the distribution of the population over the schemata. Modern GA schema theory [366, 367] provides

exact information about the distribution of the population at the next generation in terms of quantities measured at the current generation, without having to actually run the algorithm.⁵

The theory of schemata in GP has had a difficult childhood. Some excellent early efforts led to different worst-case-scenario schema theorems [3, 188, 283, 298, 330, 394]. Only very recently have the first exact schema theories become available [293–295] which give exact formulations (rather than lower bounds) for the expected number of individuals sampling a schema at the next generation. Initially [293, 295], these exact theories were only applicable to GP with one-point crossover (see Sect. 2.4). However, more recently they have been extended to the class of homologous crossovers [309] and to virtually all types of crossovers that swap subtrees [305, 306], including standard GP crossover with and without uniform selection of the crossover points, one-point crossover, context-preserving crossover and size-fair crossover which have been described in Sect. 2.4, as well as more constrained forms of crossover such as strongly-typed GP crossover (see Sect. 5.4), and many others.

9.2 Search Spaces

Exact schema-based models of GP are probabilistic descriptions of the operations of selection, reproduction, crossover, and mutation. They make it explicit how these operations determine the areas of the program space that will be sampled by GP and with which probability. However, these models treat the fitness function as a black box. That is, there is no notion of the fact that in GP, unlike other evolutionary techniques, the fitness function involves the execution of computer programs with different input data. In other words, schema theories do not tell us how fitness is distributed in the search space.

The characterization of the space of computer programs explored by GP has been another main topic of theoretical research [222].⁶ In this category are theoretical results showing that the distribution of functionality of non Turing-complete programs approaches a limit as program length increases. That is, although the number of programs of a particular length grows exponentially with length, beyond a certain threshold the fraction of programs implementing any particular functionality is effectively constant. For example, in Fig. 20 we plot the proportion of binary program trees composed of NAND gates which implement each of the $2^{2^3} = 256$ Boolean functions of three inputs.

⁵ Other models of evolutionary algorithms exist, such those based on Markov chain theory (for example [69, 271]) or on statistical mechanics (for instance, [316]). Only Markov models [258, 308, 309] have been applied to GP, but they are not as developed as schema theory.

⁶ Of course results describing the space of all possible programs are widely applicable, not only to GP and other search-based automatic programming techniques, but also to many other areas ranging from software engineering to theoretical computer science.

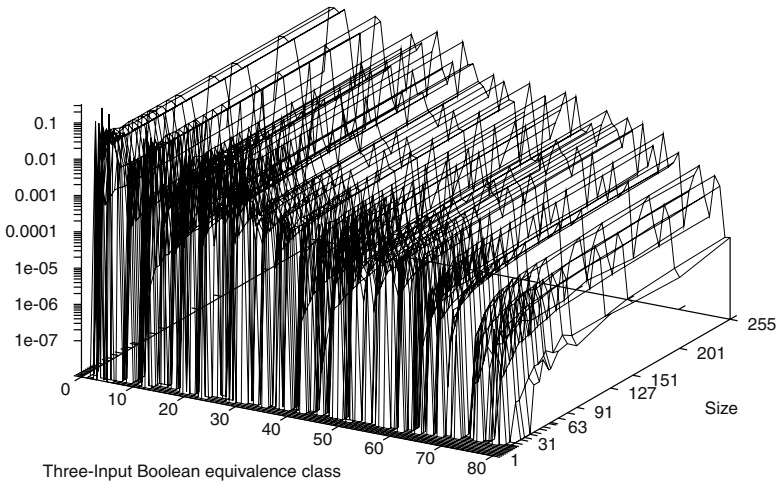


Fig. 20. Proportion of NAND-trees that yield each three-input functions; as circuit size increases the distribution approaches a limit

Notice how, as the length of programs increases, the proportion of programs implementing each function approaches a limit. This does not happen by accident. There is a very substantial body of empirical evidence indicating that this happens in a variety of other systems. In fact, we have also been able to prove mathematically these convergence results for two important forms of programs: Lisp (tree-like) S-expressions (without side effects) and machine code programs without loops [207–210, 212, 222]. Also, similar results were derived for: (a) cyclic (increment, decrement and NOP), (b) bit flip computer, (flip bit and NOP), (c) any non-reversible computer, (d) any reversible computer, (e) CCNOT (Toffoli gate) computer, (f) quantum computers, (g) the ‘average’ computer and h) AND, NAND, OR, NOR expressions (however, these are not Turing complete).

Recently, we started extending our results to Turing complete machine code programs [304]. We considered a simple but realistic Turing complete machine code language, T7. It includes: directly accessed bit addressable memory, an addition operator, an unconditional jump, a conditional branch and four copy instructions. We performed a mathematical analysis of the halting process based on a Markov chain model of program execution and halting. The model can be used to estimate, for any given program length, important quantities, such as the halting probability and the run time of halting programs. This showed a scaling law indicating that the halting probability for programs of length L is of order $1/\sqrt{L}$, while the expected number of instructions executed by halting programs is of order \sqrt{L} . In contrast to many proposed Markov models, this can be done very efficiently, making it possible to compute these quantities for programs of tens of million instructions in a few minutes. Experimental results confirmed the theory.

9.3 Bloat

There are a certain number of limits in GP: bloat, limited modularity of evolved solutions and limited scalability of GP as the problem size increases. We briefly discuss the main one, bloat, below.

Starting in the early 1990s, researchers began to notice that in addition to progressively increasing their mean and best fitness, GP populations also showed certain other dynamics. In particular, it was noted that very often the average size (number of nodes) of the programs in a population, after a certain number of generations in which it was largely static, at some point would start growing at a rapid pace. Typically the increase in program size was not accompanied by any corresponding increase in fitness. The origin of this phenomenon, which is known as *bloat*, has effectively been a mystery for over a decade.

Note that there are situations where one would expect to see program growth as part of the process of solving a problem. For example, GP runs typically start from populations of small random programs, and it may be necessary for the programs to grow in complexity for them to be able to comply with all the fitness cases (a situation which often arises in continuous symbolic regression problems). So, we should not equate bloat with growth. We should only talk of bloat when there is growth without (significant) return in terms of fitness.

Because of its surprising nature and of its practical effects (large programs are hard to interpret, may have poor generalization and are computationally expensive to evolve and later use), bloat has been a subject of intense study in GP. As a result, many theories have been proposed to explain bloat: replication accuracy theory, removal bias theory, nature of program search spaces theory, and so on. Unfortunately, only recently we have started understanding the deep reasons for bloat. So, there is a great deal of confusion in the field as to the reasons of (and the remedies for) bloat. For many people bloat is still a puzzle.

Let us briefly review these theories:

Replication accuracy theory [252]: This theory states that the success of a GP individual depends on its ability to have offspring that are functionally similar to the parent. So, GP evolves towards (bloated) representations that increase replication accuracy.

Removal bias theory [356]: ‘Inactive code’ (code that is not executed, or is executed but its output is then discarded) in a GP tree is low in the tree, forming smaller-than-average-size subtrees. Crossover events excising inactive subtrees produce offspring with the same fitness as their parents. On average the inserted subtree is bigger than the excised one, so such offspring are bigger than average.

Nature of program search spaces theory [221, 225]: Above a certain size, the distribution of fitnesses does not vary with size. Since there are more long programs, the number of long programs of a given fitness is greater than the number of short programs of the same fitness. Over time GP samples longer and longer programs simply because there are more of them.

Crossover bias theory [81, 312]: On average, each application of subtree crossover removes as much genetic material as it inserts. So, crossover in itself does not produce growth or shrinkage. However, while the mean program size is unaffected, other moments of the distribution are. In particular, we know that crossover pushes the population towards a particular distribution of program sizes (a Lagrange distribution of the second kind), where small programs have a much higher frequency than longer ones. For example, crossover generates a very high proportion of single-node individuals. In virtually all problems of practical interest, very small programs have no chance of solving the problem. As a result, programs of above average length have a selective advantage over programs of below average length. Consequently, the mean program size increases.

Several effective techniques to control bloat have been proposed [225, 355]. For example, size fair crossover or size fair mutation [64, 206], Tarpeian bloat control [296], parsimony pressure [416–418], or using many runs each lasting only a few generations. Generally the use of multiple genetic operations, each making a small change, seems to help [11, 281]. There are also several mutation operators that may help control the average tree size in the population while still introducing new genetic material. [178] proposes a mutation operator which prevents the offspring's depth being more than 15% larger than its parent. [202] proposes two mutation operators in which the new random subtree is on average the same size as the code it replaces. In Hoist mutation [180] the new subtree is selected from the subtree being removed from the parent, guaranteeing that the new program will be smaller than its parent. Shrink mutation [9] is a special case of subtree mutation where the randomly chosen subtree is replaced by a randomly chosen terminal.

10 Conclusions

In his seminal 1948 paper entitled 'Intelligent Machinery', Turing identified three ways by which human-competitive machine intelligence might be achieved. In connection with one of those ways, Turing said:

“There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being the survival value.” [384]

Turing did not specify how to conduct the 'genetical or evolutionary search' for machine intelligence. In particular, he did not mention the idea of a population-based parallel search in conjunction with sexual recombination

(crossover) as described in John Holland's 1975 book [146]. However, in his 1950 paper 'Computing Machinery and Intelligence', he did point out:

"We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution:

'Structure of the child machine' = Hereditary material
 'Changes of the child machine' = Mutations
 'Natural selection' = Judgement of the experimenter" [385]

In other words, Turing perceived that one possibly productive approach to machine intelligence would involve an evolutionary process in which a description of a computer program (the hereditary material) undergoes progressive modification (mutation) under the guidance of natural selection (that is, selective pressure in the form of what we now call 'fitness').

Today, many decades later, we can see that indeed Turing was right. GP has started fulfilling Turing's dream by providing us with a systematic method, based on Darwinian evolution, for getting computers to automatically solve hard real-life problems. To do so, it simply requires a high-level statement of what needs to be done (and enough computing power).

Turing also understood the need to evaluate objectively the behavior exhibited by machines, to avoid human biases when assessing their intelligence. This led him to propose an imitation game, now known as the *Turing test for machine intelligence*, whose goals are wonderfully summarized by Samuel's position statement quoted in the introduction of this chapter. The eight criteria for human competitiveness we discussed in Sect. 7.2 are motivated by the same goals.

At present GP is unable to produce computer programs that would pass the full Turing test for machine intelligence, and it might not be ready for this immense task for centuries. Nonetheless, thanks to the constant improvements in GP technology, in its theoretical foundations and in computing power, GP has been able to solve tens of difficult problems with human-competitive results (see Sect. 7.2). These are a small step towards fulfilling Turing and Samuel's dreams, but they are also early signs of things to come. It is, indeed, arguable that in a few years' time GP will be able to *routinely* and *competently* solve important problems for us in a variety of application domains with human-competitive performance. Genetic programming will then become an essential collaborator for many human activities. This, we believe, will be a remarkable step forward towards achieving true, human-competitive machine intelligence.

Acknowledgements

Some of the material in this book chapter has previously appeared in a more extended form in R. Poli, W.B. Langdon, N.F. McPhee, *A Field Guide to Genetic Programming*, lulu.com, 2008. Permission to reproduce it here has been granted by the copyright holders. We would like to thank Rick Riolo, Matthew Walker, Christian Gagne, Bob McKay, Giovanni Paziienza and Lee Spector for their timely assistance.

References

1. Al-Sakran SH, Koza JR, Jones LW (2005) Automated re-invention of a previously patented optical lens system using genetic programming. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds) Proceedings of the 8th European Conference on Genetic Programming, Springer, Lausanne, Switzerland, Lecture Notes in Computer Science, vol 3447, pp 25–37, URL <http://springerlink.metapress.com/openurl.asp?genre=article&i%ssn=0302-9743&volume=3447&spage=25>
2. Allen J, Davey HM, Broadhurst D, Heald JK, Rowland JJ, Oliver SG, Kell DB (2003) High-throughput classification of yeast mutants for functional genomics using metabolic footprinting. *Nature Biotechnology* 21(6):692–696, DOI doi:10.1038/nbt823, URL [http://dbkgroup.org/Papers/NatureBiotechnology21\(692-696\).pdf](http://dbkgroup.org/Papers/NatureBiotechnology21(692-696).pdf)
3. Altenberg L (1994) Emergent phenomena in genetic programming. In: Sebald AV, Fogel LJ (eds) *Evolutionary Programming—Proceedings of the Third Annual Conference*, World Scientific Publishing, San Diego, CA, USA, pp 233–241, URL <http://dynamics.org/~altenber/PAPERS/EPIGP/>
4. Alves da Silva AP, Abrao PJ (2002) Applications of evolutionary computation in electric power systems. In: Fogel DB, El-Sharkawi MA, Yao X, Greenwood G, Iba H, Marrow P, Shackleton M (eds) *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, IEEE Press, pp 1057–1062, DOI doi:10.1109/CEC.2002.1004389
5. Ando D, Dahlsted P, Nordahl M, Iba H (2007) Interactive GP with tree representation of classical music pieces. In: Giacobini M, Brabazon A, Cagnoni S, Di Caro GA, Drechsler R, Farooq M, Fink A, Lutton E, Machado P, Minner S, O’Neill M, Romero J, Rothlauf F, Squillero G, Takagi H, Uyar AS, Yang S (eds) *Applications of Evolutionary Computing, EvoWorkshops 2007: EvoCOMNET, EvoFIN, EvoIASP, EvoInteraction, EvoMUSART, EvoSTOC, EvoTransLog*, Springer Verlag, Valencia, Spain, LNCS, vol 4448, pp 577–584, DOI doi:10.1007/978-3-540-71805-5_63
6. Andre D, Koza JR (1996) Parallel genetic programming: A scalable implementation using the transputer network architecture. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 16, pp 317–338
7. Andre D, Koza JR (1998) A parallel implementation of genetic programming that achieves super-linear performance. *Information Sciences* 106(3–4):201–218, URL <http://www.sciencedirect.com/science/article/B6V0C-3TKS65B-21/2/22b9842f820b08883990bbae1d889c03>

8. Andre D, Bennett III FH, Koza JR (1996) Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) Genetic Programming 1996: Proceedings of the First Annual Conference, MIT Press, Stanford University, CA, USA, pp 3–11, URL <http://www.genetic-programming.com/jkpdf/gp1996gkl.pdf>
9. Angeline PJ (1996) An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) Genetic Programming 1996: Proceedings of the First Annual Conference, MIT Press, Stanford University, CA, USA, pp 21–29, URL <http://www.natural-selection.com/Library/1996/gp96.zip>
10. Angeline PJ (1997) Subtree crossover: Building block engine or macromutation? In: Koza JR, Deb K, Dorigo M, Fogel DB, Garzon M, Iba H, Riolo RL (eds) Genetic Programming 1997: Proceedings of the Second Annual Conference, Morgan Kaufmann, Stanford University, CA, USA, pp 9–17
11. Angeline PJ (1998) Multiple interacting programs: A representation for evolving complex behaviors. *Cybernetics and Systems* 29(8):779–806, URL <http://www.natural-selection.com/Library/1998/mips3.pdf>
12. Angeline PJ, Kinnear, Jr KE (eds) (1996) *Advances in Genetic Programming 2*. MIT Press, Cambridge, MA, USA, URL <http://www.cs.bham.ac.uk/~wbl/aigp2.html>
13. Angeline PJ, Pollack JB (1992) The evolutionary induction of subroutines. In: Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, Lawrence Erlbaum, Bloomington, Indiana, USA, pp 236–241, URL <http://www.demo.cs.brandeis.edu/papers/glib92.pdf>
14. Arkov V, Evans C, Fleming PJ, Hill DC, Norton JP, Pratt I, Rees D, Rodriguez-Vazquez K (2000) System identification strategies applied to aircraft gas turbine engines. *Annual Reviews in Control* 24(1):67–81, URL <http://www.sciencedirect.com/science/article/B6V0H-482MDPD-8/2/dd470648e2228c84efe7e14ca3841b7e>
15. Austin MP, Bates G, Dempster MAH, Leemans V, Williams SN (2004) Adaptive systems for foreign exchange trading. *Quantitative Finance* 4(4):37–45, DOI doi:10.1080/14697680400008593, URL <http://www.cfr.jbs.cam.ac.uk/archive/PRESENTATIONS/seminars/2006/dempster2.pdf>
16. Azaria Y, Sipper M (2005a) GP-gammon: Genetically programming backgammon players. *Genetic Programming and Evolvable Machines* 6(3):283–300, DOI doi:10.1007/s10710-005-2990-0, URL <http://www.cs.bgu.ac.il/~sipper/papabs/gpgammon.pdf>, published online: 12 August 2005
17. Azaria Y, Sipper M (2005b) Using GP-gammon: Using genetic programming to evolve backgammon players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds) Proceedings of the 8th European Conference on Genetic Programming, Springer, Lausanne, Switzerland, Lecture Notes in Computer Science, vol 3447, pp 132–142, URL <http://springerlink.metapress.com/openurl.asp?genre=article&iissn=0302-9743&volume=3447&spage=132>
18. Babovic V (1996) Emergence, evolution, intelligence; Hydroinformatics - A study of distributed and decentralised computing using intelligent agents. A. A. Balkema Publishers, Rotterdam, Holland
19. Bader-El-Den M, Poli R (2007a) Generating sat local-search heuristics using a gp hyper-heuristic framework. In: Proceedings of Evolution Artificielle

20. Bader-El-Den MB, Poli R (2007b) A GP-based hyper-heuristic framework for evolving 3-SAT heuristics. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM Press, London, vol 2, pp 1749–1749, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1749.pdf>
21. Bains W, Gilbert R, Sviridenko L, Gascon JM, Scoffin R, Birchall K, Harvey I, Caldwell J (2002) Evolutionary computational methods to predict oral bioavailability QSPRs. *Current Opinion in Drug Discovery and Development* 5(1):44–51
22. Baker JE (1987) Reducing bias and inefficiency in the selection algorithm. In: Grefenstette JJ (ed) Proceedings of the Second International Conference on Genetic Algorithms and their Application, Lawrence Erlbaum Associates, Cambridge, MA, USA, pp 14–21
23. Balic J (1999) Flexible Manufacturing Systems; Development - Structure - Operation - Handling - Tooling. Manufacturing technology, DAAAM International, Vienna
24. Banzhaf W (1993) Genetic programming for pedestrians. In: Forrest S (ed) Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, Morgan Kaufmann, University of Illinois at Urbana-Champaign, p 628, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/GenProg_forPed.ps.Z
25. Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann, San Francisco, CA, USA
26. Barrett SJ (2003) Recurring analytical problems within drug discovery and development. In: Scheffer T, Leser U (eds) Data Mining and Text Mining for Bioinformatics: Proceedings of the European Workshop, Dubrovnik, Croatia, pp 6–7, URL <http://www2.informatik.hu-berlin.de/~scheffer/publications/ProceedingsWS2003.pdf>, invited talk
27. Barrett SJ, Langdon WB (2006) Advances in the application of machine learning techniques in drug discovery, design and development. In: Tiwari A, Knowles J, Avineri E, Dahal K, Roy R (eds) Applications of Soft Computing: Recent Trends, Springer, On the World Wide Web, Advances in Soft Computing, pp 99–110, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/barrett_2005_WSC.pdf
28. Bennett III FH (1996) Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) Genetic Programming 1996: Proceedings of the First Annual Conference, MIT Press, Stanford University, CA, USA, pp 30–38, URL <http://cognet.mit.edu/library/books/view?isbn=0262611279>
29. Bennett III FH, Koza JR, Shipman J, Stiffelman O (1999) Building a parallel computer system for \$18,000 that performs a half peta-flop per day. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, Orlando, Florida, USA, vol 2, pp 1484–1490, URL <http://www.genetic-programming.com/jkpdf/gecco1999beowulf.pdf>

30. Bhanu B, Lin Y, Krawiec K (2005) Evolutionary Synthesis of Pattern Recognition Systems. Monographs in Computer Science, Springer-Verlag, New York, URL <http://www.springer.com/west/home/computer/imaging?SGWID=4-14%9-22-39144807-detailsPage=ppmmedia—aboutThisBook>
31. Blickle T (1996) Theory of evolutionary algorithms and application to system synthesis. PhD thesis, Swiss Federal Institute of Technology, Zurich, URL <http://www.handshake.de/user/blickle/publications/diss.pdf>
32. Brabazon A, O'Neill M (2006) Biologically Inspired Algorithms for Financial Modeling. Natural Computing Series, Springer
33. Brameier M, Banzhaf W (2001) A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation* 5(1):17–26, URL http://web.cs.mun.ca/~banzhaf/papers/ieee_taec.pdf
34. Brameier M, Banzhaf W (2007) Linear Genetic Programming. No. XVI in Genetic and Evolutionary Computation, Springer, URL <http://www.springer.com/west/home/default?SGWID=4-40356-22-173660820-0>
35. Brameier M, Haan J, Krings A, MacCallum RM (2006) Automatic discovery of cross-family sequence features associated with protein function. *BMC bioinformatics [electronic resource]* 7(16), DOI doi:10.1186/1471-2105-7-16, URL <http://www.biomedcentral.com/content/pdf/1471-2105-7-16.pdf>
36. Brave S (1996) Evolving recursive programs for tree search. In: Angeline PJ, Kinneer, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 10, pp 203–220
37. Brezocnik M (2000) *Uporaba genetskega programiranja v inteligentnih proizvodnih sistemih*. University of Maribor, Faculty of mechanical engineering, Maribor, Slovenia, URL <http://maja.uni-mb.si/slo/Knjige/2000-03-mon/index.htm>
38. Brezocnik M, Balic J, Gusel L (2000) Artificial intelligence approach to determination of flow curve. *Journal for technology of plasticity* 25(1–2):1–7
39. Buason G, Bergfeldt N, Ziemke T (2005) Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and environments. *Genetic Programming and Evolvable Machines* 6(1):25–51, DOI doi:10.1007/s10710-005-7618-x
40. Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyperheuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) *Handbook of Metaheuristics*, Kluwer Academic Publishers, pp 457–474
41. Burke EK, Hyde MR, Kendall G (2006) Evolving bin packing heuristics with genetic programming. In: Runarsson TP, Beyer HG, Burke E, Merelo-Guervos JJ, Whitley LD, Yao X (eds) *Parallel Problem Solving from Nature - PPSN IX*, Springer-Verlag, Reykjavik, Iceland, LNCS, vol 4193, pp 860–869, DOI doi:10.1007/11844297_87, URL <http://www.cs.nott.ac.uk/~mvh/ppsn2006.pdf>
42. Burke EK, Hyde MR, Kendall G, Woodward J (2007) Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM Press, London, vol 2, pp 1559–1565, URL <http://www.cs.bham.ac.uk/wbl/biblio/gecco2007/docs/p1559.pdf>

43. Buxton BF, Langdon WB, Barrett SJ (2001) Data fusion by intelligent classifier combination. *Measurement and Control* 34(8):229–234, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/mc/>
44. Cagnoni S, Bergenti F, Mordonini M, Adorni G (2005) Evolving binary classifiers through parallel computation of multiple fitness cases. *IEEE Transactions on Systems, Man and Cybernetics - Part B* 35(3):548–555, DOI doi:10.1109/TSMCB.2005.846671
45. Cai W, Pacheco-Vega A, Sen M, Yang KT (2006) Heat transfer correlations by symbolic regression. *International Journal of Heat and Mass Transfer* 49(23–24):4352–4359, DOI doi:10.1016/j.ijheatmasstransfer.2006.04.029
46. Castillo F, Kordon A, Smits G (2006) Robust pareto front genetic programming parameter selection based on design of experiments and industrial data. In: Riolo RL, Soule T, Worzel B (eds) *Genetic Programming Theory and Practice IV, Genetic and Evolutionary Computation*, vol 5, Springer, Ann Arbor
47. Chami M, Robilliard D (2002) Inversion of oceanic constituents in case I and II waters with genetic programming algorithms. *Applied Optics* 41(30):6260–6275, URL <http://ao.osa.org/ViewMedia.cfm?id=70258&seq=0>
48. Channon A (2006) Unbounded evolutionary dynamics in a system of agents that actively process and transform their environment. *Genetic Programming and Evolvable Machines* 7(3):253–281, DOI doi:10.1007/s10710-006-9009-3
49. Chao DL, Forrest S (2003) Information immune systems. *Genetic Programming and Evolvable Machines* 4(4):311–331, DOI doi:10.1023/A:1026139027539
50. Cheang SM, Leung KS, Lee KH (2006) Genetic parallel programming: Design and implementation. *Evolutionary Computation* 14(2):129–156, DOI doi:10.1162/evco.2006.14.2.129
51. Chen SH (ed) (2002) *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer Academic Publishers, Dordrecht, URL <http://www.springer.com/west/home/business?SGWID=4-40517-22-3%3195998-detailsPage=ppmmedia|toc>
52. Chen SH, Liao CC (2005) Agent-based computational modeling of the stock price-volume relation. *Information Sciences* 170(1):75–100, DOI doi:10.1016/j.ins.2003.03.026, URL <http://www.sciencedirect.com/science/article/B6V0C-4B3JHTS-6/2/9e023835b1c70f176d1903dd3a8b638e>
53. Chen SH, Wang HS, Zhang BT (1999) Forecasting high-frequency financial time series with evolutionary neural trees: The case of heng-sheng stock index. In: Arabnia HR (ed) *Proceedings of the International Conference on Artificial Intelligence, IC-AI '99*, CSREA Press, Las Vegas, Nevada, USA, vol 2, pp 437–443, URL <http://bi.snu.ac.kr/Publications/Conferences/International/ICAI99.ps>
54. Chen SH, Duffy J, Yeh CH (2002) Equilibrium selection via adaptation: Using genetic programming to model learning in a coordination game. *The Electronic Journal of Evolutionary Modeling and Economic Dynamics*
55. Chitty DM (2007) A data parallel approach to genetic programming using programmable graphics hardware. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM Press, London, vol 2, pp 1566–1573, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1566.pdf>

56. Chong FS, Langdon WB (1999) Java based distributed genetic programming on the internet. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, Orlando, Florida, USA, vol 2, p 1229, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/p.chong/DGPposter.pdf>, full text in technical report CSRP-99-7
57. Ciesielski V, Li X (2004) Analysis of genetic programming runs. In: Mckay RI, Cho SB (eds) Proceedings of The Second Asian-Pacific Workshop on Genetic Programming, Cairns, Australia, URL <http://goanna.cs.rmit.edu.au/~xiali/pub/ai04.vc.pdf>
58. Cilibrasi R, Vitanyi PMB (2005) Clustering by compression. *IEEE Transactions on Information Theory* 51(4):1523–1545, URL <http://homepages.cwi.nl/~paulv/papers/cluster.pdf>
59. Cilibrasi R, Vitanyi P, de Wolf R (2004) Algorithmic clustering of music based on string compression. *Computer Music Journal* 28(4):49–67, URL <http://homepages.cwi.nl/~paulv/papers/music.pdf>
60. Collins RJ (1992) Studies in artificial evolution. PhD thesis, UCLA, Artificial Life Laboratory, Department of Computer Science, University of California, Los Angeles, LA CA 90024, USA
61. Corno F, Sanchez E, Squillero G (2005) Evolving assembly programs: how games help microprocessor validation. *Evolutionary Computation, IEEE Transactions on* 9(6):695–706
62. Costelloe D, Ryan C (2007) Towards models of user preferences in interactive musical evolution. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM Press, London, vol 2, pp 2254–2254, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p2254.pdf>
63. Cranmer K, Bowman RS (2005) PhysicsGP: A genetic programming approach to event selection. *Computer Physics Communications* 167(3):165–176, DOI doi:10.1016/j.cpc.2004.12.006
64. Crawford-Marks R, Spector L (2002) Size control via size fair genetic operators in the PushGP genetic programming system. In: Langdon WB, Cantú-Paz E, Mathias K, Roy R, Davis D, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke E, Jonoska N (eds) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, New York, pp 733–739, URL <http://alum.hampshire.edu/~rpc01/gp234.pdf>
65. Crepeau RL (1995) Genetic evolution of machine language software. In: Rosca JP (ed) Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, Tahoe City, California, USA, pp 121–134, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/GEMS_Article.pdf
66. Curry R, Lichodziejewski P, Heywood MI (2007) Scaling genetic programming to large datasets using hierarchical dynamic subset selection. *IEEE Transactions on Systems, Man, and Cybernetics: Part B - Cybernetics* 37(4):1065–1073, DOI doi:10.1109/TSMCB.2007.896406, URL <http://www.cs.dal.ca/~mheywood/X-files/GradPubs.html#curry>

67. Daida JM, Hommes JD, Bersano-Begey TF, Ross SJ, Vesecky JF (1996) Algorithm discovery using the genetic programming paradigm: Extracting low-contrast curvilinear features from SAR images of arctic ice. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 21, pp 417–442, URL http://sitemaker.umich.edu/daida/files/GP2_cha21.pdf
68. Dassau E, Grosman B, Lewin DR (2006) Modeling and temperature control of rapid thermal processing. *Computers and Chemical Engineering* 30(4):686–697, DOI doi:10.1016/j.compchemeng.2005.11.007, URL http://tx.technion.ac.il/~dlewin/publications/rtp_paper_v9.pdf
69. Davis TE, Principe JC (1993) A Markov chain framework for the simple genetic algorithm. *Evolutionary Computation* 1(3):269–288
70. Day JP, Kell DB, Griffith GW (2002) Differentiation of phytophthora infestans sporangia from other airborne biological particles by flow cytometry. *Applied and Environmental Microbiology* 68(1):37–45, DOI doi:10.1128/AEM.68.1.37-45.2002, URL <http://intl-aem.asm.org/cgi/reprint/68/1/37.pdf>
71. de Sousa JS, de CT Gomes L, Bezerra GB, de Castro LN, Von Zuben FJ (2004) An immune-evolutionary algorithm for multiple rearrangements of gene expression data. *Genetic Programming and Evolvable Machines* 5(2):157–179, DOI doi:10.1023/B:GENP.0000023686.59617.57
72. De Stefano C, Cioppa AD, Marcelli A (2002) Character preclassification based on genetic programming. *Pattern Recognition Letters* 23(12):1439–1448, DOI doi:10.1016/S0167-8655(02)00104-6, URL <http://www.sciencedirect.com/science/article/B6V15-45J91MV-4/2/3e5c2ac0c51428d0f7ea9fc0142f6790>
73. Deb K (2001) *Multi-objective optimization using evolutionary algorithms*. Wiley
74. Dempster MAH, Jones CM (2000) A real-time adaptive trading system using genetic programming. *Quantitative Finance* 1:397–413, URL <http://mahd-pc.jbs.cam.ac.uk/archive/PAPERS/2000/geneticprogramming.pdf>
75. Dempster MAH, Payne TW, Romahi Y, Thompson GWP (2001) Computational learning techniques for intraday FX trading using popular technical indicators. *IEEE Transactions on Neural Networks* 12(4):744–754, DOI doi:10.1109/72.935088, URL <http://mahd-pc.jbs.cam.ac.uk/archive/PAPERS/2000/ieeetrading.pdf>
76. Deschaine L (2006) Using information fusion, machine learning, and global optimisation to increase the accuracy of finding and understanding items interest in the subsurface. *GeoDrilling International* (122):30–32, URL http://www.mining-journal.com/gdi_magazine/pdf/GDI0605scr.pdf
77. Deschaine LM, Patel JJ, Guthrie RD, Grimski JT, Ades MJ (2001) Using linear genetic programming to develop a C/C++ simulation model of a waste incinerator. In: Ades M (ed) *Advanced Technology Simulation Conference*, Seattle, URL <http://www.aimlearning.com/Environmental.Engineering.pdf>
78. Deschaine LM, Hoover RA, Skibinski JN, Patel JJ, Francone F, Nordin P, Ades MJ (2002) Using machine learning to compliment and extend the accuracy of UXO discrimination beyond the best reported results of the jefferson proving ground technology demonstration. In: *2002 Advanced Technology Simulation Conference*, San Diego, CA, USA, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/deschaine/ASTC_2002_UXO_Finder_Invention_Paper.pdf

79. D'haeseleer P (1994) Context preserving crossover in genetic programming. In: Proceedings of the 1994 IEEE World Congress on Computational Intelligence, IEEE Press, Orlando, Florida, USA, vol 1, pp 256–261, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/WCCI94.CPC.ps.Z>
80. D'haeseleer P, Bluming J (1994) Effects of locality in individual and population evolution. In: Kinnear, Jr KE (ed) *Advances in Genetic Programming*, MIT Press, chap 8, pp 177–198, URL <http://cognet.mit.edu/library/books/view?isbn=0262111888>
81. Dignum S, Poli R (2007) Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM Press, London, vol 2, pp 1588–1595, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1588.pdf>
82. Dolinsky JU, Jenkinson ID, Colquhoun GJ (2007) Application of genetic programming to the calibration of industrial robots. *Computers in Industry* 58(3):255–264, DOI doi:10.1016/j.compind.2006.06.003
83. Domingos RP, Schirru R, Martinez AS (2005) Soft computing systems applied to PWR's xenon. *Progress in Nuclear Energy* 46(3–4):297–308, DOI doi:10.1016/j.pnucene.2005.03.011
84. Dracopoulos DC (1997) *Evolutionary Learning Algorithms for Neural Adaptive Control. Perspectives in Neural Computing*, Springer Verlag, P.O. Box 31 13 40, D-10643 Berlin, Germany, URL <http://www.springer.de/catalog/html-files/deutsch/comp/3540761616.html>
85. Droste S, Jansen T, Rudolph G, Schwefel HP, Tinnefeld K, Wegener I (2003) Theory of evolutionary algorithms and genetic programming. In: Schwefel HP, Wegener I, Weinert K (eds) *Advances in Computational Intelligence: Theory and Practice*, Natural Computing Series, Springer, chap 5, pp 107–144
86. Ebner M, Reinhardt M, Albert J (2005) Evolution of vertex and pixel shaders. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds) *Proceedings of the 8th European Conference on Genetic Programming*, Springer, Lausanne, Switzerland, Lecture Notes in Computer Science, vol 3447, pp 261–270, DOI doi:10.1007/b107383, URL <http://springerlink.metapress.com/openurl.asp?genre=article&zissn=0302-9743&volume=3447&spage=261>
87. Eiben AE, Smith JE (2003) *Introduction to Evolutionary Computing*. Springer, URL <http://www.cs.vu.nl/~gusz/ecbook/ecbook.html>
88. Eklund SE (2002) A massively parallel GP engine in VLSI. In: Fogel DB, El-Sharkawi MA, Yao X, Greenwood G, Iba H, Marrow P, Shackleton M (eds) *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, IEEE Press, pp 629–633
89. Ellis DI, Broadhurst D, Kell DB, Rowland JJ, Goodacre R (2002) Rapid and quantitative detection of the microbial spoilage of meat by fourier transform infrared spectroscopy and machine learning. *Applied and Environmental Microbiology* 68(6):2822–2828, DOI doi:10.1128/AEM.68.6.2822-2828.2002, URL [http://dbkgroup.org/Papers/app_%20env_microbiol_68_\(2822\).pdf](http://dbkgroup.org/Papers/app_%20env_microbiol_68_(2822).pdf)
90. Ellis DI, Broadhurst D, Goodacre R (2004) Rapid and quantitative detection of the microbial spoilage of beef by fourier transform infrared spectroscopy and

- machine learning. *Analytica Chimica Acta* 514(2):193–201, DOI doi:10.1016/j.aca.2004.03.060, URL http://dbkgroup.org/dave_files/ACAbef04.pdf
91. Eriksson R, Olsson B (2004) Adapting genetic regulatory models by genetic programming. *Biosystems* 76(1–3):217–227, DOI doi:10.1016/j.biosystems.2004.05.014, URL <http://www.sciencedirect.com/science/article/B6T2K-4D09KY2-7/2/1abfe196bb4afc60afc3311cadb75d66>
 92. Esparcia-Alcazar AI, Sharman KC (1996) Genetic programming techniques that evolve recurrent neural networks architectures for signal processing. In: *IEEE Workshop on Neural Networks for Signal Processing*, Seiko, Kyoto, Japan
 93. Evans C, Fleming PJ, Hill DC, Norton JP, Pratt I, Rees D, Rodriguez-Vazquez K (2001) Application of system identification techniques to aircraft gas turbine engines. *Control Engineering Practice* 9(2):135–148, URL <http://www.sciencedirect.com/science/article/B6V2H-4280YP2-3/1/24d44180070f91dea854032d98f9187a>
 94. Federman F, Sparkman G, Watt S (1999) Representation of music in a learning classifier system utilizing bach chorales. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, vol 1, p 785
 95. Felton MJ (2000) Survival of the fittest in drug design. *Modern Drug Discovery* 3(9):49–50, URL <http://pubs.acs.org/subscribe/journals/mdd/v03/i09/html/felton.html>
 96. Fernandez F, Sanchez JM, Tomassini M, Gomez JA (1999) A parallel genetic programming tool based on PVM. In: Dongarra J, Luque E, Margalef T (eds) *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Proceedings of the 6th European PVM/MPI Users' Group Meeting, Springer-Verlag, Barcelona, Spain, *Lecture Notes in Computer Science*, vol 1697, pp 241–248
 97. Fernandez F, Tomassini M, Vanneschi L (2003) An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines* 4(1):21–51, DOI doi:10.1023/A:1021873026259
 98. Folino G, Pizzuti C, Spezzano G (2003) A scalable cellular implementation of parallel genetic programming. *IEEE Transactions on Evolutionary Computation* 7(1):37–53
 99. Foster JA (2001) Review: Discipulus: A commercial genetic programming system. *Genetic Programming and Evolvable Machines* 2(2):201–203, DOI doi:10.1023/A:1011516717456
 100. Francone FD, Deschaine LM (2004) Getting it right at the very start – building project models where data is expensive by combining human expertise, machine learning and information theory. In: *2004 Business and Industry Symposium*, Washington, DC, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/deschaine/ASTC_2004_Getting_It_Right_from_the_Very_Start.pdf
 101. Francone FD, Conrads M, Banzhaf W, Nordin P (1999) Homologous crossover in genetic programming. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, vol 2, pp 1021–1026, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GP-463.pdf>

102. Francone FD, Deschaine LM, Warren JJ (2007) Discrimination of munitions and explosives of concern at F.E. warren AFB using linear genetic programming. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM Press, London, vol 2, pp 1999–2006, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1999.pdf>
103. Fukunaga A (2002) Automated discovery of composite SAT variable selection heuristics. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp 641–648
104. Fukunaga AS (2004) Evolving local search heuristics for SAT using genetic programming. In: Deb K, Poli R, Banzhaf W, Beyer HG, Burke E, Darwen P, Dasgupta D, Floreano D, Foster J, Harman M, Holland O, Lanzi PL, Spector L, Tettamanzi A, Thierens D, Tyrrell A (eds) Genetic and Evolutionary Computation – GECCO-2004, Part II, Springer-Verlag, Seattle, WA, USA, Lecture Notes in Computer Science, vol 3103, pp 483–494, DOI doi:10.1007/b98645, URL <http://alex04.maclisp.org/gecco2004.pdf>
105. Funes P, Sklar E, Juille H, Pollack J (1998a) Animal-animat coevolution: Using the animal population as fitness function. In: Pfeifer R, Blumberg B, Meyer JA, Wilson SW (eds) From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior., MIT Press, Zurich, Switzerland, pp 525–533, URL <http://www.demo.cs.brandeis.edu/papers/tronsab98.html>
106. Funes P, Sklar E, Juille H, Pollack J (1998b) Animal-animat coevolution: Using the animal population as fitness function. In: Pfeifer R, Blumberg B, Meyer JA, Wilson SW (eds) From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior, MIT Press, Zurich, Switzerland, pp 525–533, URL <http://www.demo.cs.brandeis.edu/papers/tronsab98.pdf>
107. Gagne C, Parizeau M (2006) Genetic engineering of hierarchical fuzzy regional representations for handwritten character recognition. *International Journal on Document Analysis and Recognition* 8(4):223–231, DOI doi:10.1007/s10032-005-0005-6, URL http://vision.gel.ulaval.ca/fr/publications/Id_607/PublDetails.php
108. Gagné C, Parizeau M (2007) Co-evolution of nearest neighbor classifiers. *International Journal of Pattern Recognition and Artificial Intelligence* 21(5):921–946, DOI doi:10.1142/S0218001407005752, URL http://vision.gel.ulaval.ca/en/publications/Id_692/PublDetails.php
109. Garcia-Almanza AL, Tsang EPK (2006) Forecasting stock prices using genetic programming and chance discovery. In: 12th International Conference On Computing In Economics And Finance, p number 489, URL <http://repec.org/sce2006/up.13879.1141401469.pdf>
110. Gathercole C, Ross P (1994) Dynamic training subset selection for supervised learning in genetic programming. In: Davidor Y, Schwefel HP, Männer R (eds) Parallel Problem Solving from Nature III, Springer-Verlag, Jerusalem, LNCS, vol 866, pp 312–321, URL <http://citeseer.ist.psu.edu/gathercole94dynamic.html>
111. Gathercole C, Ross P (1997) Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In: Koza JR, Deb K, Dorigo

- M, Fogel DB, Garzon M, Iba H, Riolo RL (eds) Genetic Programming 1997: Proceedings of the Second Annual Conference, Morgan Kaufmann, Stanford University, CA, USA, pp 119–127, URL <http://citeseer.ist.psu.edu/79389.html>
112. Gelly S, Teytaud O, Bredeche N, Schoenauer M (2006) Universal consistency and bloat in GP. *Revue d'Intelligence Artificielle* 20(6):805–827, URL <http://hal.inria.fr/docs/00/11/28/40/PDF/riabloat.pdf>, issue on New Methods in Machine Learning. Theory and Applications
 113. Gilbert RJ, Goodacre R, Woodward AM, Kell DB (1997) Genetic programming: A novel method for the quantitative analysis of pyrolysis mass spectral data. *ANALYTICAL CHEMISTRY* 69(21):4381–4389, DOI doi:10.1021/ac970460j, URL <http://pubs.acs.org/journals/ancham/article.cgi/ancham/1997/69/i21/pdf/ac970460j.pdf>
 114. Globus A, Lawton J, Wipke T (1998) Automatic molecular design using evolutionary techniques. In: Globus A, Srivastava D (eds) The Sixth Foresight Conference on Molecular Nanotechnology, Westin Hotel in Santa Clara, CA, USA, URL <http://www.foresight.org/Conferences/MNT6/Papers/Globus/index.html>
 115. Goldberg DE (1989) Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley
 116. Goldberg DE, Kargupta H, Horn J, Cantu-Paz E (1995) Critical deme size for serial and parallel genetic algorithms. Tech. rep., Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois at Urbana-Champaign, Il 61801, USA, illiGAL Report no 95002
 117. Goodacre R (2003) Explanatory analysis of spectroscopic data using machine learning of simple, interpretable rules. *Vibrational Spectroscopy* 32(1):33–45, DOI doi:10.1016/S0924-2031(03)00045-6, URL <http://www.biospec.net/learning/Metab06/Goodacre-FTIRmaps.pdf>, a collection of Papers Presented at Shedding New Light on Disease: Optical Diagnostics for the New Millennium (SPEC 2002) Reims, France 23–27 June 2002
 118. Goodacre R, Gilbert RJ (1999) The detection of caffeine in a variety of beverages using curie-point pyrolysis mass spectrometry and genetic programming. *The Analyst* 124:1069–1074
 119. Goodacre R, Shann B, Gilbert RJ, Timmins EM, McGovern AC, Alsberg BK, Kell DB, Logan NA (2000) The detection of the dipicolinic acid biomarker in bacillus spores using curie-point pyrolysis mass spectrometry and fourier-transform infrared spectroscopy. *Analytical Chemistry* 72(1):119–127, DOI doi:10.1021/ac990661i, URL <http://pubs.acs.org/cgi-bin/article.cgi/ancham/2000/72/i01/html/ac990661i.html>
 120. Goodacre R, Vaidyanathan S, Dunn WB, Harrigan GG, Kell DB (2004) Metabolomics by numbers: acquiring and understanding global metabolite data. *Trends in Biotechnology* 22(5):245–252, DOI doi:10.1016/j.tibtech.2004.03.007, URL [http://dbkgroup.org/Papers/trends%20in%20biotechnology_22_\(24%5\).pdf](http://dbkgroup.org/Papers/trends%20in%20biotechnology_22_(24%5).pdf)
 121. Gruau F (1994a) Genetic micro programming of neural networks. In: Kinnear, Jr KE (ed) *Advances in Genetic Programming*, MIT Press, chap 24, pp 495–518, URL <http://cognet.mit.edu/library/books/view?isbn=0262111888>
 122. Gruau F (1994b) Neural network synthesis using cellular encoding and the genetic algorithm. PhD thesis, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, URL <ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/PhD/PhD1994/PhD1994-01-E.ps.Z>

123. Gruau F (1996) On using syntactic constraints with genetic programming. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 19, pp 377–394
124. Gruau F, Whitley D (1993) Adding learning to the cellular development process: a comparative study. *Evolutionary Computation* 1(3):213–233
125. Gustafson S, Burke EK (2006) The speciating island model: An alternative parallel evolutionary algorithm. *Journal of Parallel and Distributed Computing* 66(8):1025–1036, DOI doi:10.1016/j.jpdc.2006.04.017, parallel Bioinspired Algorithms
126. Gustafson S, Burke EK, Krasnogor N (2005) On improving genetic programming for symbolic regression. In: Corne D, Michalewicz Z, Dorigo M, Eiben G, Fogel D, Fonseca C, Greenwood G, Chen TK, Raidl G, Zalzal A, Lucas S, Paechter B, Willies J, Guervos JJM, Eberbach E, McKay B, Channon A, Tiwari A, Volkert LG, Ashlock D, Schoenauer M (eds) *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, IEEE Press, Edinburgh, UK, vol 1, pp 912–919
127. Hampo RJ, Marko KA (1992) Application of genetic programming to control of vehicle systems. In: *Proceedings of the Intelligent Vehicles '92 Symposium*, june 29 July 1, 1992, Detroit, Mi, USA
128. Handley S (1993) Automatic learning of a detector for alpha-helices in protein sequences via genetic programming. In: Forrest S (ed) *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, Morgan Kaufmann, University of Illinois at Urbana-Champaign, pp 271–278
129. Handley S (1994) On the use of a directed acyclic graph to represent a population of computer programs. In: *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, IEEE Press, Orlando, Florida, USA, vol 1, pp 154–159, DOI doi:10.1109/ICEC.1994.350024
130. Harding S, Banzhaf W (2007) Fast genetic programming on GPUs. In: Ebner M, O'Neill M, Ekárt A, Vanneschi L, Esparcia-Alcázar AI (eds) *Proceedings of the 10th European Conference on Genetic Programming*, Springer, Valencia, Spain, *Lecture Notes in Computer Science*, vol 4445, pp 90–101, DOI doi:10.1007/978-3-540-71605-1_9
131. Harrigan GG, LaPlante RH, Cosma GN, Cockerell G, Goodacre R, Maddox JF, Luyendyk JP, Ganey PE, Roth RA (2004) Application of high-throughput fourier-transform infrared spectroscopy in toxicology studies: contribution to a study on the development of an animal model for idiosyncratic toxicity. *Toxicology Letters* 146(3):197–205, DOI doi:10.1016/j.toxlet.2003.09.011
132. Harris C, Buxton B (1996) GP-COM: A distributed, component-based genetic programming system in C++. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, Stanford University, CA, USA, p 425, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/gp96com.ps.gz>
133. Harvey B, Foster J, Frincke D (1999) Towards byte code genetic programming. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, vol 2, p 1234, URL <http://citeseer.ist.psu.edu/468509.html>
134. Hasan S, Daugelat S, Rao PSS, Schreiber M (2006) Prioritizing genomic drug targets in pathogens: Application to mycobacterium tuberculosis. *PLoS Computational Biology* 2(6):e61, DOI doi:10.1371/journal.pcbi.0020061

135. Hauptman A, Sipper M (2005) GP-endchess: Using genetic programming to evolve chess endgame players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds) Proceedings of the 8th European Conference on Genetic Programming, Springer, Lausanne, Switzerland, Lecture Notes in Computer Science, vol 3447, pp 120–131, URL <http://www.cs.bgu.ac.il/~sipper/papabs/eurogpchess-final.pdf>
136. Hauptman A, Sipper M (2007) Evolution of an efficient search algorithm for the mate-in-N problem in chess. In: Ebner M, O’Neill M, Ekárt A, Vanneschi L, Esparcia-Alcázar AI (eds) Proceedings of the 10th European Conference on Genetic Programming, Springer, Valencia, Spain, Lecture Notes in Computer Science, vol 4445, pp 78–89, DOI doi:10.1007/978-3-540-71605-1_8
137. Haynes T, Wainwright R, Sen S, Schoenefeld D (1995) Strongly typed genetic programming in evolving cooperation strategies. In: Eshelman L (ed) Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95), Morgan Kaufmann, Pittsburgh, PA, USA, pp 271–278, URL <http://www.mcs.utulsa.edu/~rogerw/papers/Haynes-icga95.pdf>
138. Haynes TD, Schoenefeld DA, Wainwright RL (1996) Type inheritance in strongly typed genetic programming. In: Angeline PJ, Kinnear, Jr KE (eds) Advances in Genetic Programming 2, MIT Press, Cambridge, MA, USA, chap 18, pp 359–376, URL <http://www.mcs.utulsa.edu/~rogerw/papers/Haynes-hier.pdf>
139. Heidema AG, Boer JMA, Nagelkerke N, Mariman ECM, van der A DL, Feskens EJM (2006) The challenge for genetic epidemiologists: how to analyze large numbers of SNPs in relation to complex diseases. *BMC Genetics* 7(23), DOI doi:10.1186/1471-2156-7-23, URL <http://www.biomedcentral.com/content/pdf/1471-2156-7-23.pdf>
140. Hillis WD (1992) Co-evolving parasites improve simulated evolution as an optimization procedure. In: Langton CG, Taylor CE, Farmer JD, Rasmussen S (eds) Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity, vol X, Addison-Wesley, Santa Fe Institute, New Mexico, USA, pp 313–324
141. Hinchliffe MP, Willis MJ (2003) Dynamic systems modeling using genetic programming. *Computers & Chemical Engineering* 27(12):1841–1854, URL <http://www.sciencedirect.com/science/article/B6TFT-49MDYGW-2/2/742bcc7f22240c7a0381027aa5ff7e73>
142. Ho SY, Hsieh CH, Chen HM, Huang HL (2006) Interpretable gene expression classifier with an accurate and compact fuzzy rule base for microarray data analysis. *Biosystems* 85(3):165–176, DOI doi:10.1016/j.biosystems.2006.01.002
143. Hoai NX, McKay RI, Abbass HA (2003) Tree adjoining grammars, language bias, and genetic programming. In: Ryan C, Soule T, Keijzer M, Tsang E, Poli R, Costa E (eds) Genetic Programming, Proceedings of EuroGP’2003, Springer-Verlag, Essex, LNCS, vol 2610, pp 335–344, URL <http://www.cs.adfa.edu.au/~abbass/publications/hardcopies/TAG3P-EuroGp-03.pdf>
144. Hoai NX, McKay RIB, Essam D (2006) Representation and structural difficulty in genetic programming. *IEEE Transactions on Evolutionary Computation* 10(2):157–166, DOI doi:10.1109/TEVC.2006.871252, URL <http://sc.snu.ac.kr/courses/2006/fall/pg/aai/GP/nguyen/Structdiff.pdf>
145. Holland J (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, USA

146. Holland JH (1992) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, first Published by University of Michigan Press 1975
147. Hong JH, Cho SB (2006) The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming. *Artificial Intelligence In Medicine* 36(1):43–58, DOI doi:10.1016/j.artmed.2005.06.002
148. Howard D, Roberts SC (2004) Incident detection on highways. In: O'Reilly UM, Yu T, Riolo RL, Worzel B (eds) *Genetic Programming Theory and Practice II*, Springer, Ann Arbor, chap 16, pp 263–282
149. Howard D, Roberts SC, Brankin R (1999) Target detection in imagery by genetic programming. *Advances in Engineering Software* 30(5):303–311, URL <http://www.sciencedirect.com/science/article/B6V1P-3W1XV4H-1/1/6e7aee809f33757d0326c62a21824411>
150. Howard D, Roberts SC, Ryan C (2006) Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters* 27(11):1275–1288, DOI doi:10.1016/j.patrec.2005.07.025, *evolutionary Computer Vision and Image Understanding*
151. Iba H (1996) *Genetic Programming*. Tokyo Denki University Press
152. Iba H, de Garis H, Sato T (1994) Genetic programming using a minimum description length principle. In: Kinnear, Jr KE (ed) *Advances in Genetic Programming*, MIT Press, chap 12, pp 265–284, URL <http://citeseer.ist.psu.edu/327857.html>
153. Inagaki Y (2002) On synchronized evolution of the network of automata. *IEEE Transactions on Evolutionary Computation* 6(2):147–158, URL <http://ieeexplore.ieee.org/iel5/4235/21497/00996014.pdf?tp=&arnumber=996014&isnumber=21497&arSt=147&ared=158&arAuthor=Inagaki%2C+Y.%3B>
154. Jacob C (1997) *Principia Evolvica – Simulierte Evolution mit Mathematica*. dpunkt.verlag, Heidelberg, Germany
155. Jacob C (2000) The art of genetic programming. *IEEE Intelligent Systems* 15(3):83–84, URL <http://ieeexplore.ieee.org/iel5/5254/18363/00846288.pdf>
156. Jacob C (2001) *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann, URL http://www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-637-8
157. Jeong KS, Kim DK, Whigham P, Joo GJ (2003) Modeling microcystis aeruginosa bloom dynamics in the nakdong river by means of evolutionary computation and statistical approach. *Ecological Modeling* 161(1–2):67–78, DOI doi:10.1016/S0304-3800(02)00280-6, URL http://www.business.otago.ac.nz/infosci/SIRC/PeterW/Publications/Jeong_EcolMod_V161_Is_1_2_pg67_78.pdf
158. Jin N, Tsang E (2006) Co-adaptive strategies for sequential bargaining problems with discount factors and outside options. In: *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, IEEE Press, Vancouver, pp 7913–7920
159. Johnson HE, Gilbert RJ, Winson MK, Goodacre R, Smith AR, Rowland JJ, Hall MA, Kell DB (2000) Explanatory analysis of the metabolome using genetic programming of simple, interpretable rules. *Genetic Programming and Evolvable Machines* 1(3):243–258, DOI doi:10.1023/A:1010014314078
160. Jones A, Young D, Taylor J, Kell DB, Rowland JJ (1998) Quantification of microbial productivity via multi-angle light scattering and supervised learning. *Biotechnology and Bioengineering* 59(2):131–143

161. Jordaan E, Kordon A, Chiang L, Smits G (2004) Robust inferential sensors based on ensemble of predictors generated by genetic programming. In: Yao X, Burke E, Lozano JA, Smith J, Merelo-Guervós JJ, Bullinaria JA, Rowe J, Kabán PTA, Schwefel HP (eds) *Parallel Problem Solving from Nature - PPSN VIII*, Springer-Verlag, Birmingham, UK, LNCS, vol 3242, pp 522–531, DOI doi:10.1007/b100601, URL <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=3242&spage=522>
162. Juille H, Pollack JB (1996) Massively parallel genetic programming. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 17, pp 339–358, URL <http://www.demon.cs.brandeis.edu/papers/gp2.pdf>
163. Kaboudan M (1999) A measure of time series predictability using genetic programming applied to stock returns. *Journal of Forecasting* 18:345–357
164. Kaboudan M (2005) Extended daily exchange rates forecasts using wavelet temporal resolutions. *New Mathematics and Natural Computing* 1:79–107
165. Kaboudan MA (2000) Genetic programming prediction of stock prices. *Computational Economics* 6(3):207–236
166. Keijzer M (1996) Efficiently representing populations in genetic programming. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 13, pp 259–278
167. Keijzer M (2004) Scaled symbolic regression. *Genetic Programming and Evolvable Machines* 5(3):259–269, DOI doi:10.1023/B:GENP.0000030195.77571.f9
168. Kell D (2002a) Defence against the flood. *Bioinformatics World* pp 16–18, URL http://dbkgroup.org/Papers/biwpp16-18_as_publ.pdf
169. Kell DB (2002b) Genotype-phenotype mapping: genes as computer programs. *Trends in Genetics* 18(11):555–559, DOI doi:10.1016/S0168-9525(02)02765-8, URL [http://dbkgroup.org/Papers/trends_genet_18_\(555\).pdf](http://dbkgroup.org/Papers/trends_genet_18_(555).pdf)
170. Kell DB (2002c) Metabolomics and machine learning: Explanatory analysis of complex metabolome data using genetic programming to produce simple, robust rules. *Molecular Biology Reports* 29(1–2):237–241, DOI doi:10.1023/A:1020342216314, URL <http://dbkgroup.org/Papers/btk2002-dbk.pdf>
171. Kell DB, Darby RM, Draper J (2001) Genomic computing. explanatory analysis of plant expression profiling data using machine learning. *Plant Physiology* 126(3):943–951
172. Keller RE, Poli R (2007a) Cost-benefit investigation of a genetic-programming hyperheuristic. In: *Proceedings of Evolution Artificielle*
173. Keller RE, Poli R (2007b) Linear genetic programming of metaheuristics. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM Press, London, vol 2, pp 1753–1753, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1753.pdf>
174. Keller RE, Poli R (2007c) Linear genetic programming of parsimonious metaheuristics. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*
175. KHosraviani B (2003) Organization design optimization using genetic programming. In: Koza JR (ed) *Genetic Algorithms and Genetic Programming at Stanford 2003*, Stanford Bookstore, Stanford, California, 94305-3079 USA, pp 109–117, URL <http://www.genetic-programming.org/sp2003/KHosraviani.pdf>

176. KHosraviani B, Levitt RE, Koza JR (2004) Organization design optimization using genetic programming. In: Keijzer M (ed) Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference, Seattle, Washington, USA, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2004/LBP056.pdf>
177. Kibria RH, Li Y (2006) Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming. In: Collet P, Tomassini M, Ebner M, Gustafson S, Ekárt A (eds) Proceedings of the 9th European Conference on Genetic Programming, Springer, Budapest, Hungary, Lecture Notes in Computer Science, vol 3905, pp 331–340, URL <http://link.springer.de/link/service/series/0558/papers/3905/39050331.pdf>
178. Kinnear, Jr KE (1993) Evolving a sort: Lessons in genetic programming. In: Proceedings of the 1993 International Conference on Neural Networks, IEEE Press, San Francisco, USA, vol 2, pp 881–888, DOI doi:10.1109/ICNN.1993.298674, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/kinnear.icnn93.ps.Z>
179. Kinnear, Jr KE (ed) (1994a) Advances in Genetic Programming. MIT Press, Cambridge, MA, URL <http://mitpress.mit.edu/book-home.tcl?isbn=0262111888>
180. Kinnear, Jr KE (1994b) Fitness landscapes and difficulty in genetic programming. In: Proceedings of the 1994 IEEE World Conference on Computational Intelligence, IEEE Press, Orlando, Florida, USA, vol 1, pp 142–147, DOI doi:10.1109/ICEC.1994.350026, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/kinnear.wcci.ps.Z>
181. Kinnear, Jr KE (1994c) A perspective on the work in this book. In: Kinnear, Jr KE (ed) Advances in Genetic Programming, MIT Press, chap 1, pp 3–19, URL <http://cognet.mit.edu/library/books/view?isbn=0262111888>
182. Klassen TJ, Heywood MI (2002) Towards the on-line recognition of arabic characters. In: Proceedings of the 2002 International Joint Conference on Neural Networks IJCNN'02, IEEE Press, Hilton Hawaiian Village Hotel, Honolulu, Hawaii, pp 1900–1905, URL <http://users.cs.dal.ca/~mheywood/X-files/Publications/IEEEArabic.pdf>
183. Klein J, Spector L (2007) Unwitting distributed genetic programming via asynchronous javascript and XML. In: Thierens D, Beyer HG, Bongard J, Branke J, Clark JA, Cliff D, Congdon CB, Deb K, Doerr B, Kovacs T, Kumar S, Miller JF, Moore J, Neumann F, Pelikan M, Poli R, Sastry K, Stanley KO, Stutzle T, Watson RA, Wegener I (eds) GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM Press, London, vol 2, pp 1628–1635, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1628.pdf>
184. Kordon A (2006) Evolutionary computation at dow chemical. SIGEVolution 1(3):4–9, URL <http://www.sigevolution.org/2006/03/issue.pdf>
185. Kordon A, Castillo F, Smits G, Kotanchek M (2005) Application issues of genetic programming in industry. In: Yu T, Riolo RL, Worzel B (eds) Genetic Programming Theory and Practice III, Genetic Programming, vol 9, Springer, Ann Arbor, chap 16, pp 241–258
186. Kovacic M, Balic J (2003) Evolutionary programming of a CNC cutting machine. International journal for advanced manufacturing technology 22(1–2):118–124, DOI doi:10.1007/s00170-002-1450-8, URL <http://www.springerlink.com/openurl.asp?genre=article&eissn=1433-3015&volume=22&issue=1&spage=118>

187. Koza JR (1990) A genetic approach to econometric modeling. In: Sixth World Congress of the Econometric Society, Barcelona, Spain, URL <http://www.genetic-programming.com/jkpdf/wces1990.pdf>
188. Koza JR (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA
189. Koza JR (1994a) Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge Massachusetts
190. Koza JR (1994b) Genetic Programming II Videotape: The next generation. MIT Press, 55 Hayward Street, Cambridge, MA, USA
191. Koza JR, Andre D (1996) Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 8, pp 155–176, URL <http://www.genetic-programming.com/jkpdf/aigp2aatmjk1996.pdf>
192. Koza JR, Poli R (2005) Genetic programming. In: Burke EK, Kendall G (eds) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer, chap 5, URL <http://www.springer.com/sgw/cda/frontpage/0,11855,4-10045-22-67933962-0,00.html>
193. Koza JR, Andre D, Bennett III FH, Keane MA (1996a) Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, Stanford University, CA, USA, pp 132–149, URL <http://www.genetic-programming.com/jkpdf/gp1996adfaa.pdf>
194. Koza JR, Bennett III FH, Andre D, Keane MA (1996b) Automated WYWI-WYG design of both the topology and component values of electrical circuits using genetic programming. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, Stanford University, CA, USA, pp 123–131, URL <http://www.genetic-programming.com/jkpdf/gp1996nielsen.pdf>
195. Koza JR, Andre D, Bennett III FH, Keane M (1999a) *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman, URL <http://www.genetic-programming.org/gpbook3toc.html>
196. Koza JR, Bennett III FH, Stiffelman O (1999b) Genetic programming as a Darwinian invention machine. In: Poli R, Nordin P, Langdon WB, Fogarty TC (eds) *Genetic Programming, Proceedings of EuroGP'99*, Springer-Verlag, Goteborg, Sweden, LNCS, vol 1598, pp 93–108, URL <http://www.genetic-programming.com/jkpdf/eurogp1999.pdf>
197. Koza JR, Keane MA, Streeter MJ, Mydlowec W, Yu J, Lanza G (2003) *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, URL <http://www.genetic-programming.org/gpbook4toc.html>
198. Koza JR, Al-Sakran SH, Jones LW (2005) Automated re-invention of six patented optical lens systems using genetic programming. In: Beyer HG, O'Reilly UM, Arnold DV, Banzhaf W, Blum C, Bonabeau EW, Cantu-Paz E, Dasgupta D, Deb K, Foster JA, de Jong ED, Lipson H, Llorca X, Mancoridis S, Pelikan M, Raidl GR, Soule T, Tyrrell AM, Watson JP, Zitzler E (eds) *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, ACM Press, Washington DC, USA, vol 2, pp 1953–1960, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005/docs/p1953.pdf>

199. Krasnogor N (2004) Self generating metaheuristics in bioinformatics: The proteins structure comparison case. *Genetic Programming and Evolvable Machines* 5(2):181–201, DOI doi:10.1023/B:GENP.0000023687.41210.d7
200. Krawiec K (2004) *Evolutionary Feature Programming: Cooperative learning for knowledge discovery and computer vision*. 385, Wydawnictwo Politechniki Poznanskiej, Poznan University of Technology, Poznan, Poland, URL http://idss.cs.put.poznan.pl/~krawiec/pubs/hab/krawiec_hab.pdf
201. Langdon WB (1996) A bibliography for genetic programming. In: Angeline PJ, Kinneer, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap B, pp 507–532, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL.aigp2.appx.ps.gz>
202. Langdon WB (1998a) The evolution of size in variable length representations. In: 1998 IEEE International Conference on Evolutionary Computation, IEEE Press, Anchorage, Alaska, USA, pp 633–638, DOI doi:10.1109/ICEC.1998.700102, URL http://www.cs.bham.ac.uk/~wbl/ftp/papers/WBL.wcci98_bloat.pdf
203. Langdon WB (1998b) *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, Genetic Programming, vol 1. Kluwer, Boston, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/gpdata>
204. Langdon WB (1999a) Scaling of program tree fitness spaces. *Evolutionary Computation* 7(4):399–428, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL.fitnessspaces.pdf>
205. Langdon WB (1999b) Size fair and homologous tree genetic programming crossovers. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, vol 2, pp 1092–1097, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL.gecco99.fairxo.ps.gz>
206. Langdon WB (2000) Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines* 1(1/2):95–119, DOI doi:10.1023/A:1010024515191, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL.fairxo.pdf>
207. Langdon WB (2002a) Convergence rates for the distribution of program outputs. In: Langdon WB, Cantú-Paz E, Mathias K, Roy R, Davis D, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke E, Jonoska N (eds) *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann Publishers, New York, pp 812–819, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_gecco2002.pdf
208. Langdon WB (2002b) How many good programs are there? How long are they? In: De Jong KA, Poli R, Rowe JE (eds) *Foundations of Genetic Algorithms VII*, Morgan Kaufmann, Torremolinos, Spain, pp 183–202, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_foga2002.pdf, published 2003
209. Langdon WB (2003a) Convergence of program fitness landscapes. In: Cantú-Paz E, Foster JA, Deb K, Davis D, Roy R, O'Reilly UM, Beyer HG, Standish R, Kendall G, Wilson S, Harman M, Wegener J, Dasgupta D, Potter MA, Schultz AC, Dowsland K, Jonoska N, Miller J (eds) *Genetic and Evolutionary Computation – GECCO-2003*, Springer-Verlag, Chicago, LNCS,

- vol 2724, pp 1702–1714, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_gecco2003.pdf
210. Langdon WB (2003b) The distribution of reversible functions is Normal. In: Riolo RL, Worzel B (eds) *Genetic Programming Theory and Practise*, Kluwer, chap 11, pp 173–188, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_reversible.pdf
 211. Langdon WB (2004) Global distributed evolution of L-systems fractals. In: Keijzer M, O'Reilly UM, Lucas SM, Costa E, Soule T (eds) *Genetic Programming, Proceedings of EuroGP'2004*, Springer-Verlag, Coimbra, Portugal, LNCS, vol 3003, pp 349–358, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/egp2004_pfeiffer.pdf
 212. Langdon WB (2005a) The distribution of amorphous computer outputs. In: Stepney S, Emmott S (eds) *The Grand Challenge in Non-Classical Computation: International Workshop*, York, UK, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/grand_2005.pdf
 213. Langdon WB (2005b) Pfeiffer – A distributed open-ended evolutionary system. In: Edmonds B, Gilbert N, Gustafson S, Hales D, Krasnogor N (eds) *AISB'05: Proceedings of the Joint Symposium on Socially Inspired Computing (METAS 2005)*, University of Hertfordshire, Hatfield, UK, pp 7–13, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_metas2005.pdf, sSAISB 2005 Convention
 214. Langdon WB (2006) Mapping non-conventional extensions of genetic programming. In: Calude CS, Dinneen MJ, Paun G, Rozenberg G, Stepney S (eds) *Unconventional Computing 2006*, Springer-Verlag, York, LNCS, vol 4135, pp 166–180, DOI doi:10.1007/11839132_14, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_uc2002.pdf
 215. Langdon WB, Banzhaf W (2005) Repeated sequences in linear genetic programming genomes. *Complex Systems* 15(4):285–306, URL http://www.cs.ucl.ac.uk/staff/rW.Langdon/ftp/papers/wbl_repeat_linear.pdf
 216. Langdon WB, Banzhaf W (2007) A SIMD interpreter for genetic programming on GPU graphics cards. In preparation
 217. Langdon WB, Buxton BF (2004) Genetic programming for mining DNA chip data from cancer patients. *Genetic Programming and Evolvable Machines* 5(3):251–257, DOI doi:10.1023/B:GENP.0000030196.55525.f7, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_dnachip.pdf
 218. Langdon WB, Harrison AP (2008) GP on SPMD parallel graphics hardware for mega bioinformatics data mining, To appear
 219. Langdon WB, Nordin P (2001) Evolving hand-eye coordination for a humanoid robot with machine code genetic programming. In: Miller JF, Tomassini M, Lanzi PL, Ryan C, Tettamanzi AGB, Langdon WB (eds) *Genetic Programming, Proceedings of EuroGP'2001*, Springer-Verlag, Lake Como, Italy, LNCS, vol 2038, pp 313–324, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_handeye.ps.gz
 220. Langdon WB, Poli R (2008) Mapping non-conventional extensions of genetic programming. *Natural Computing* 7:21–43. Invited contribution to special issue on Unconventional computing
 221. Langdon WB, Poli R (1997) Fitness causes bloat. In: Chawdhry PK, Roy R, Pant RK (eds) *Soft Computing in Engineering Design and Manufacturing*, Springer-Verlag London, pp 13–22, URL http://www.rcs.bham.ac.uk/~wbl/ftp/papers/WBL.bloat_wsc2.ps.gz

222. Langdon WB, Poli R (2002) *Foundations of Genetic Programming*. Springer-Verlag, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/FOGP/>
223. Langdon WB, Poli R (2006a) The halting probability in von Neumann architectures. In: Collet P, Tomassini M, Ebner M, Gustafson S, Ekárt A (eds) *Proceedings of the 9th European Conference on Genetic Programming*, Springer, Budapest, Hungary, Lecture Notes in Computer Science, vol 3905, pp 225–237, URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_egp2006.pdf
224. Langdon WB, Poli R (2006b) On turing complete T7 and MISC F-4 program fitness landscapes. In: Arnold DV, Jansen T, Vose MD, Rowe JE (eds) *Theory of Evolutionary Algorithms, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany*, no. 06061 in *Dagstuhl Seminar Proceedings*, URL <http://drops.dagstuhl.de/opus/volltexte/2006/595>, <<http://drops.dagstuhl.de/opus/volltexte/2006/595>> [date of citation: 2006-01-01]
225. Langdon WB, Soule T, Poli R, Foster JA (1999) The evolution of size and shape. In: Spector L, Langdon WB, O'Reilly UM, Angeline PJ (eds) *Advances in Genetic Programming 3*, MIT Press, Cambridge, MA, USA, chap 8, pp 163–190, URL <http://www.cs.bham.ac.uk/~wbl/aigp3/ch08.pdf>
226. Leung KS, Lee KH, Cheang SM (2002) Genetic parallel programming - evolving linear machine codes on a multiple-ALU processor. In: Yaacob S, Nagarajan R, Chekima A (eds) *Proceedings of International Conference on Artificial Intelligence in Engineering and Technology - ICAIET 2002*, Universiti Malaysia Sabah, pp 207–213
227. Lew TL, Spencer AB, Scarpa F, Worden K, Rutherford A, Hemez F (2006) Identification of response surface models using genetic programming. *Mechanical Systems and Signal Processing* 20(8):1819–1831, DOI doi:10.1016/j.ymssp.2005.12.003
228. Lewin DR, Lachman-Shalem S, Grosman B (2006) The role of process system engineering (PSE) in integrated circuit (IC) manufacturing. *Control Engineering Practice* 15(7):793–802, DOI doi:10.1016/j.conengprac.2006.04.003, special Issue on Award Winning Applications, 2005 IFAC World Congress
229. Li L, Jiang W, Li X, Moser KL, Guo Z, Du L, Wang Q, Topol EJ, Wang Q, Rao S (2005) A robust hybrid between genetic algorithm and support vector machine for extracting an optimal feature gene subset. *Genomics* 85(1):16–23, DOI doi:10.1016/j.ygeno.2004.09.007
230. Linden R, Bhaya A (2007) Evolving fuzzy rules to model gene expression. *Biosystems* 88(1-2):76–91, DOI doi:10.1016/j.biosystems.2006.04.006
231. Lipson H (2004) How to draw a straight line using a GP: Benchmarking evolutionary design against 19th century kinematic synthesis. In: Keijzer M (ed) *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2004/LBP063.pdf>
232. Liu W, Schmidt B (2006) Mapping of hierarchical parallel genetic algorithms for protein folding onto computational grids. *IEICE Transactions on Information and Systems* E89-D(2):589–596, DOI doi:10.1093/ietisy/e89-d.2.589
233. Lohn J, Hornby G, Linden D (2004) Evolutionary antenna design for a NASA spacecraft. In: O'Reilly UM, Yu T, Riolo RL, Worzel B (eds) *Genetic Programming Theory and Practice II*, Springer, Ann Arbor, chap 18, pp 301–315

234. Lohn JD, Hornby GS, Linden DS (2005) Rapid re-evolution of an X-band antenna for NASA's space technology 5 mission. In: Yu T, Riolo RL, Worzel B (eds) Genetic Programming Theory and Practice III, Genetic Programming, vol 9, Springer, Ann Arbor, chap 5, pp 65–78
235. Louchet J (2001) Using an individual evolution strategy for stereovision. *Genetic Programming and Evolvable Machines* 2(2):101–109, DOI doi:10.1023/A:1011544128842
236. Louchet J, Guyon M, Lesot MJ, Boumaza A (2002) Dynamic flies: a new pattern recognition tool applied to stereo sequence processing. *Pattern Recognition Letters* 23(1–3):335–345, DOI doi:10.1016/S0167-8655(01)00129-5
237. Loviscach J, Meyer-Spradow J (2003) Genetic programming of vertex shaders. In: Chover M, Hagen H, Tost D (eds) *Proceedings of EuroMedia 2003*, pp 29–31
238. Luke S (1998) Evolving soccerbots: A retrospective. In: *Proceedings of the 12th Annual Conference of the Japanese Society for Artificial Intelligence*, URL <http://www.cs.gmu.edu/~sean/papers/robocupShort.pdf>
239. Luke S (2000) Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation* 4(3):274–283, URL <http://ieeexplore.ieee.org/iel5/4235/18897/00873237.pdf>
240. Lukschandl E, Borgvall H, Nohle L, Nordahl M, Nordin P (2000) Distributed java bytecode genetic programming. In: Poli R, Banzhaf W, Langdon WB, Miller JF, Nordin P, Fogarty TC (eds) *Genetic Programming, Proceedings of EuroGP'2000*, Springer-Verlag, Edinburgh, LNCS, vol 1802, pp 316–325, URL <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=1802&spage=316>
241. Machado P, Romero J (eds) (2008) *The Art of Artificial Evolution*. Springer
242. Marenbach P (1998) Using prior knowledge and obtaining process insight in data based modeling of bioprocesses. *System Analysis Modeling Simulation* 31:39–59
243. Markose S, Tsang E, Er H, Salhi A (2001) Evolutionary arbitrage for FTSE-100 index options and futures. In: *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, pp 275–282, DOI doi:10.1109/CEC.2001.934401, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/TsangCEE2001.pdf>
244. Marney JP, Miller D, Fyfe C, Tarbert HFE (2001) Risk adjusted returns to technical trading rules: a genetic programming approach. In: *7th International Conference of Society of Computational Economics*, Yale
245. Martin MC (2006) Evolving visual sonar: Depth from monocular images. *Pattern Recognition Letters* 27(11):1174–1180, DOI doi:10.1016/j.patrec.2005.07.015, URL <http://martinmartin.com/papers/EvolvingVisualSonar-PatternRecognitionLetters2006.pdf>, *evolutionary Computer Vision and Image Understanding*
246. Martin P (2001) A hardware implementation of a genetic programming system using FPGAs and Handel-C. *Genetic Programming and Evolvable Machines* 2(4):317–343, DOI doi:10.1023/A:1012942304464, URL <http://www.naiadhome.com/gpem-d.pdf>
247. Massey P, Clark JA, Stepney S (2005) Evolution of a human-competitive quantum fourier transform algorithm using genetic programming. In: Beyer HG, O'Reilly UM, Arnold DV, Banzhaf W, Blum C, Bonabeau EW, Cantu-Paz E,

- Dasgupta D, Deb K, Foster JA, de Jong ED, Lipson H, Llorca X, Mancoridis S, Pelikan M, Raidl GR, Soule T, Tyrrell AM, Watson JP, Zitzler E (eds) GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM Press, Washington DC, USA, vol 2, pp 1657–1663, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005/docs/p1657.pdf>
248. Maxwell III SR (1994) Experiments with a coroutine model for genetic programming. In: Proceedings of the 1994 IEEE World Congress on Computational Intelligence, IEEE Press, Orlando, Florida, USA, vol 1, pp 413–417a, URL <http://ieeexplore.ieee.org/iel2/1125/8059/00349915.pdf?isNumber=8059>
249. McCormack J (2006) New challenges for evolutionary music and art. *SIGEvolution* 1(1):5–11, URL <http://www.sigevolution.org/2006/01/issue.pdf>
250. McGovern AC, Broadhurst D, Taylor J, Kaderbhai N, Winson MK, Small DA, Rowland JJ, Kell DB, Goodacre R (2002) Monitoring of complex industrial bioprocesses for metabolite concentrations using modern spectroscopies and machine learning: Application to gibberellic acid production. *Biotechnology and Bioengineering* 78(5):527–538, DOI doi:10.1002/bit.10226, URL [http://dbkgroup.org/Papers/biotechnol_bioeng_78_\(527\).pdf](http://dbkgroup.org/Papers/biotechnol_bioeng_78_(527).pdf)
251. McKay B, Willis M, Searson D, Montague G (2000) Nonlinear continuum regression: an evolutionary approach. *Transactions of the Institute of Measurement and Control* 22(2):125–140, doi:10.1177/014233120002200202, URL <http://www.ingentaconnect.com/content/arn/tm/2000/00000022/00000002/art00007>
252. McPhee NF, Miller JD (1995) Accurate replication in genetic programming. In: Eshelman L (ed) *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Morgan Kaufmann, Pittsburgh, PA, USA, pp 303–309, URL http://www.mrs.umn.edu/~mcphee/Research/Accurate_replication.ps
253. McPhee NF, Hopper NJ, Reiersen ML (1998) Sutherland: An extensible object-oriented software framework for evolutionary computation. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA, p 241, URL http://www.mrs.umn.edu/~mcphee/Research/Sutherland/rsutherland_gp98_announcement.ps.gz
254. Mercure PK, Smits GF, Kordon A (2001) Empirical emulators for first principle models. In: *AICHE Fall Annual Meeting*, Reno Hilton, URL <http://www.aiche.org/conferences/techprogram/paperdetail.asp?PaperID=2373&DSN=annual01>
255. Meyer-Spradow J, Loviscach J (2003) Evolutionary design of BRDFs. In: Chover M, Hagen H, Tost D (eds) *Eurographics 2003 Short Paper Proceedings*, pp 301–306, URL http://viscg.uni-muenster.de/publications/2003/ML03/evolutionary_web.pdf
256. Miller JF (1999) An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, vol 2, pp 1135–1142, URL <http://citeseer.ist.psu.edu/153431.html>

257. Miller JF, Smith SL (2006) Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation* 10(2):167–174, DOI doi:10.1109/TEVC.2006.871253
258. Mitavskiy B, Rowe J (2006) Some results about the markov chains associated to GPs and to general EAs. *Theoretical Computer Science* 361(1):72–110, DOI doi:10.1016/j.tcs.2006.04.006
259. Montana DJ (1995) Strongly typed genetic programming. *Evolutionary Computation* 3(2):199–230, URL <http://vishnu.bbn.com/papers/stgp.pdf>
260. Moore GE (1965) Cramming more components onto integrated circuits. *Electronics* 38(8):114–117
261. Moore JH, Parker JS, Olsen NJ, Aune TM (2002) Symbolic discriminant analysis of microarray data in automimmune disease. *Genetic Epidemiology* 23:57–69
262. Motsinger AA, Lee SL, Mellick G, Ritchie MD (2006) GPNN: Power studies and applications of a neural network method for detecting gene-gene interactions in studies of human disease. *BMC bioinformatics* [electronic resource] 7(1):39–39, DOI doi:10.1186/1471-2105-7-39, URL <http://www.biomedcentral.com/1471-2105/7/39>
263. Neely CJ (2003) Risk-adjusted, ex ante, optimal technical trading rules in equity markets. *International Review of Economics and Finance* 12(1):69–87, DOI doi:10.1016/S1059-0560(02)00129-6, URL <http://research.stlouisfed.org/wp/1999/1999-015.pdf>
264. Neely CJ, Weller PA (1999) Technical trading rules in the european monetary system. *Journal of International Money and Finance* 18(3):429–458, DOI doi:10.1016/S0261-5606(99)85005-0, URL <http://research.stlouisfed.org/wp/1997/97-015.pdf>
265. Neely CJ, Weller PA (2001a) Predicting exchange rate volatility: Genetic programming vs. GARCH and risk metrics. Working Paper 2001-009B, Economic, Research, Federal Reserve Bank of St. Louis, 411 Locust Street, St. Louis, MO 63102-0442, USA, URL <http://research.stlouisfed.org/wp/2001/2001-009.pdf>
266. Neely CJ, Weller PA (2001b) Technical analysis and central bank intervention. *Journal of International Money and Finance* 20(7):949–970, DOI doi:10.1016/S0261-5606(01)00033-X, URL <http://research.stlouisfed.org/wp/1997/97-002.pdf>
267. Neely CJ, Weller PA, Dittmar R (1997) Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *The Journal of Financial and Quantitative Analysis* 32(4):405–426, URL <http://links.jstor.org/sici?sici=0022-1090%28199712%2932%3A4%3C405%3AITAITF%3E2.0.CO%3B2-T>
268. Neely CJ, Weller PA, Ulrich JM (2006) The adaptive markets hypothesis: evidence from the foreign exchange market. Working Paper 2006-046B, Federal Reserve Bank of St. Louis, Research Division, P.O. Box 442, St. Louis, MO 63166, USA, URL <http://research.stlouisfed.org/wp/2006/2006-046.pdf>, revised March 2007
269. Nikolaev N, Iba H (2006) Adaptive Learning of Polynomial Networks Genetic Programming, Backpropagation and Bayesian Methods. No. 4 in *Genetic and Evolutionary Computation*, Springer, june
270. Nikolaev NY, Iba H (2002) Genetic programming of polynomial models for financial forecasting. In: Chen SH (ed) *Genetic Algorithms and Genetic*

- Programming in Computational Finance, Kluwer Academic Press, chap 5, pp 103–123
271. Nix AE, Vose MD (1992) Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence* 5:79–88
 272. Nordin P (1994) A compiling genetic programming system that directly manipulates the machine code. In: Kinnear, Jr KE (ed) *Advances in Genetic Programming*, MIT Press, chap 14, pp 311–331, URL <http://cognet.mit.edu/library/books/view?isbn=0262111888>
 273. Nordin P (1997) Evolutionary program induction of binary machine code and its applications. PhD thesis, der Universitat Dortmund am Fachereich Informatik
 274. Nordin P, Johanna W (2003) *Humanoider: Sjavlarande robotar och artificiell intelligens*. Liber
 275. Nordin P, Banzhaf W, Francone FD (1999) Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In: Spector L, Langdon WB, O'Reilly UM, Angeline PJ (eds) *Advances in Genetic Programming 3*, MIT Press, Cambridge, MA, USA, chap 12, pp 275–299, URL <http://www.aimlearning.com/aigp31.pdf>
 276. Oakley H (1994) Two scientific applications of genetic programming: Stack filters and non-linear equation fitting to chaotic data. In: Kinnear, Jr KE (ed) *Advances in Genetic Programming*, MIT Press, chap 17, pp 369–389, URL <http://cognet.mit.edu/library/books/view?isbn=0262111888>
 277. Oltean M (2005) Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation* 13(3):387–410, DOI doi:10.1162/1063656054794815, URL <http://www.ingentaconnect.com/content/mitpress/evco/2005/00000013/00000003/art00006>
 278. Oltean M, Dumitrescu D (2004) Evolving TSP heuristics using multi expression programming. In: Bubak M, van Albada GD, Sloot PMA, Dongarra J (eds) *Computational Science - ICCS 2004: 4th International Conference, Part II*, Springer-Verlag, Krakow, Poland, Lecture Notes in Computer Science, vol 3037, pp 670–673, DOI doi:10.1007/b97988, URL <http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3037&spage=670>
 279. O'Neill M, Ryan C (2003) *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, Genetic programming, vol 4. Kluwer Academic Publishers, URL <http://www.wkap.nl/prod/b/1-4020-7444-1>
 280. Openshaw S, Turton I (1994) Building new spatial interaction models using genetic programming. In: Fogarty TC (ed) *Evolutionary Computing*, Springer-Verlag, Leeds, UK, Lecture Notes in Computer Science, URL <http://www.geog.leeds.ac.uk/papers/94-1/94-1.pdf>
 281. O'Reilly UM (1995) An analysis of genetic programming. PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/oreilly/abstract.ps.gz>
 282. O'Reilly UM, Hemberg M (2007) Integrating generative growth and evolutionary computation for form exploration. *Genetic Programming and Evolvable Machines* 8(2):163–186, DOI doi:10.1007/s10710-007-9025-y, special issue on developmental systems
 283. O'Reilly UM, Oppacher F (1994) The troubling aspects of a building block hypothesis for genetic programming. In: Whitley LD, Vose MD (eds)

- Foundations of Genetic Algorithms 3, Morgan Kaufmann, Estes Park, Colorado, USA, pp 73–88, URL <http://citeseer.ist.psu.edu/cache/papers/cs/163/http:zSzzSzwww.ai.mit.eduSzpeoplezSzunamayzSzpaperszSzfoga.pdf/oreilly92troubling.pdf>, published 1995
284. O'Reilly UM, Yu T, Riolo RL, Worzel B (eds) (2004) Genetic Programming Theory and Practice II, Genetic Programming, vol 8, Springer, Ann Arbor, MI, USA, URL <http://www.springeronline.com/sgw/cda/frontpage/0,11855,5-40356-22-34954683-0,00.html>
 285. Oussaidène M, Chopard B, Pictet OV, Tomassini M (1997) Parallel genetic programming and its application to trading model induction. *Parallel Computing* 23(8):1183–1198, URL <http://citeseer.ist.psu.edu/cache/papers/cs/166/http:zSzzSzslwww.epfl.chzSzmarcozSzparcomp.pdf/oussaidene97parallel.pdf>
 286. Owens JD, David, Govindaraju N, Harris M, Kruger J, Lefohn AE, Purcell TJ (2007) A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum* 26(1):80–113, DOI doi:10.1111/j.1467-8659.2007.01012.x
 287. Parrott D, Li X, Ciesielski V (2005) Multi-objective techniques in genetic programming for evolving classifiers. In: Corne D, Michalewicz Z, Dorigo M, Eiben G, Fogel D, Fonseca C, Greenwood G, Chen TK, Raidl G, Zalzala A, Lucas S, Paechter B, Willies J, Guervos JJM, Eberbach E, McKay B, Channon A, Tiwari A, Volkert LG, Ashlock D, Schoenauer M (eds) *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, IEEE Press, Edinburgh, UK, vol 2, pp 1141–1148, URL <http://goanna.cs.rmit.edu.au/~xiaodong/publications/183.pdf>
 288. Perks T (1994) Stack-based genetic programming. In: *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, IEEE Press, Orlando, Florida, USA, vol 1, pp 148–153, URL <http://citeseer.ist.psu.edu/432690.html>
 289. Pillay N (2003) Evolving solutions to ASCII graphics programming problems in intelligent programming tutors. In: Akerkar R (ed) *International Conference on Applied Artificial Intelligence (ICAAI'2003)*, TMRF, Fort Panhala, Kolhapur, India, pp 236–243
 290. Poli R (1996a) Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. Tech. Rep. CSRP-96-14, University of Birmingham, School of Computer Science, URL <ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-14.ps.gz>, presented at 3rd International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'97
 291. Poli R (1996b) Genetic programming for image analysis. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, Stanford University, CA, USA, pp 363–368, URL <http://cswww.essex.ac.uk/staff/rpoli/papers/Poli-GP1996.pdf>
 292. Poli R (1999) Parallel distributed genetic programming. In: Corne D, Dorigo M, Glover F (eds) *New Ideas in Optimization*, Advanced Topics in Computer Science, McGraw-Hill, Maidenhead, Berkshire, England, chap 27, pp 403–431, URL <http://citeseer.ist.psu.edu/328504.html>
 293. Poli R (2000a) Exact schema theorem and effective fitness for GP with one-point crossover. In: Whitley D, Goldberg D, Cantu-Paz E, Spector L, Parmee I, Beyer HG (eds) *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Las Vegas, pp 469–476

294. Poli R (2000b) Hyperschema theory for GP with one-point crossover, building blocks, and some new results in GA theory. In: Poli R, Banzhaf W, Langdon WB, Miller JF, Nordin P, Fogarty TC (eds) Genetic Programming, Proceedings of EuroGP'2000, Springer-Verlag, Edinburgh, LNCS, vol 1802, pp 163–180, URL <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=1802&spage=163>
295. Poli R (2001) Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines* 2(2):123–163
296. Poli R (2003) A simple but theoretically-motivated method to control bloat in genetic programming. In: Ryan C, Soule T, Keijzer M, Tsang E, Poli R, Costa E (eds) Genetic Programming, Proceedings of EuroGP'2003, Springer-Verlag, Essex, LNCS, vol 2610, pp 204–217, URL <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2610&spage=204>
297. Poli R (2005) Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms. In: Wright AH, Vose MD, De Jong KA, Schmitt LM (eds) Foundations of Genetic Algorithms 8, Springer-Verlag, Aizu-Wakamatsu City, Japan, Lecture Notes in Computer Science, vol 3469, pp 132–155, URL http://www.cs.essex.ac.uk/staff/rpoli/papers/foga2005_Poli.pdf
298. Poli R, Langdon WB (1997) A new schema theory for genetic programming with one-point crossover and point mutation. In: Koza JR, Deb K, Dorigo M, Fogel DB, Garzon M, Iba H, Riolo RL (eds) Genetic Programming 1997: Proceedings of the Second Annual Conference, Morgan Kaufmann, Stanford University, CA, USA, pp 278–285, URL <http://citeseer.ist.psu.edu/327495.html>
299. Poli R, Langdon WB (1998a) On the search properties of different crossover operators in genetic programming. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) Genetic Programming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA, pp 293–301, URL <http://www.cs.essex.ac.uk/staff/poli/papers/Poli-GP1998.pdf>
300. Poli R, Langdon WB (1998b) Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation* 6(3):231–252, URL <http://cswww.essex.ac.uk/staff/poli/papers/Poli-ECJ1998.pdf>
301. Poli R, Langdon WB (2005a) Running genetic programming backward. In: Riolo RL, Worzel B, Yu T (eds) Genetic Programming Theory and Practice, Kluwer
302. Poli R, Langdon WB (2005b) Running genetic programming backward. In: Yu T, Riolo RL, Worzel B (eds) Genetic Programming Theory and Practice III, Genetic Programming, vol 9, Springer, Ann Arbor, chap 9, pp 125–140, URL <http://www.cs.essex.ac.uk/staff/poli/papers/GPTP2005.pdf>
303. Poli R, Langdon WB (2006a) Backward-chaining evolutionary algorithms. *Artificial Intelligence* 170(11):953–982, DOI doi:10.1016/j.artint.2006.04.003, URL <http://www.cs.essex.ac.uk/staff/poli/papers/aijournal2006.pdf>
304. Poli R, Langdon WB (2006b) Efficient markov chain model of machine code program execution and halting. In: Riolo RL, Soule T, Worzel B (eds) Genetic Programming Theory and Practice IV, Genetic and Evolutionary Computation, vol 5, Springer, Ann Arbor, chap 13, URL <http://www.cs.essex.ac.uk/staff/poli/papers/GPTP2006.pdf>

305. Poli R, McPhee NF (2003a) General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation* 11(1):53–66, DOI doi:10.1162/106365603321829005, URL <http://cswww.essex.ac.uk/staff/rpoli/papers/ecj2003partI.pdf>
306. Poli R, McPhee NF (2003b) General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation* 11(2):169–206, DOI doi:10.1162/106365603766646825, URL <http://cswww.essex.ac.uk/staff/rpoli/papers/ecj2003partII.pdf>
307. Poli R, Page J, Langdon WB (1999) Smooth uniform crossover, sub-machine code GP and demes: A recipe for solving high-order boolean parity problems. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, vol 2, pp 1162–1169, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GP-466.pdf>
308. Poli R, Rowe JE, McPhee NF (2001) Markov chain models for GP and variable-length GAs with homologous crossover. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt HM, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, Morgan Kaufmann, San Francisco, California, USA, pp 112–119, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2001/d01.pdf>
309. Poli R, McPhee NF, Rowe JE (2004) Exact schema theory and markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. *Genetic Programming and Evolvable Machines* 5(1):31–70, DOI doi:10.1023/B:GENP.0000017010.41337.a7, URL <http://cswww.essex.ac.uk/staff/rpoli/papers/GPEM2004.pdf>
310. Poli R, Di Chio C, Langdon WB (2005a) Exploring extended particle swarms: a genetic programming approach. In: Beyer HG, O’Reilly UM, Arnold DV, Banzhaf W, Blum C, Bonabeau EW, Cantu-Paz E, Dasgupta D, Deb K, Foster JA, de Jong ED, Lipson H, Llorca X, Mancoridis S, Pelikan M, Raidl GR, Soule T, Tyrrell AM, Watson JP, Zitzler E (eds) *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, ACM Press, Washington DC, USA, vol 1, pp 169–176, URL <http://www.cs.essex.ac.uk/staff/poli/papers/geccopso2005.pdf>
311. Poli R, Langdon WB, Holland O (2005b) Extending particle swarm optimisation via genetic programming. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JI, Tomassini M (eds) *Proceedings of the 8th European Conference on Genetic Programming*, Springer, Lausanne, Switzerland, *Lecture Notes in Computer Science*, vol 3447, pp 291–300, URL <http://www.cs.essex.ac.uk/staff/poli/papers/eurogpPSO2005.pdf>
312. Poli R, Langdon WB, Dignum S (2007a) On the limiting distribution of program sizes in tree-based genetic programming. In: Ebner M, O’Neill M, Ekárt A, Vanneschi L, Esparcia-Alcázar AI (eds) *Proceedings of the 10th European Conference on Genetic Programming*, Springer, Valencia, Spain, *Lecture Notes in Computer Science*, vol 4445, pp 193–204, DOI doi:10.1007/978-3-540-71605-1_18
313. Poli R, Woodward J, Burke E (2007b) A histogram-matching approach to the evolution of bin-packing strategies. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, Singapore, accepted

314. Potter MA (1997) The design and analysis of a computational model of cooperative coevolution. PhD thesis, George Mason University, Washington, DC, URL <http://www.cs.gmu.edu/~mpotter/dissertation.html>
315. Priesterjahn S, Kramer O, Weimer A, Goebels A (2006) Evolution of human-competitive agents in modern computer games. In: Yen GG, Lucas SM, Fogel G, Kendall G, Salomon R, Zhang BT, Coello CAC, Runarsson TP (eds) Proceedings of the 2006 IEEE Congress on Evolutionary Computation, IEEE Press, Vancouver, BC, Canada, pp 777–784, URL <http://ieeexplore.ieee.org/servlet/opac?punumber=11108>
316. Prügel-Bennett A, Shapiro JL (1994) An analysis of genetic algorithms using statistical mechanics. *Physical Review Letters* 72:1305–1309
317. Quintana MI, Poli R, Claridge E (2006) Morphological algorithm design for binary images using genetic programming. *Genetic Programming and Evolvable Machines* 7(1):81–102, DOI doi:10.1007/s10710-006-7012-3, URL <http://cswwww.essex.ac.uk/staff/rpoli/papers/gpem2005.pdf>
318. Ratle A, Sebag M (2000) Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In: Schoenauer M, Deb K, Rudolph G, Yao X, Lutton E, Merelo JJ, Schwefel HP (eds) *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Springer Verlag, Paris, France, LNCS, vol 1917, pp 211–220, URL <http://www.lri.fr/~sebag/REF/PPSN00.ps>
319. Reggia J, Tagamets M, Contreras-Vidal J, Jacobs D, Weems S, Naqvi W, Winder R, Chabuk T, Jung J, Yang C (2006) Development of a large-scale integrated neurocognitive architecture - part 2: Design and architecture. Tech. Rep. TR-CS-4827, UMIACS-TR-2006-43, University of Maryland, USA, URL <https://drum.umd.edu/dspace/bitstream/1903/3957/1/MarylandPart2.pdf>
320. Reif DM, White BC, Moore JH (2004) Integrated analysis of genetic, genomic, and proteomic data. *Expert Review of Proteomics* 1(1):67–75, DOI doi:10.1586/14789450.1.1.67, URL <http://www.future-drugs.com/doi/abs/10.1586/14789450.1.1.67>
321. Reynolds CW (1987) Flocks, herds, and schools: A distributed behavioral model. *SIGGRAPH Computer Graphics* 21(4):25–34, URL <http://www.red3d.com/cwr/papers/1987/boids.html>
322. Riolo RL, Worzel B (2003) *Genetic Programming Theory and Practice*, Genetic Programming, vol 6. Kluwer, Boston, MA, USA, URL <http://www.wkap.nl/prod/b/1-4020-7581-2>, series Editor - John Koza
323. Riolo RL, Soule T, Worzel B (eds) (2007a) *Genetic Programming Theory and Practice IV*, Genetic and Evolutionary Computation, vol 5, Springer, Ann Arbor, URL <http://www.springer.com/west/home/computer/foundations?SGWID=%4-156-22-173660377-0>
324. Riolo RL, Soule T, Worzel B (eds) (2007b) *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, Springer, Ann Arbor
325. Ritchie MD, White BC, Parker JS, Hahn LW, Moore JH (2003) Optimization of neural network architecture using genetic programming improves detection and modeling of gene-gene interactions in studies of human diseases. *BMC Bioinformatics* 4(28), DOI doi:10.1186/1471-2105-4-28, URL <http://www.biomedcentral.com/1471-2105/4/28>
326. Ritchie MD, Motsinger AA, Bush WS, Coffey CS, Moore JH (2007) Genetic programming neural networks: A powerful bioinformatics tool for human

- genetics. *Applied Soft Computing* 7(1):471–479, DOI doi:10.1016/j.asoc.2006.01.013
327. Rivero D, nal JRR, Dorado J, Pazos A (2004) Using genetic programming for character discrimination in damaged documents. In: Raidl GR, Cagnoni S, Branke J, Corne DW, Drechsler R, Jin Y, Johnson CR, Machado P, Marchiori E, Rothlauf F, Smith GD, Squillero G (eds) *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, Springer Verlag, Coimbra, Portugal, LNCS, vol 3005, pp 349–358
 328. Robinson A, Spector L (2002) Using genetic programming with multiple data types and automatic modularization to evolve decentralized and coordinated navigation in multi-agent systems. In: Cantú-Paz E (ed) *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, AAAI, New York, NY, pp 391–396
 329. Rodriguez-Vazquez K, Fonseca CM, Fleming PJ (2004) Identifying the structure of nonlinear dynamic systems using multiobjective genetic programming. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 34(4):531–545
 330. Rosca JP (1997) Analysis of complexity drift in genetic programming. In: Koza JR, Deb K, Dorigo M, Fogel DB, Garzon M, Iba H, Riolo RL (eds) *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Morgan Kaufmann, Stanford University, CA, USA, pp 286–294, URL <ftp://ftp.cs.rochester.edu/pub/u/rosca/gp/97.gp.ps.gz>
 331. Rosca JP, Ballard DH (1996) Discovery of subroutines in genetic programming. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 9, pp 177–202, URL <ftp://ftp.cs.rochester.edu/pub/u/rosca/gp/96.aigp2.dsgp.ps.gz>
 332. Ross BJ, Gualtieri AG, Fueten F, Budkewitsch P (2005) Hyperspectral image analysis using genetic programming. *Applied Soft Computing* 5(2):147–156, DOI doi:10.1016/j.asoc.2004.06.003, URL http://www.cosc.brocku.ca/~bross/research/gp_hyper.pdf
 333. Rothlauf F (2006) *Representations for genetic and evolutionary algorithms*, 2nd edn. Springer-Verlag, pub-SV:adr, URL <http://download-ebook.org/index.php?target=desc&ebookid=5771>, first published 2002, 2nd edition available electronically
 334. Ryan C (1999) *Automatic Re-engineering of Software Using Genetic Programming*, Genetic Programming, vol 2. Kluwer Academic Publishers, URL <http://www.wkap.nl/book.htm/0-7923-8653-1>
 335. Ryan C, Ivan L (1999) An automatic software re-engineering tool based on genetic programming. In: Spector L, Langdon WB, O'Reilly UM, Angeline PJ (eds) *Advances in Genetic Programming 3*, MIT Press, Cambridge, MA, USA, Ann Arbor, URL <http://www.cs.bham.ac.uk/~wbl/aigp3/ch02.pdf>
 336. Ryan C, Collins JJ, O'Neill M (1998) Grammatical evolution: Evolving programs for an arbitrary language. In: Banzhaf W, Poli R, Schoenauer M, Fogarty TC (eds) *Proceedings of the First European Workshop on Genetic Programming*, Springer-Verlag, Paris, LNCS, vol 1391, pp 83–95, URL <http://www.lania.mx/~ccoello/eurogp98.ps.gz>
 337. Samuel AL (1983) AI, where it has been and where it is going. In: *IJCAI*, pp 1152–1157

338. Schmidt MD, Lipson H (2006) Co-evolving fitness predictors for accelerating and reducing evaluations. In: Riolo RL, Soule T, Worzel B (eds) *Genetic Programming Theory and Practice IV, Genetic and Evolutionary Computation*, vol 5, Springer, Ann Arbor
339. Schoenauer M, Sebag M (2001) Using domain knowledge in evolutionary system identification. In: Giannakoglou KC, Tsahalis D, Periaux J, Papailiou K, Fogarty TC (eds) *Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, Athens
340. Schoenauer M, Lamy B, Jouve F (1995) Identification of mechanical behavior by genetic programming part II: Energy formulation. Tech. rep., Ecole Polytechnique, 91128 Palaiseau, France
341. Schoenauer M, Sebag M, Jouve F, Lamy B, Maitournam H (1996) Evolutionary identification of macro-mechanical models. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 23, pp 467–488, URL <http://citeseer.ist.psu.edu/cache/papers/cs/902/http:zSzzSzwww.eeaax.polytechnique.frzSzpaperszSzmarczSzAGP2.pdf/schoenauer96evolutionary.pdf>
342. Searson DP, Montague GA, Willis MJ (1998) Evolutionary design of process controllers. In: *In Proceedings of the 1998 United Kingdom Automatic Control Council International Conference on Control (UKACC International Conference on Control '98)*, Institution of Electrical Engineers (IEE), University of Wales, Swansea, UK, IEE Conference Publications, vol 455, URL <http://www.staff.ncl.ac.uk/d.p.searson/docs/Searsoncontrol98.pdf>
343. Sekanina L (2003) *Evolvable Components: From Theory to Hardware Implementations*. Natural Computing, Springer-Verlag, URL <http://www.fit.vutbr.cz/~sekanina/rehw/books.html.en>
344. Setzkorn C (2005) *On the use of multi-objective evolutionary algorithms for classification rule induction*. PhD thesis, University of Liverpool, UK
345. Shah SC, Kusiak A (2004) Data mining and genetic algorithm based gene/SNP selection. *Artificial Intelligence in Medicine* 31(3):183–196, DOI doi:10.1016/j.artmed.2004.04.002, URL http://www.icaen.uiowa.edu/~ankusiak/Journalpapers/Gen_Shital.pdf
346. Sharabi S, Sipper M (2006) GP-sumo: Using genetic programming to evolve sumobots. *Genetic Programming and Evolvable Machines* 7(3):211–230, DOI doi:10.1007/s10710-006-9006-6
347. Sharman KC, Esparcia-Alcazar AI (1993) Genetic evolution of symbolic signal models. In: *Proceedings of the Second International Conference on Natural Algorithms in Signal Processing, NASP'93*, Essex University, UK, URL <http://www.iti.upv.es/~anna/papers/natalg93.ps>
348. Sharman KC, Esparcia Alcazar AI, Li Y (1995) Evolving signal processing algorithms by genetic programming. In: Zalzal AMS (ed) *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, GALESIA, IEE, Sheffield, UK, vol 414, pp 473–480, URL <http://www.iti.upv.es/~anna/papers/galesi95.ps>
349. Shaw AD, Winson MK, Woodward AM, McGovern AC, Davey HM, Kaderbhai N, Broadhurst D, Gilbert RJ, Taylor J, Timmins EM, Goodacre R, Kell DB, Alsberg BK, Rowland JJ (2000) Bioanalysis and biosensors for bioprocess monitoring rapid analysis of high-dimensional bioprocesses using multivariate spectroscopies and advanced chemometrics. *Advances in Biochemical*

- Engineering/Biotechnology 66:83–113, URL <http://www.springerlink.com/link.asp?id=t8b4ya0bl42jnjj3>
350. Shichel Y, Ziserman E, Sipper M (2005) GP-robocode: Using genetic programming to evolve robocode players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert JJ, Tomassini M (eds) Proceedings of the 8th European Conference on Genetic Programming, Springer, Lausanne, Switzerland, Lecture Notes in Computer Science, vol 3447, pp 143–154, URL <http://www.cs.bgu.ac.il/~sipper/papabs/eurogprobo-final.pdf>
 351. Si HZ, Wang T, Zhang KJ, Hu ZD, Fan BT (2006) QSAR study of 1,4-dihydropyridine calcium channel antagonists based on gene expression programming. *Bioorganic & Medicinal Chemistry* 14(14):4834–4841, DOI doi: 10.1016/j.bmc.2006.03.019
 352. Siegel EV (1994) Competitively evolving decision trees against fixed training cases for natural language processing. In: Kinnear, Jr KE (ed) *Advances in Genetic Programming*, MIT Press, chap 19, pp 409–423, URL http://www1.cs.columbia.edu/nlp/papers/1994/siegel_94.pdf
 353. Sims K (1991) Artificial evolution for computer graphics. *ACM Computer Graphics* 25(4):319–328, URL <http://delivery.acm.org/10.1145/130000/122752/p319-sims.pdf>, SIGGRAPH '91 Proceedings
 354. Smart W, Zhang M (2004) Applying online gradient descent search to genetic programming for object recognition. In: Hogan J, Montague P, Purvis M, Stekette C (eds) *CRPIT '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, Australian Computer Society, Inc., Dunedin, New Zealand, vol 32 no. 7, pp 133–138, URL <http://crpit.com/confpapers/CRPITV32Smart.pdf>
 355. Soule T, Foster JA (1998a) Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation* 6(4):293–309, URL <http://mitpress.mit.edu/journals/EVCO/Soule.pdf>
 356. Soule T, Foster JA (1998b) Removal bias: a new cause of code growth in tree based evolutionary programming. In: 1998 IEEE International Conference on Evolutionary Computation, IEEE Press, Anchorage, Alaska, USA, pp 781–186, URL <http://citeseer.ist.psu.edu/313655.html>
 357. Spector L (2001) Autoconstructive evolution: Push, pushGP, and pushpop. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt HM, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, Morgan Kaufmann, San Francisco, California, USA, pp 137–146, URL <http://hampshire.edu/lspector/pubs/ace.pdf>
 358. Spector L (2004) *Automatic Quantum Computer Programming: A Genetic Programming Approach*, Genetic Programming, vol 7. Kluwer Academic Publishers, Boston/Dordrecht/New York/London, URL <http://www.wkap.nl/prod/b/1-4020-7894-3>
 359. Spector L, Alpern A (1994) Criticism, culture, and the automatic generation of artworks. In: *Proceedings of Twelfth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Seattle, Washington, USA, pp 3–8
 360. Spector L, Alpern A (1995) Induction and recapitulation of deep musical structure. In: *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI'95 Workshop on Music and AI*, Montreal, Quebec, Canada, URL <http://hampshire.edu/lspector/pubs/IJCAI95mus-toappear.ps>

361. Spector L, Barnum H, Bernstein HJ (1998) Genetic programming for quantum computers. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) Genetic Programming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA, pp 365–373
362. Spector L, Barnum H, Bernstein HJ, Swamy N (1999a) Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In: Angeline PJ, Michalewicz Z, Schoenauer M, Yao X, Zalzala A (eds) Proceedings of the Congress on Evolutionary Computation, IEEE Press, Mayflower Hotel, Washington D.C., USA, vol 3, pp 2239–2246, URL <http://hampshire.edu/~lasCCS/pubs/spector-cec99.ps>
363. Spector L, Langdon WB, O'Reilly UM, Angeline PJ (eds) (1999b) Advances in Genetic Programming 3. MIT Press, Cambridge, MA, USA, URL <http://www.cs.bham.ac.uk/~wbl/aigp3>
364. Spector L, Klein J, Keijzer M (2005) The push3 execution stack and the evolution of control. In: Beyer HG, O'Reilly UM, Arnold DV, Banzhaf W, Blum C, Bonabeau EW, Cantu-Paz E, Dasgupta D, Deb K, Foster JA, de Jong ED, Lipson H, Llorca X, Mancoridis S, Pelikan M, Raidl GR, Soule T, Tyrrell AM, Watson JP, Zitzler E (eds) GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM Press, Washington DC, USA, vol 2, pp 1689–1696, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005/docs/p1689.pdf>
365. Stender J (ed) (1993) Parallel Genetic Algorithms: Theory and Applications. IOS press
366. Stephens CR, Waelbroeck H (1997) Effective degrees of freedom in genetic algorithms and the block hypothesis. In: Bäck T (ed) Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97), Morgan Kaufmann, East Lansing, pp 34–40
367. Stephens CR, Waelbroeck H (1999) Schemata evolution and building blocks. *Evolutionary Computation* 7(2):109–124
368. Sterling T (1998) Beowulf-class clustered computing: Harnessing the power of parallelism in a pile of PCs. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) Genetic Programming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA, p 883, invited talk
369. Szymanski JJ, Brumby SP, Pope P, Eads D, Esch-Mosher D, Galassi M, Harvey NR, McCulloch HDW, Perkins SJ, Porter R, Theiler J, Young AC, Bloch JJ, David N (2002) Feature extraction from multiple data sources using genetic programming. In: Shen SS, Lewis PE (eds) Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery VIII, SPIE, vol 4725, pp 338–345, URL <http://www.cs.rit.edu/~dre9227/papers/szymanskiSPIE4725.pdf>
370. Tackett WA (1993) Genetic generation of “dendritic” trees for image classification. In: Proceedings of WCNN93, IEEE Press, pp IV 646–649, URL <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/GP.feature.discovery.ps.Z>
371. Takagi H (2001) Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE* 89(9):1275–1296, invited Paper

372. Tanev I, Uozumi T, Akhmetov D (2004) Component object based single system image for dependable implementation of genetic programming on clusters. *Cluster Computing Journal* 7(4):347–356, DOI doi:10.1023/B:CLUS.0000039494.39217.c1, URL <http://www.kluweronline.com/issn/1386-7857>
373. Taylor J, Goodacre R, Wade WG, Rowland JJ, Kell DB (1998) The deconvolution of pyrolysis mass spectra using genetic programming: application to the identification of some eubacterium species. *FEMS Microbiology Letters* 160:237–246, DOI doi:10.1016/S0378-1097(98)00038-X
374. Teller A (1994) Genetic programming, indexed memory, the halting problem, and other curiosities. In: *Proceedings of the 7th annual Florida Artificial Intelligence Research Symposium*, IEEE Press, Pensacola, Florida, USA, pp 270–274, URL <http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/Curiosities.ps>
375. Teller A (1996) Evolving programmers: The co-evolution of intelligent recombination operators. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 3, pp 45–68, URL <http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/AiGP2.ps>
376. Teller A, Andre D (1997) Automatically choosing the number of fitness cases: The rational allocation of trials. In: Koza JR, Deb K, Dorigo M, Fogel DB, Garzon M, Iba H, Riolo RL (eds) *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Morgan Kaufmann, Stanford University, CA, USA, pp 321–328, URL <http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/GR.ps>
377. Teredesai A, Govindaraju V (2005) GP-based secondary classifiers. *Pattern Recognition* 38(4):505–512, DOI doi:10.1016/j.patcog.2004.06.010
378. Theiler JP, Harvey NR, Brumby SP, Szymanski JJ, Alferink S, Perkins SJ, Porter RB, Bloch JJ (1999) Evolving retrieval algorithms with a genetic programming scheme. In: Descour MR, Shen SS (eds) *Proceedings of SPIE 3753 Imaging Spectrometry V*, pp 416–425, URL <http://public.lanl.gov/jt/Papers/ga-spie.ps>
379. Todd PM, Werner GM (1999) Frankensteinian approaches to evolutionary music composition. In: Griffith N, Todd PM (eds) *Musical Networks: Parallel Distributed Perception and Performance*, MIT Press, pp 313–340, URL <http://www-abc.mpib-berlin.mpg.de/users/ptodd/publications/99evmus/99evmus.pdf>
380. Tomassini M, Luthi L, Giacobini M, Langdon WB (2007) The structure of the genetic programming collaboration network. *Genetic Programming and Evolvable Machines* 8(1):97–103, DOI doi:10.1007/s10710-006-9018-2
381. Trujillo L, Olague G (2006a) Synthesis of interest point detectors through genetic programming. In: Keijzer M, Cattolico M, Arnold D, Babovic V, Blum C, Bosman P, Butz MV, Coello Coello C, Dasgupta D, Ficici SG, Foster J, Hernandez-Aguirre A, Hornby G, Lipson H, McMinn P, Moore J, Raidl G, Rothlauf F, Ryan C, Thierens D (eds) *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ACM Press, Seattle, Washington, USA, vol 1, pp 887–894, DOI doi:10.1145/1143997.1144151, URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2006/docs/p887.pdf>
382. Trujillo L, Olague G (2006b) Using evolution to learn how to perform interest point detection. In: et al XYT (ed) *ICPR 2006 18th International Conference on Pattern Recognition*, IEEE, vol 1, pp 211–214, DOI doi:10.1109/ICPR.2006.1153, URL <http://www.genetic-programming.org/hc2006/Olague-Paper-2-ICPR%-2006.pdf>

383. Tsang EPK, Li J, Butler JM (1998) EDDIE beats the bookies. *Software: Practice and Experience* 28(10):1033–1043, DOI doi:10.1002/(SICI)1097-024X(199808)28:10<1033::AID-SPE198>3.0.CO;2--1, URL <http://cswwww.essex.ac.uk/CSP/finance/papers/TsBuLi-Eddie-Software98.pdf>
384. Turing AM (1948) *Intelligent machinery*, report for National Physical Laboratory. Reprinted in Ince, D. C. (editor). 1992. *Mechanical Intelligence: Collected Works of A. M. Turing*. Amsterdam: North Holland. Pages 107127. Also reprinted in Meltzer, B. and Michie, D. (editors). 1969. *Machine Intelligence r5*. Edinburgh: Edinburgh University Press
385. Turing AM (1950) Computing machinery and intelligence. *Mind* 49:433–460, URL <http://www.cs.umbc.edu/471/papers/turing.pdf>
386. Usman I, Khan A, Chamlawi R, Majid A (2007) Image authenticity and perceptual optimization via genetic algorithm and a dependence neighborhood. *International Journal of Applied Mathematics and Computer Sciences* 4(1):615–620, URL <http://www.waset.org/ijamcs/v4/v4-1-7.pdf>
387. Vaidyanathan S, Broadhurst DI, Kell DB, Goodacre R (2003) Explanatory optimization of protein mass spectrometry via genetic search. *Analytical Chemistry* 75(23):6679–6686, DOI doi:10.1021/ac034669a, URL [http://dbkggroup.org/Papers/AnalChem75\(6679-6686\).pdf](http://dbkggroup.org/Papers/AnalChem75(6679-6686).pdf)
388. Venkatraman V, Dalby AR, Yang ZR (2004) Evaluation of mutual information and genetic programming for feature selection in QSAR. *Journal of Chemical Information and Modeling* 44(5):1686–1692, DOI doi:10.1021/ci049933v
389. Vowk B, Wait AS, Schmidt C (2004) An evolutionary approach generates human competitive coreware programs. In: Bedau M, Husbands P, Hutton T, Kumar S, Suzuki H (eds) *Workshop and Tutorial Proceedings Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife XI)*, Boston, Massachusetts, pp 33–36, *artificial Chemistry and its applications workshop*
390. Vukusic I, Greltscheid SN, Wiehe T (2007) Applying genetic programming to the prediction of alternative mRNA splice variants. *Genomics* 89(4):471–479, DOI doi:10.1016/j.ygeno.2007.01.001
391. Walker RL (2001) Search engine case study: searching the web using genetic programming and MPI. *Parallel Computing* 27(1–2):71–89, URL <http://www.sciencedirect.com/science/article/B6V12-42K5HNX-4/1/57eb870c72fb7768bb7d824557444b72>
392. Walsh P, Ryan C (1996) Paragen: A novel technique for the autoparallelisation of sequential programs using genetic programming. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, Stanford University, CA, USA, pp 406–409, URL <http://cognet.mit.edu/library/books/view?isbn=0262611279>
393. Weaver DC (2004) Applying data mining techniques to library design, lead generation and lead optimization. *Current Opinion in Chemical Biology* 8(3):264–270, DOI doi:10.1016/j.cbpa.2004.04.005, URL <http://www.sciencedirect.com/science/article/B6V12-42K5HNX-4/1/57eb870c72fb7768bb7d824557444b72>
394. Whigham PA (1995) A schema theorem for context-free grammars. In: 1995 IEEE Conference on Evolutionary Computation, IEEE Press, Perth, Australia, vol 1, pp 178–181
395. Whigham PA (1996) Search bias, language bias, and genetic programming. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) *Genetic Programming 1996:*

- Proceedings of the First Annual Conference, MIT Press, Stanford University, CA, USA, pp 230–237
396. Whitley D (2001) An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology* 43(14):817–831, DOI doi:10.1016/S0950-5849(01)00188-4, URL <http://www.cs.colostate.edu/~genitor/2001/overview.pdf>
 397. Whitley LD (1994) A Genetic Algorithm Tutorial. *Statistics and Computing* 4:65–85
 398. Willis M, Hiden H, Marenbach P, McKay B, Montague GA (1997a) Genetic programming: An introduction and survey of applications. In: Zalzal A (ed) *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, GALESIA, Institution of Electrical Engineers, University of Strathclyde, Glasgow, UK, URL <http://www.staff.ncl.ac.uk/d.p.searson/docs/galesia97surveyofGP.pdf>
 399. Willis MJ, Hiden HG, Montague GA (1997b) Developing inferential estimation algorithms using genetic programming. In: *IFAC/ADCHEM International Symposium on Advanced Control of Chemical Processes*, Banaff, Canada, pp 219–224
 400. Wilson G, Heywood M (2007) Introducing probabilistic adaptive mapping developmental genetic programming with redundant mappings. *Genetic Programming and Evolvable Machines* 8(2):187–220, DOI doi:10.1007/s10710-007-9027-9, special issue on developmental systems
 401. Wong ML (1998) An adaptive knowledge-acquisition system using generic genetic programming. *Expert Systems with Applications* 15(1):47–58, URL <http://cptra.ln.edu.hk/~mlwong/journal/esa1998.pdf>
 402. Wong ML (2005) Evolving recursive programs by using adaptive grammar based genetic programming. *Genetic Programming and Evolvable Machines* 6(4):421–455, DOI doi:10.1007/s10710-005-4805-8, URL <http://cptra.ln.edu.hk/~mlwong/journal/gpem2005.pdf>
 403. Wong ML, Leung KS (1995) Inducing logic programs with genetic algorithms: the genetic logicprogramming system genetic logic programming and applications. *IEEE Expert* 10(5):68–76, DOI doi:10.1109/64.464935
 404. Wong ML, Leung KS (1996) Evolving recursive functions for the even-parity problem using genetic programming. In: Angeline PJ, Kinnear, Jr KE (eds) *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chap 11, pp 221–240
 405. Wong ML, Leung KS (2000) *Data Mining Using Grammar Based Genetic Programming and Applications*, Genetic Programming, vol 3. Kluwer Academic Publishers
 406. Wong ML, Wong TT, Fok KL (2005) Parallel evolutionary algorithms on graphics processing unit. In: Corne D, Michalewicz Z, McKay B, Eiben G, Fogel D, Fonseca C, Greenwood G, Raidl G, Tan KC, Zalzal A (eds) *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, IEEE Press, Edinburgh, Scotland, UK, vol 3, pp 2286–2293, URL <http://ieeexplore.ieee.org/servlet/opac?punumber=10417&isvol=3>
 407. Woodward AM, Gilbert RJ, Kell DB (1999) Genetic programming as an analytical tool for non-linear dielectric spectroscopy. *Bioelectrochemistry and Bioenergetics* 48(2):389–396, DOI doi:10.1016/S0302-4598(99)00022-7, URL <http://www.sciencedirect.com/science/article/B6TF7-3WJ72RJ-T/2/19fd01a6eb6ae0b8e12b2bb2218fb6e9>

408. Wright S (1932) The roles of mutation, inbreeding, crossbreeding and selection in evolution. In: Jones DF (ed) Proceedings of the Sixth International Congress on Genetics, vol 1, pp 356–366
409. Xie H, Zhang M, Andrae P (2006) Genetic programming for automatic stress detection in spoken english. In: Rothlauf F, Branke J, Cagnoni S, Costa E, Cotta C, Drechsler R, Lutton E, Machado P, Moore JH, Romero J, Smith GD, Squillero G, Takagi H (eds) Applications of Evolutionary Computing, EvoWorkshops2006: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoInteraction, EvoMUSART, EvoSTOC, Springer Verlag, Budapest, LNCS, vol 3907, pp 460–471, DOI doi:10.1007/11732242_41, URL <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=3907&spage=460>
410. Yangiya M (1995) Efficient genetic programming based on binary decision diagrams. In: 1995 IEEE Conference on Evolutionary Computation, IEEE Press, Perth, Australia, vol 1, pp 234–239
411. Yu J, Bhanu B (2006) Evolutionary feature synthesis for facial expression recognition. Pattern Recognition Letters 27(11):1289–1298, DOI doi:10.1016/j.patrec.2005.07.026, evolutionary Computer Vision and Image Understanding
412. Yu J, Yu J, Almal AA, Dhanasekaran SM, Ghosh D, Worzel WP, Chinnaiyan AM (2007) Feature selection and molecular classification of cancer using genetic programming. Neoplasia 9(4):292–303, DOI doi:10.1593/neo.07121
413. Yu T (2001) Hierarchical processing for evolving recursive and modular programs using higher order functions and lambda abstractions. Genetic Programming and Evolvable Machines 2(4):345–380, DOI doi:10.1023/A:1012926821302
414. Yu T, Chen SH (2004) Using genetic programming with lambda abstraction to find technical trading rules. In: Computing in Economics and Finance, University of Amsterdam
415. Yu T, Riolo RL, Worzel B (eds) (2005) Genetic Programming Theory and Practice III, Genetic Programming, vol 9, Springer, Ann Arbor
416. Zhang BT, Mühlenbein H (1993) Evolving optimal neural networks using genetic algorithms with Occam’s razor. Complex Systems 7:199–220, URL <http://citeseer.ist.psu.edu/zhang93evolving.html>
417. Zhang BT, Mühlenbein H (1995) Balancing accuracy and parsimony in genetic programming. Evolutionary Computation 3(1):17–38, URL http://www.ais.fraunhofer.de/~muehlen/publications/gmd_as_ga-94_09.ps
418. Zhang BT, Ohm P, Mühlenbein H (1997) Evolutionary induction of sparse neural trees. Evolutionary Computation 5(2):213–236, URL <http://bi.snu.ac.kr/Publications/Journals/International/EC5-2.ps>
419. Zhang M, Smart W (2006) Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. Pattern Recognition Letters 27(11):1266–1274, DOI doi:10.1016/j.patrec.2005.07.024, evolutionary Computer Vision and Image Understanding
420. Zhang Y, Rockett PI (2006) Feature extraction using multi-objective genetic programming. In: Jin Y (ed) Multi-Objective Machine Learning, Studies in Computational Intelligence, vol 16, Springer, chap 4, pp 79–106, invited chapter

Resources

Following the publication of [188], the field of GP took off in about 1990 with a period of exponential growth common in the initial stages of successful technologies. Many influential initial papers from that period can be found in the proceedings of the *Intl. Conf. Genetic Algorithms* (ICGA-93, ICGA-95), the *IEEE Confs. on Evolutionary Computation* (EC-1994), and the *Evolutionary Programming Conference*. A surprisingly large number of these are now available online. After almost twenty years, GP has matured and is used in a wondrous array of applications. From banking [265] to betting [383], from bomb detection [102] to architecture [282], from the steel industry to the environment [157], from space [234] to biology [159], and many others (as we have seen in Sect. 7). In 1996 it was possible to list (almost all) GP applications [201], but today the range is far too great, so here we simply list some GP resources, which, we hope, will guide readers towards their goals.

1 Key Books

There are today more than 31 books written in English principally on GP or its applications, with more being written. These start with John Koza's 1992 *Genetic Programming* (often referred to as 'Jaws'). Koza has published four books on GP: *Genetic Programming II: Automatic Discovery of Reusable Programs* (1994) deals with ADFs; *Genetic Programming 3* (1999) covers, in particular, the evolution of analogue circuits; *Genetic Programming 4* (2003) uses GP for automatic invention.

MIT Press published three volumes in the series *Advances in Genetic Programming* (1994, 1996, 1999).

The joint GP/genetic algorithms Kluwer book series edited by Koza and Goldberg now contains 14 books, starting with *Genetic Programming and*

Data Structures [203]. Apart from ‘Jaws’, these tend to be for the GP specialist.

1997 saw the introduction of the first textbook dedicated to GP [25].

Eiben [87] and Goldberg [115] provide general treatment on evolutionary algorithms.

Other titles include: *Principia Evolvica – Simulierte Evolution mit Mathematica* (in German) [154] (English version [156]), *Data Mining Using Grammar Based Genetic Programming and Applications* [405], *Genetic Programming* (in Japanese) [151], and *Humanoider: Sjavlarande robotar och artificiell intelligens* (in Swedish) [274].

Readers interested in mathematical and empirical analyses of GP behavior may find *Foundations of Genetic Programming* [222] useful.

2 Videos

Each of Koza’s four books has an accompanying illustrative video. These are now available in DVD format. Furthermore, a small set of videos on specific GP techniques and applications is available from [Google Video](#) and [YouTube](#).

3 Key Journals

In addition to GP’s own *Genetic Programming and Evolvable Machines* journal (Kluwer), *Evolutionary Computation*, the *IEEE Trans. Evolutionary Computation*, *Complex Systems* (Complex Systems Publication, Inc.), and many others publish GP articles. The *GP bibliography* (<http://www.cs.bham.ac.uk/~wbl/biblio/>) lists a further 375 different journals worldwide that have published articles related to GP.

4 Key International Conferences/Workshops

EuroGP has been held every year since 1998. All *EuroGP* papers are available on line as part of Springer’s LNCS series. The original annual *Genetic Programming* conference was hosted by Koza in 1996 at Stanford. Since 1999 it has been combined with the *Intl. Conf. Genetic Algorithms* to form *GECCO*; 98% of *GECCO* papers are available online. The Michigan-based *Genetic Programming Theory and Practice Workshop* [284,322,323,415] will shortly publish its fifth proceedings [324]. Other EC conferences, such as *CEC*, *PPSN*, *Evolution Artificielle*, and *WSC*, also regularly contain GP papers.

5 Online Resources

One of the reasons behind the success of GP is that it is easy to implement your own version. People have coded GP in a huge range of different languages, such as Lisp, C, C++, Java, JavaScript, Perl, Prolog, Mathematica, Pop-11, MATLAB, Fortran, Occam and Haskell. Typically these evolve code which looks like a very cut down version of Lisp. However, admirers of grammars claim the evolved language can be arbitrarily complex, and certainly programs in functional and other high level languages have been automatically evolved. Conversely, many successful programs in machine code or low-level languages have also climbed from the primordial ooze of initial randomness.

Many GP implementations can be freely downloaded. Two that have been available for a long time and remain popular are: Sean Luke's ECJ (in Java), and Douglas Zongker's 'little GP' `lilGP` (in C). A number of older (unsupported) tools can be found at <ftp://cs.ucl.ac.uk/genetic/ftp.io.com/>. The most prominent commercial implementation remains *Discipulus* [99].

There is a lot of information available on the world wide web, although, unfortunately, Internet addresses (URLs) change rapidly. Therefore we simply name useful pages here (rather than give their URL). A web search will usually quickly locate them.

At the time of writing, the *GP bibliography* contains about 5000 GP entries. About half the entries can be downloaded immediately. There are a variety of interfaces including a graphical representation of GP's collaborative network (see Fig. 1). The HTML pages are perhaps the easiest to use. They allow quick jumps between papers linked by authors, show paper concentrations and in many cases direct paper downloads. The collection of computer sciences bibliographies provides a comprehensive Lucene syntax search engine. Bibtex and Refer files can also be searched but are primarily intended for direct inclusion of bibliographic references in papers written in LaTeX and Microsoft Word, respectively.

Almost since the beginning there has been an open active email discussion list: the *GP discussion group*, which is hosted by Yahoo! For more reflective discussions, the *EC-Digest* comes out once a fortnight and often contains GP related announcements, while the organization behind *GECCO* also runs a quarterly *SIGEvolution* newsletter.

Koza's <http://www.genetic-programming.org/> contains a ton of useful information for the novice, including a short tutorial on 'What is Genetic Programming', as well as LISP code for implementing GP, as in [188].

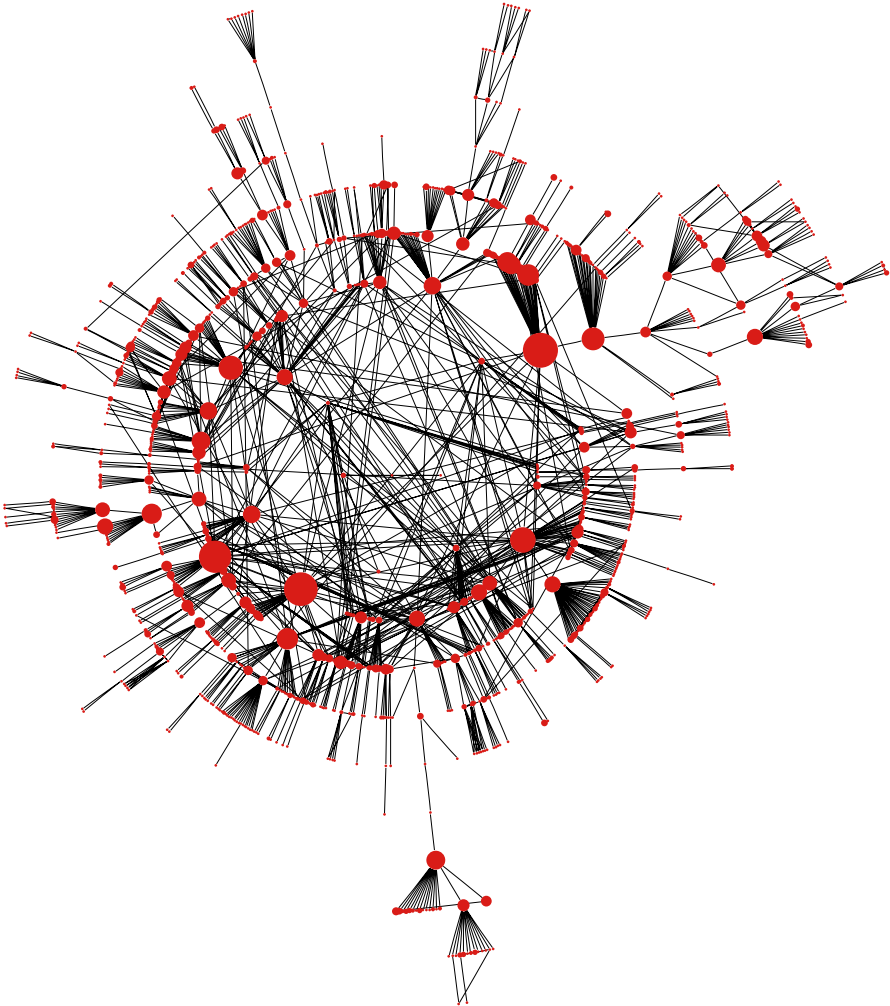


Fig. 1. Co-author connections within GP. Each of the 1141 dots indicates an author. The lines link people who have co-authored one or more papers. (To reduce clutter only links to first authors are shown.) The online version is annotated by JavaScript and contains hyperlinks to authors and their GP papers. The graph was created by GraphViz `twopi`, which tries to place strongly connected people close together. It is the ‘centrally connected component’ [380] and contains approximately half of all GP papers. The remaining papers are not linked by co-authorship to this graph. Several of the larger unconnected graphs are also available online via the *gp-bibliography* web pages

The Particle Swarm Algorithm

Tim Hendtlass

Centre for Information Technology Research, Swinburne University of Technology,
Hawthorn VIC 3125, Australia, thendtlass@swin.edu.au

1 Introduction

Many algorithms are the result of biological inspiration and particle swarm optimization (PSO) is no exception. However, the PSO algorithm has slightly different end goals to the biological behavior that provides its inspiration and so needs to differ from its biological inspiration in some, perhaps non-biological, ways. PSO takes its inspiration from the flocking of birds and fish. In the real world, the flock needs to be compact for protection, and once food is found the flock should settle to feed. In the artificial particle swarm optimization the aim of the algorithm is to find an optimum solution to some problem, rather than the protection or food sought in the natural environment. For PSO the correct behavior once an optimum is found is not for all the particles in the swarm to converge on this, possibly local, optimum as the goal is to check *many* optima in the hope of finding the global optimum. Instead of converging, once an optimum has been found, it should be noted and the particles should immediately disperse to look for another, perhaps better, optimum. In Nature the time will come when a swarm that is feeding has consumed the food so that the place is no longer optimal: if swarming for protection the threat may change or even disappear completely. Then the swarm will again set out. Such extended periods of convergence serve no useful purpose so far as an artificial swarm is concerned.

If we model our PSO algorithm too closely on the behavior of birds and fish we run the risk that we will achieve those aspects of the natural behavior that we don't want at the expense of the artificial behavior that we *do* want. While retaining (possibly modified versions of) components that give the natural swarm its efficient search capability, we should be prepared to add such non-biological components as necessary so as to modify the natural behavior into the type of behavior we desire. This Chapter is concerned with developing the ideas behind a range of PSO algorithms, ranging from the simple (which

loosely models real life and so has substantially real life behavior) through a series of progressively less biologically plausible algorithms that enable us to achieve useful and practical, but un-biological, aims. The basics are the same for all these algorithms. Since the extra abilities generally come at extra computational cost, this Chapter will describe the strengths and limitations of each so that the reader can make an informed choice as to which might be best for any particular problem.

2 The Basic Particle Swarm Optimization Algorithm

In all particle swarm algorithms the position of a particular particle in some way encodes a possible solution to the problem for which we seek, ideally an optimal, but at least a very good solution. Particles move under the combined influence of a number of factors in such a way that they tend to converge towards an optimum. A number of slight variants of the particle swarm algorithm have been proposed since the original algorithm was introduced by [19]. All of these try to balance several aspects of the behavior. See, for example, [5, 15, 23]

For an optimization algorithm to be more efficient than a random search the choice of the next position to be searched must be guided by the positions previously tested, and how good a solution those positions represented. Some way must be found that allows this previous knowledge to be exploited so that, on average, the next positions explored represent better solutions than would have been found if the next positions had just been picked randomly. Obviously storing all positions previously explored would rapidly produce an unsupportable demand for memory, and manipulating them in any way an insupportable computational load. So in the PSO only a few – often only two – kinds of good positions are kept by each particle, and probably some (and possibly all) the information about a particular particle’s stored positions is shared with some or all of the other members of the swarm. There are a number of candidates for the types of position to keep:

1. All particles try to exploit (are influenced to move towards) at least one good position already found by some particle in the swarm. Often the position that is exploited is the best position yet found by any member of the swarm, a position known as the ‘global best’ or *gbest* position. This obviously requires communication between the members of the swarm sharing the best position each has found and forms a sort of social collective memory of the current global best *gbest* position.
2. Rather than using all the members of the swarm, sometimes each particle uses the best position – *lbest* – found by some subset of the swarm, the subset to which this particle belongs. Often this is defined as the N physically closest particles (in problem space) to this particle, but also the N closest in terms of index can be used. For example, if this particle has index 7 in

the list of particles, particles 5, 6, 8 and 9 might be the other members of its subset. Although for this latter option the particles may be far apart in problem space initially, attraction between the subset members has the effect of moving them closer together over time so that the net effect is quite similar to that of using the N physically closest particles. Not having to calculate the distance between each pair of particles can be a significant saving in time. Either the best current or the best position ever found by all the particles that make up the subset can be used. While this information is only directly shared among members of the subset, each particle can be a member of many subsets and so information in time is spread across the swarm.

3. The last position commonly used by a particle is the best position yet found by this particle – *pbest*. Since each particle is attracted to its own personal best position this does not depend on any inter-particle communication at all (although of course this is the same information that is shared as part of the calculation of whichever of *gbest* or *lbest* is also being used).

All particles are assumed to have mass so that they cannot change direction instantaneously. Furthermore, each time their velocity is updated it must contain a component that is in the same direction as their previously calculated velocity. This effectively provides a low pass filter to any change in their velocity, and smooths the track the particle follows through the problem space. The fraction of the previous component that is retained is called the ‘momentum’ and is a number greater than zero but less than one.

With only the use of *momentum* and *gbest*, particles would engage in a headlong rush toward the first reasonable position found by any particle, only changing this if some particle happens upon a better position during this rush. Eventually all particles would reach the same best-known position and exploration would stop. The particles would in time come to rest as, once a particle overshoot a position, it would be attracted back to it and, with the momentum term being less than one, the velocity would drop with each reversal. This behavior would mimic real life birds settling at the best-known food source, but little exploration would occur during the headlong convergence and this would represent the balance between exploration and exploitation being tipped firmly towards exploitation.

With only the use of momentum and *lbest*, the behavior would depend on whether each particle is sharing the current best or the best ever position it has found. If the best ever is being kept the swarm could tend to divide and small groups of particles converge on a number of different positions. Particularly in the early stages the exact membership of the subsets would change, quite possibly changing the position towards which the current subgroup’s particles are being attracted. This is less social than using *gbest* and more exploration will occur as the sub-swarms move towards their (generally) individual convergence positions. The balance between exploration and exploitation is still tipped strongly towards exploitation, but not as strongly as using *momentum* and *gbest*.

If using *momentum* and *lbest* and the position recorded is the current best, then the position that the particles in the sub-swarm are converging on is likely to change often, but can as easily change to a lesser fitness as to a greater fitness. With no collective memory of good positions found, the final position where the swarm members end up is largely a matter of chance. The balance between exploration and exploitation is now tipped quite strongly towards exploration, but some small exploitation occurs as the particles that make up a sub swarm share their current but not their historic information.

With only the use of *momentum* and *pbest*, each particle will end up on the best position it has happened across during its travels. While this obviously means that there will be some, probably many, positions explored by the swarm with poorer fitness than those positions on which the swarm members finally end up, there is no guarantee that the final positions will be good in a global sense. This is pure exploration with very little attempt at exploitation at all.

It is when *momentum* is combined with an attraction to two of these positions that the performance of the swarm improves sufficiently to make it attractive as a practical optimization algorithm. Which two you choose determines how greedy the algorithm would be.¹ A very greedy algorithm will converge fast, but not necessarily to a good position: a less greedy algorithm will converge more slowly, possibly *much* more slowly, but the probability that it will converge to a good position is enhanced.

The general form of the equation that governs the updating of each particle's velocity is given as:

$$\begin{aligned} \bar{V}_{T+t} = M \times \bar{V}_T + (1 - M) \times & \left(\left(\frac{G \times R_1 \times (\overline{Gbest} - \bar{X}_T)}{t} \right) \right. \\ & \left. + \left(\frac{L \times R_2 \times (\overline{Lbest} - \bar{X}_T)}{t} \right) \right) \end{aligned} \tag{1}$$

or

$$\begin{aligned} \bar{V}_{T+t} = M \times \bar{V}_T + (1 - M) \times & \left(\left(\frac{G \times R_1 \times (\overline{Gbest} - \bar{X}_T)}{t \times |\overline{Gbest} - \bar{X}_T|} \right) \right) \\ & + \left(\frac{L \times R_2 \times (\overline{Lbest} - \bar{X}_T)}{t \times |\overline{Lbest} - \bar{X}_T|} \right) \end{aligned} \tag{2}$$

In both of these equations, the parameter *M* is the momentum of the particle [0..1] and controls how fast the particle is able to respond to the

¹ A very greedy algorithm is one which attempts to optimize each step without regard to the final result; a less greedy algorithm is prepared to make one or more less optimal steps for now in the hope that these would set it up to make a 'killer' move later on.

two positions of attraction. A high momentum will make the particle slow to respond, which encourages exploration (and may enable the particle to move through small local optima entirely); conversely a low momentum makes the particle very quick to respond. Each of the two points of attraction has a constant that sets its maximum importance (G and L , respectively) but the two random numbers (R_1 and R_2 both in the range 0..1) introduce a stochastic element into the actual attraction at any time. The introduction of the $1/t$ terms is to ensure that the dimensions of each part of the equation are the same, namely the dimensions of velocity. The units of time are usually arbitrary, allowing t to be set to one and so saving it having to be written, but this time (whatever its value) is of great significance and has been included in Eqn. (1) deliberately so it is not overlooked. It must be remembered that, unlike real life, the algorithm assumes that a particle's velocity remains unchanged between fitness evaluations. A particle may travel a considerable distance, and even pass through several possible optima, between evaluations if t is large. Further implications of using a finite value for t will be discussed below.

Commonly the attraction to each position is also made dependent on the distance the particle is from the point of attraction: the further the particle is away, the higher the attraction. This is the form of the above update equations. While this can work well in simple problem spaces it may become unhelpful in more complex problem spaces. When a particle is far from a point of attraction, the acceleration of the particle towards that point of attraction would be high. By the time the particle reaches the point of attraction the particle's speed would be very high and so the distance it travels between evaluations is also large. An alternate approach (as expressed in Eqn. (1)/(2)) is to make the attraction to a point independent of the distance it is from that point. Although a particle will still accelerate as a result of the continued attraction to a point, this approach has the effect of limiting the maximum velocity particles can have and thus the maximum distance it can travel between fitness evaluations.

A limiting speed may be introduced for use with either version of the update equation. No particle is permitted to go faster than this limiting speed, if the magnitude of the right hand side of Eqns. (1) or (2) is greater than this limiting speed, the new speed of the particle is set to this limiting value but with the direction calculated from the equation. However the initial acceleration will be faster than if Eqn. (1)(Eqn. (2)) as stated above is used, and more of the journey will tend to occur at this limiting velocity, reducing the number of fitness evaluations along the journey.

To complete the description of Eqn. (1)(Eqn. (2)) it only remains to discuss which pair of attraction points to use. Originally, $gbest$ and $pbest$ were used, the former providing the social exploitation and the latter the personal exploration. Replacing $pbest$ with $lbest$ is a viable alternative as long as the size of the neighbourhood remains a modest fraction (say 10%) of the total

swarm. If this is done the algorithm is made slightly greedier but a significant amount of exploration is still undertaken. Using *lbest* and *pbest* is the least greedy version of all and provides the least exploitation and the most exploration. The swarm may finally converge owing to the overlap between the sub-swarms of each of the particles.

Having calculated the new velocity for each particle the only other step in the algorithm is to move each particle ready for the next iteration. As mentioned above, it is assumed that the velocity of the particle does not change during the time t between updates (and evaluations). The position update equation is given as Eqn. (3), again with the inter-evaluation time t explicitly shown so as to make the equation dimensionally consistent.

$$\overline{X}_{T+t} = \overline{X}_T + t \times \overline{V}_{T+t} \tag{3}$$

2.1 Pseudo Code Algorithm for the Basic PSO

Algorithm 4 Basic Particle Swarm Optimization Algorithm

1. Randomly assign each particle to some point in problem space with some (small) random velocity and evaluate the fitness of each particle at its current position. **while** no stopping condition is met **do**
 2. Update *gbest* for the swarm, *pbest* for each particle (if these are to be used). **if** *lbest* is to be used **then**
 - find the other particles that make up the sub-swarm of each particle, and update *lbest* for each particle from either the current best or the personal best fitness of it and its neighbours.
 3. Calculate the new velocity of each particle using Eqn. (1)/(2) (if not using either *gbest* or *lbest*, replace then with the position you are using).
 4. Move each particle using Eqn. (3) and evaluate its fitness at this new position.
-

The stopping condition could be achieving acceptable performance OR the swarm having converged without achieving adequate performance OR some maximum number of fitness evaluations having been made without either of the first two conditions being met. A swarm has converged when all the particles are (eventually) at the same position and the velocity of each particle is approaching zero.

3 Enhancements to the Basic Particle Swarm Algorithm

3.1 Constriction Factors

The whole right side of Eqns. (1) and (2) can be multiplied by a constriction factor. The purpose of this factor is to drop the average speed of the particles as time goes on. By doing this one of the problems with using a finite time between updates (t) can be addressed. If a particle is travelling fast it can

cover a considerable distance in problem space in this time and may well pass over a point of interest without observing it, as no evaluation was made. This risk may be acceptable when the swarm is dispersed as the aim is to find regions of interest and there are probably other reasonably interesting points in the vicinity of this unobserved one. However, this is not so acceptable when the swarm is settling and trying to find the best point in a more restricted region as it will result in many unnecessary crossings and re-crossings of the best point thus increasing the time taken to find the best point.

The constriction factor could be just a fixed number less than one, but finding a suitable value for this number *a priori* is not easy. A better approach [8,9,11] takes into account the current behavior of the swarm and adjusts the constriction factor accordingly.

3.2 Adding Controlled Diversification

The tendency of a swarm to converge is beneficial when the swarm is exploring in the vicinity of an optimum but may well be counter productive if the swarm as a whole is still in transit searching for a suitable point to explore. Then it would be desirable for the swarm to diverge so as to explore more territory. Comparing the average velocity and the average speed of the swarm can help identify these two situations; if the average speed is significantly higher than the average velocity then it is reasonable to assume that the swarm particles are around a region of interest but approaching it in different directions. However, if the average speed and the average velocity have similar non-zero magnitudes then the swarm is probably sweeping through problem space. In this latter case deliberate measures can be taken to counter the natural tendency of the swarm to converge so as to more efficiently search the problem space – for more details see [17].

3.3 Handling Problem Constraints

Problem constraints can take many forms, for example possibly limited ranges of values for one or more variables or regions of problem space that correspond to non-viable solutions.

The first of these can be addressed by only allowing problem space to be of finite duration in the direction that corresponds to that particular variable. Any movement past the limit results in the particle re-appearing at the beginning. Effectively the axis is no longer an infinite straight line but a circle. This ensures that only permitted values of this variable are ever explored. The formula to update the position is no longer complete in itself; it has to have two conditional clauses added. For example, if the range of values is from 2 to 8, the clauses would be:

- **if** position \geq 8 **then** position = position - 6
- **if** position is $<$ 2 **then** position = position + 6

It might be necessary to apply these conditional clauses repeatedly until the position was within the meaningful range. The calculation of the distance between two particles would now have to take into account the fact that, for this dimension at least, the shortest distance (the one that should be used) could be in either direction around the circular axis. Should the acceptable values for this variable form a series of discontinuous ranges (say, 2 – 5 and 8 – 12) the range of the axis should be set to be the sum of these ranges (3 + 4 = 7 in this example). A mapping needs to be made between the position of the axis and the value this represents before the fitness is evaluated. For the example given here this would be:

- **if** position ≥ 0 and < 3 **then** value = position + 2
- **if** position is ≥ 3 and < 7 **then** value = position + 5

The distance between two particles would be calculated as described above, that is without taking into account the mapping.

The second constraint can be accommodated by a small change to the rule to update the global best and local best positions. A particle can only update these if its value is better AND it is in a region of problem space that the fitness function reports as being feasible. A particle can only include in its neighbourhood other particles that are currently in feasible space. It is possible that a particle itself and all other particles in infeasible space, in this case there is no local best and the contribution of the term involving the local best to the velocity update (Eqn. (1) or (2)) is set to zero. The momentum of the particles, useful in helping them sweep through local optima, will also help them sweep through regions of non-feasible space. Should there be no particle in valid problem space at the first fitness evaluation there will be no global best position and that term in Eqns. (1) and (2) will also be zero, and all particles will move in straight lines. However as soon as some particle finds a region of feasible space it can update and the convergence process will start. A problem space that is mostly infeasible would probably form quite a challenge, but more feasible problem spaces can be handled, as described above.

4 Particle Swarm Optimization of Multiple Optima

The performance of the basic particle swarm algorithm is good for problems with one best global optimum as long as the total number of optima does not get too high; ‘good’ in this context means both in terms of the quality of the results obtained and in the speed with which they are found. The quality of the solutions found for a given problem is comparable with those found by a Genetic Algorithm (GA), but the time take to find them is typically one tenth of that required.

However, not all problems fit into this simple category. Modifications can be made to the basic swarm algorithm to produce variations that are suited to at least some more complex classes of problem.

4.1 Exploring Multiple Optima

Many problems may have a number of optima whose fitness is similar. Allowing the swarm to converge to just one may not be the best move. Firstly there is no guarantee that convergence will occur to the global best optimum, indeed practical considerations may make the choice as to which is the best require considerations that are over and above just the numeric fitness value. Under these conditions we would like the algorithm to find several good optima and allow someone (or something) with knowledge of the broader picture to make the final choice.

An example might help. Suppose that the problem being solved is to find a good manufacturing schedule. When selecting the schedule for Thursday (say), the manager may know that certain members of their workforce would have attended a lively social occasion on Wednesday night. As a result it might be better to give them a lighter load on Thursday morning while they overcame the effects of the previous night. Given a range of possible schedules they might choose one that was likely to have the best practical outcome (given the extraneous factors) rather than chose the one with the best theoretical outcome. There could be many possible outside factors occurring too infrequently to be worth building into the fitness calculation for possible schedules. Better to give a range of good possibilities and let the human use their extra knowledge when making the final choice.

There are two basic approaches: if the number of good optima is small we might like to quickly perform a parallel exploration of these, with subsections of the total swarm each exploring a different optimum. Of course the absolute (and impractical) limit to the number of optima that can be explored in parallel is the number of particles in the swarm. This is impractical as a sub-swarm of one particle cannot use any social exploitation at all. If the number of potential optima that should be explored is large, then rather than exploring them all at once we would explore them in turn (serial exploration). Both of these approaches will be considered in this Section.

4.2 Achieving Parallel Exploration of Several Positions of Interest (niching)

The aim here is for the swarm to break automatically into sub-swarms and for these sub-swarms to converge onto different optima. While easy to state, achieving a practical realization presents many problems. If *gbest* is used the swarm will tend to converge all to one point. If *gbest* is replaced by *lbest* how many optima the swarm finds depends on the degree of overlap between the

local regions from which best is derived. This degree of overlap has proved hard to control.

An alternate, and more successful, approach has been to develop a niching PSO [13: 67–69] that starts with a single main swarm and finishes with a number of smaller independent swarms. The particles in each of the smaller swarms communicate only with other members of its own swarm; there is no communication *between* swarms. The main swarm is generally trained using only the *pbest* position. Once some member of the main swarm is deemed to be in the vicinity of an optimum, a sub-swarm is formed by grouping together a number of particles in close proximity to the potential optimum. These communicate only with each other so as to further explore the optimum – now also using *gbest* attraction. These chosen particles refine the absolute best position, never leaving the vicinity of the optimum. Meanwhile the members of the main swarm that were not chosen for this sub-swarm continue using only *pbest* and look for another optimum (or at least a place that seems worthy of closer study).

Ideally the sub-swarms would never come close and no member of the main swarm would wander into a region being explored by a sub-swarm. In reality, of course, both things do happen. In the first case the two sub-swarms are merged (become aware of the performance of each other’s members), which at the cost of using more particles may provide a more thorough exploration of the local space. In the second case the particle is just recruited to (joins) the sub-swarm it is moving through.

The niching PSO can explore a few different regions but since the particles that form a sub-swarm never leave the region they are exploring, the maximum number of regions that can be explored is set by the size of the original main swarm and by how many particles are recruited to form each sub-swarm. As the number of potential regions that require investigation increases, parallel exploration will obviously become quite inefficient.

4.3 Achieving Serial Exploration of Many Positions of Interest (WoSP)

An alternate approach that uses Waves of Swarm Particles (WoSP), introduced by [14], achieves serial exploration of an, in principle, infinite number of positions of interest. Actually it is not strictly serial as at any given time a small number of regions of interest are typically being explored in parallel. This behavior is achieved by reinforcing the tendency to totally converge rather than trying to slow or even inhibit total convergence, but once they have converged forcing particles to be ejected with significant velocities so that they carry on searching for other optima. This behavior is achieved by adding an extra short-range force of attraction to the basic swarm equation

(as shown in Eqn. (4)) and making use of the finite time between velocity updates for particles.

$$\begin{aligned} \bar{V}_{T+t} = M \times \bar{V}_T + (1 - M) \times & \left(\left(\frac{G \times R_1 \times (\overline{Gbest} - \bar{X}_T)}{t \times |\overline{Gbest} - \bar{X}_T|} \right) \right. \\ & \left. + \left(\frac{L \times R_2 \times (\overline{Lbest} - \bar{X}_T)}{t \times |\overline{Lbest} - \bar{X}_T|} \right) \right) + \overline{SRF} \end{aligned} \quad (4)$$

where \overline{SRF} is the net short range force acting on this particle. The i th component of the short range force exerted on particle x by particle y is given by

$$SRF_{xyi} - SRF^{factor} \times \frac{V_{xyi}}{D_{xy}^{SRF_{power}}} \quad (5)$$

where V_{xyi} is the i th component of the velocity of particle x with respect to particle y , D_{xy} is the distance from particle x to particle y , SRF_{factor} is the magnitude of the short range force at unit distance, and SRF_{power} sets how fast this force decreases with distance.

As a result of the discrete way in which fitness evaluations and updates to the velocity of the particles is done, an aliasing effect causes pairs of particles to approach, pass each other and then continue at very high velocities. The closer particles approach each other the higher the probability of this happening. There is no longer a need to try to stop the particles fully converging; once converged this aliasing effect will cause particles to be ‘ejected’. As the velocity with which the particles leave the swarm is variable, exploration can continue both locally and at a distance.

The way in which this aliasing effect works is as follows. As particles approach each other the magnitude of the short-range force will increase significantly, producing a substantial increase in the velocity of the particles towards each other. For discrete evaluation, *by the time of the next evaluation*, particles may have passed each other and be at such a distance apart that the short-range attraction that might bring them back together is far too weak to do this. As a result, the particles will continue to move rapidly apart with almost undiminished velocity, exploring beyond their previous positions. This process is shown in Fig. 1. The ‘snapshots’ are taken starting at some arbitrary time T (at the top), with the lower ‘snapshots’ being taken progressively later. The time interval between ‘snapshots’ is the basic time interval for the PSO and is represented by t .

At time T , the separation between the particles is moving into the region in which the magnitude of the short-range attraction (shown by broad arrows) is becoming significant. This augments the effect of their velocities (shown by thin arrows) so that the particles move close together. By time $T + t$ the particles are close and the short-range effect is large. As a result, the velocity

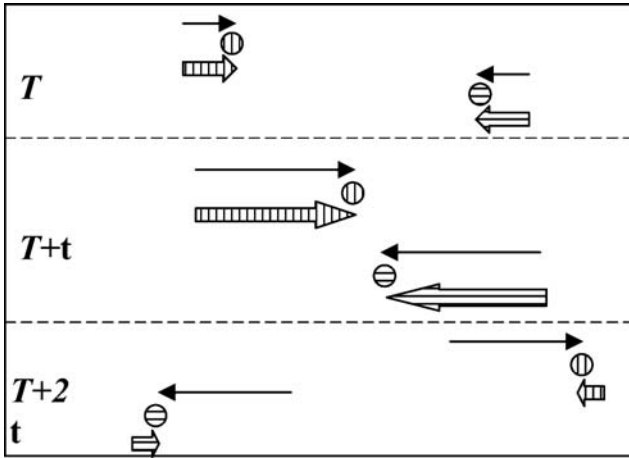


Fig. 1. A series of ‘snapshots’ showing the two particles (shown as circles), their velocities (thin arrows), and forces (thick arrows)

of the particles increases substantially, almost entirely as a consequence of the short-range attraction. By time $T + 2t$ when the next evaluation is made the particles have passed each other, and are so far apart the short-range force is weak. Consequently, the particles continue to diverge, retaining at $T + 2t$ much of the velocity obtained as a result of the short-range forces acting at time $T + t$. The short-range forces will continue to decrease as the particles move apart, leaving only the normal swarm factors to influence their future movement in the absence of other close encounters.

The total swarm automatically becomes broken into a number of sub-swarms called ‘waves’, each with its own *gbest* value. Particles that leave a converging swarm as a result of this aliasing effect leave the wave they were in and are ‘promoted’ to join the highest numbered (most recently created) wave. Should the particle already be part of the highest numbered wave, the particle becomes the founder member of a new ‘most recently created’ wave. Importantly, a form of evolution is introduced by making all particles from some lower performing wave be compulsorily recruited into a better performing higher numbered wave. Waves that run out of particles (owing to promotion or recruitment) die out. In this way there is a continual automatic updating of best position information available to the successive waves.

For static problems² each particle keeps a tabu list of places that it has already explored and is repelled from any place on its tabu list. In this way re-exploration of any point in problem space is largely (but not totally) eliminated and much pointless computation saved.

² The minor changes to suit WoSP to dynamic problems, ones that change even whilst the optimization algorithm is running, will be discussed later.

5 Controlling the Computational Expense

Whichever version of PSO is used, every time a particle moves to a new position the fitness of this particle at this position has to be calculated. For real life complex problems this fitness calculation may well dominate the computational load of the algorithm. Although the PSO lends itself to being run on a number of computers in parallel (for example each particle's current fitness being evaluated simultaneously by a different computer), for really complex problems even this approach is not adequate and it behoves us considering ways to improve the computational efficiency of the PSO algorithm. Two possible computational enhancement possibilities will be mentioned here.

5.1 Using a Dynamic Swarm Size

Since the fitness of each particle has to be evaluated for every iteration of the algorithm, one possible approach is to limit the number of particles. While swarms do not need to be large, too few swarm members will limit the search capability and thus the average quality of the results obtained. However, the number of particles does not need to be constant. When the swarm is converging and the particles get very close together it may be a waste of resources to support so many particles. Some particles could simply be 'switched off' and take no further part in the swarm thus saving their evaluations. Of course, the monitoring required and deciding when such action should be taken, must not add so much computational cost as to negate the computational advantage we seek to gain. In addition, knowing precisely when and which particles should be switched off is in itself a non-trivial matter.

5.2 Fitness Estimation

It is possible to avoid having to measure the fitness at every position if you can instead estimate it. This has been shown to work for genetic algorithms [23] and a variation has been shown to work for PSO for a range of problems [12]. The idea is to estimate the fitness of a particle at a new position using the fitness of this particle last time it was estimated or evaluated together with the fitness of the particle that last iteration was closest to this new position. These fitnesses may be estimates themselves and may have been estimated from other fitnesses that were themselves estimates. Obviously a fitness based on estimates that were based on estimates, and so forth, would not be very accurate and so a new parameter is associated with a particle's fitness – its 'reliability'. This is set to one if the fitness was found as a result of a true evaluation, and this figure is decreased every time an estimation of the fitness is made. When a fitness drops below a threshold, the estimation is discarded and a true evaluation done, returning the reliability to one.

The formulae for estimating the fitness and reliability of the particle in the new position (F_{new} and R_{new}) in terms of the fitness and reliability of the

two closest positions to its new position during the last iteration (F_1, R_1 and F_2, R_2) are:

$$F_{new} = \frac{W_1 F_1 R_1 + W_2 F_2 R_2}{W_1 R_1 + W_2 R_2} \tag{6}$$

and

$$R_{new} = \frac{(W_1 R_1)^2 + (W_2 R_2)^2}{W_1 R_1 + W_2 R_2} \tag{7}$$

where W_1 and W_2 are the relative weightings to be placed on the two closest positions. These weightings are derived from the Cartesian distances between the new position and each of the two closest positions last iteration (D_1 and D_2):

$$W_1 = 1 - \frac{D_1}{D_1 + D_2} \tag{8}$$

and

$$W_2 = 1 - \frac{D_2}{D_1 + D_2} \tag{9}$$

Figure 2 shows the fitness values (and their reliabilities) that would be calculated for a simple one-dimensional case, where the values are derived from a fitness of 1 (reliability 0.8) at position 15 and another fitness of 2 (reliability 0.6) at position 25. Notice how the fitness and reliability matches at each known point. The fitness is linearly interpolated between the two known points and moves asymptotically to the average of the two fitnesses at points far from them. The reliability however falls away the further we move

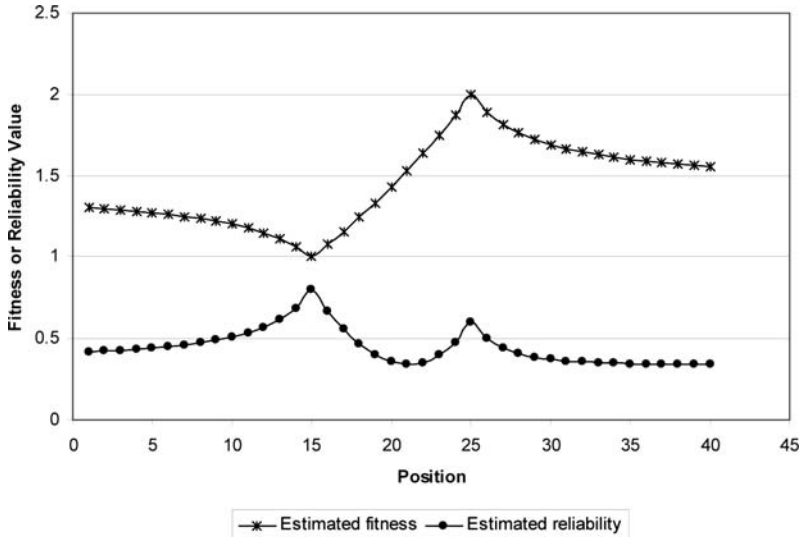


Fig. 2. An example fitness and reliability calculation in one dimension

from a known point – note how the value is influenced by the reliability of the closest known point.

The only new parameter introduced to the algorithm is the *reliability threshold* T [0..1] that is used to determine when a true fitness evaluation is required (a threshold of 1 would result in true evaluations every time as in a conventional PSO). The initial positions of all particles must be truly evaluated and their reliability set to 1. However, after this the following algorithm is used whenever a particle's fitness is required. Note that it only requires a list to be kept of the positions, fitness and reliability of all particles' last iteration. It would, of course, be possible to keep a list of positions, fitness and reliability triads derived from more than just the last iteration, but experience so far suggests that any modest increase in performance is not worth the computational cost.

Algorithm 5 PSO position, fitness and reliability algorithm

1. Record the distance this particle has moved since last iteration and this particle's fitness and reliability last iteration.
 2. Find the particle whose position last iteration was closest to this particle's current position and record its fitness and reliability, together with the distance from this particle's current position.
 3. Using Eqns. (8) and (9) calculate the relevant weighing factors for each of these two positions.
 4. Using Eqns. (6) and (7) estimate the fitness and reliability of the fitness estimate for this particle's current position.
- if** this reliability is greater than the threshold **then**
 keep these fitness and reliability estimates.
- else**
 discard the fitness estimate and perform a true fitness evaluation;
 set reliability of this fitness to 1.
-

Experiments have shown that there is no significant difference in the average performance at any number of iterations of the algorithm with or without fitness estimation. However, using fitness estimation would have required far fewer real fitness evaluations, a saving in time if the time for a true fitness evaluation is greater than the time required to find the closest particle and calculate the fitness estimate. This condition will be met for many real life practical problems, and so for these problems using fitness estimation will allow you to obtain essentially the same result in a shorter time.

6 Dynamic Optimization Problems

Dynamic optimization problems are problems in which the objective function being optimized changes in some way while the optimization is taking place. Obviously there will be some limit to the rate of change that can be tracked,

but swarms are able to adapt to changing conditions as long as they are not too rapid. The changes can be divided into three groups [22]:

1. *The actual problem being changes alters as time passes.* For example, when scheduling the delivery of goods to multiple places the original aim of minimizing fuel usage is replaced by an overwhelming need to minimize the number of late deliveries.
2. *The components available to use in building the solution change.* To continue the same example used above, some delivery trucks break down and others are returned from repair and become available.
3. *There is a change to the constraints on the problem.* Still continuing the same example, some of the roads that might be used now have altered speed restrictions.

In practice however, much of the work in the area of dynamic problems has been done on function optimization problems in which the position and/or magnitude of peaks in the function vary with time.

When attempting to find and track optima in dynamic problems, the swarm behavior must become even more un-biological. It is a more biologically plausible scenario to have to track an optimum that slowly changes position (while remaining a good optimum) than to have to find the global optimum from a number of local optima that change their relative quality ranking (and possibly position). It is not surprising therefore that, provided the swarm is not allowed to fully converge so that the velocity of each particle is reduced to zero, a slowly moving peak can expect to be followed in the sense of there being a high probability that one or more particles will traverse sufficiently close to the moving peak at the time of a fitness evaluation that the new position of the optimum is discovered. A number of methods to ensure incomplete convergence will be described below but on their own none of these is a complete solution as some mechanism also has to be introduced to update the best fitness known based on a combination of the value found and the time at which it was found.

To observe the emerging eminence of a local optimum far from the position towards which the swarm was just converging requires some swarm particles to be exploring in the vicinity of this distant position. Various methods to achieve this will also be described below. Again the maintenance of a number of explorer particles is not a complete solution. The social factors (the best position found by each particle and by the swarm) must also be updated as the old information goes out of date.

6.1 Ways to Achieve these Adaptations

As suggested above there are a number of non-biological adaptations that need to be made to the classical swarm algorithm to suit it for dynamic problems. These can be summarized as:

- preventing the complete convergence of the swarm,
- keeping personal and social reference points up-to-date, and
- maintaining or generating explorer particles far from any current point of convergence.

Approaches that achieve at least one of these aims will be considered.

6.2 Preventing Total Convergence

Social influences between particles – attractions to *gbest* and *lbest* – will clearly tend to result in total convergence. In order to change this it is necessary to introduce some counter influence.

One method, introduced by [1] is to give at least some particles a charge so that, by analogy with electrostatics, two particles would experience a repulsion force as they approached and the swarm would then not be able to fully converge. The particles would in time reach some (possibly dynamic) equilibrium between the convergence and divergence effects, but this does not mean that they are actively exploring.

A second method, introduced by [5], is to divide the swarm into sub-swarms so that not all particles are converging on the same point. As well as the main swarm, a particle and its closest neighbours may form a sub-swarm if the variance in the fitness of the particles is less than some threshold. Any particle that is not a member of a sub-swarm belongs to the main swarm. These sub-swarms may merge or acquire extra particles from the main swarm or collapse back into the main swarm. While developed for multi-modal functions this niching behavior could also be used, in principle, to limit total swarm convergence. However the algorithm depends on a uniform distribution of particles in the search space, a condition that may be able to be met after initialization but which is not met after convergence into the sub-swarms has taken place.

6.3 Refreshing the Best Positions

If an attraction to *pbest* is being used these best positions may be updated by allowing particles to replace their previous best position with the current position periodically [6]. Choosing a suitable period without knowledge of the problem being optimized can be problematic. If an attraction to *gbest* is being used then the fitness at this position may be periodically re-evaluated [2]. As the fitness at that point deteriorates, the probability that it will be replaced by another position as a result of the current fitness at that position increases. Again a suitable re-calculation frequency has to be chosen.

6.4 Forcing Explorer Particles

The simplest approach just requires that a number of particles be periodically moved to randomly chosen points and have their fitness re-evaluated [16]. Another approach organizes particles in a tree with each particle being influenced by the particle above it (social) and itself (best position and momentum). A particle swaps with the one above it if it out performs it. This gives a dynamic neighbourhood that does require extensive calculation. This has been adapted to dynamic problems by [17, 18]. After the value of the best-known position (*gbest*) changes (it is re-evaluated every cycle) a few sub-swarms are re-initialized while the rest are reset (have their old personal best information erased and replaced with the current position). The sub-swarms then search for the new optimum.

[1] introduce a more elaborate approach using quantum particles. Using an analogy to quantum mechanics, a particle on measurement is placed randomly within a given radius of its net current point of attraction. A uniform (and very un-physical) distribution is used, but this could be changed so that there was not a uniform probability of the particle being at every distance, and a function chosen so that a finite probability exists of a movement to a distance far from the point of attraction.

6.5 Adapting WoSP to Dynamic Problems

WoSP, because of the sequential way it explores optima, is inherent suited to dynamic problems. All that needs to be changed is the removal of the tabu list that each particle keeps, recording the positions from which it has been promoted. While for static problems repulsion for these points makes sense, for dynamic problems a particular point in problem space may be a good optimum at several disjointed times and a poor optimum in between these times. Completely removing the list may be inefficient, it may be better to periodically review the entries on each particle's list. Extending the idea from [18], each of the previously explored optima on all the lists could be periodically re-examined and all points for which significant changes were found to have occurred in the fitness would be removed from the tabu lists of the particles. This would need to be done frequently and whether the reduced re-exploration would be worth the computational expense of this housekeeping is not clear.

7 Particle Swarm and Quantized Problem Spaces

So far all the descriptions of the PSO have been in terms of continuous problem spaces, indeed all the swarm update equations presented in this Chapter explicitly require a continuous problem space. However, this does not mean

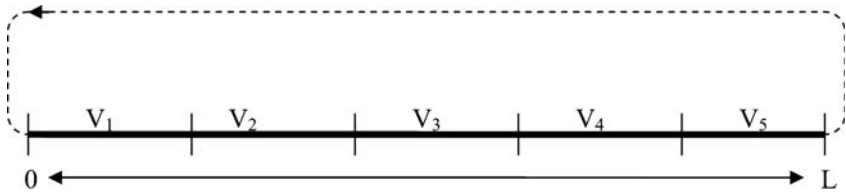


Fig. 3. An axis in PSO space suitable for a five value quantized parameter in problem space

that the PSO cannot handle other types of problems with other types of problem spaces. All that is required is the ability to map the continuously variable positions in the PSO particle space to the problem space. There are many problem spaces that are not continuous problem spaces and there is not room to describe possible mappings for more than one. The one problem space chosen is the quantized problem space, a space in which the values associated with some parameter are constant for a while and then change instantaneously to a new value. The values of this parameter are discrete (quantized) values. As an example of the mapping required, consider the case where a certain parameter can only have one of five values, it does not matter what the actual values are, we will refer to them as V_1, V_2, V_3, V_4 and V_5 .³ Let the length of the axis in PSO space be L . Figure 3 shows this axis.

Note that the axis wraps around, a particle reaching the end of the axis at L immediately reappears at position zero. Effectively, Eqn. (3) becomes:

$$\bar{X}_{T+t} = (\bar{X}_{T+t} \times \bar{V}_{T+t}) \bmod(L) \tag{10}$$

If L is made equal to the number of categories, the value that a particle’s position corresponds to can be found by applying Eqn. (10) and then taking the integer part of the answer. However, there is obviously a length of L/N on an axis that will correspond to the same quantized value (where N is the number of values – five in this example). The fitness function is required to be continuous in the sense that adjacent positions should in general return different fitness values so as to provide guidance to the swarm as it converges. For this reason the distance of a particle from the closest cell centre may also be recorded (expressed as a fraction of L/N). The fitness function now becomes comprised of two parts, the ‘normal’ fitness function (F) and the average of these fractional distances across all axes in the problem (f). When deciding if one position is fitter than another the ‘normal’ fitness function parts of the total fitness (the two F values) are consulted first. If these differ,

³ Each quantum value has been allocated an equal length on the axis in this example. This is not essential and different lengths could be allocated, each length being set proportional to the relative probability of this quantized value occurring, perhaps.

the answer is clear. However, should they be the same then the position with the better value of f is chosen.

8 Some Sample Results

8.1 Problems used as Examples in this Chapter

This Chapter has described a number of variants of the PSO algorithm and a number of problem domains to which they may be applied. Space will not allow results to be presented for every variant and every domain; indeed the limitations of fixed type do not readily allow the presentation of dynamic problem results in a simple and clear way. The results from four problems have been chosen in order to show the behavior of the basic and WoSP variants of PSO both with and without fitness estimation and on continuous and quantized problem spaces. These problems are described below.

Finding the Origin

The first of these is the apparently trivial problem of finding the origin, the fitness of each particle being its distance from the origin, as shown in Eqn. (11).

$$f = \sum_{i=1}^{100} \sqrt{(x_i)^2} \quad (11)$$

where f is the fitness and x_i is the i th component of the position of the particle.

This becomes quite hard as the number of dimensions increases for any algorithm that makes simultaneous updates to all dimensions (as the PSO does). For a new position to be more successful than the old position the net effect of all the changes in all the dimensions must be a decrease. As the number of dimensions increases this becomes harder, especially approaching the origin. Results will be presented for PSO seeking the origin in 100 dimensions.

Rastrigin's Function

The second problem is Rastrigin's Function Eqn. (12).

$$f = ((x_i)^2 - 10\cos(2\pi x_i) - 10) \quad x_i \in [-5.12 \dots 5.12] \quad (12)$$

where f is the fitness and x_i is the i th component of the position of the particle.

This is a well known function commonly used as a test problem for optimization algorithms. This can be readily solved by a traditional PSO algorithm.

Schwefel's Function

The third problem is Schwefel's function [24] in 30 dimensions. This is another well known function commonly used as a test problem for optimization algorithms. It has a large number of local optima but one unique global optimum. No matter the number of dimensions, the position and size of this global optimum can be readily calculated, as can the positions and sizes of any local optimum.⁴

$$f = \sum_{i=1}^{30} x_i \sin(\sqrt{|x_i|}) \quad x_i \in [-512.03 \cdots 511.97] \quad (13)$$

where f is the fitness and x_i is the i th component of the position of the particle.

This problem is included as the chance of the conventional PSO algorithm finding the global optimum position is very low. This is because in 30 dimensions there are 1.2×10^{27} local optima that need to be explored. Sequentially exploring optima using the WoSP PSO variant gives an approximately 40% chance of finding the one global optimum [4]. Like many real life problems that also have many local optima, fitness evaluation now constitutes a significant fraction of the total computational load.

A Timetabling Problem

Finally a simple quantized problem space is used to illustrate how the PSO may solve this type of problem. The problem used is a simple timetabling problem involving scheduling nineteen classes for four groups of students in three rooms for one day of six time periods. While all classes can occur in any of the six available time periods there are various constraints as to the rooms each class may occur in and the group(s) of students that will be involved. The aim is to timetable the classes so that these constraints are met and no student is required to undertake two classes at once and no room is required to contain more than one class at a time. The constraint details are shown in Table 1, with the classes, room and groups identified by numbers.

There are approximately 1.7×10^{26} ways in which these classes can be arranged but only slightly more than 2,500 that meet all constraints. While this is a trivial problem as far as timetabling is concerned, it is more than adequate to explore the behavior of PSO particles in quantized problem spaces as it is easy to comprehend and has the advantage of fast fitness evaluation.

⁴ The version of Schwefel's function given here is the form in which the value of the fitness function can be negative at some places in problem space. If it is more convenient for the fitness to always be positive (for example if in a GA with the breeding probability directly proportional to the fitness) this can be achieved by adding 418.9829 times the number of dimensions to the result given by equation 13.

Table 1. Timetable problem constraints

Class	Possible rooms	Groups involved
1, 2, 3	1, 2, 3	1
4	4	1
5, 6, 7	1, 2, 3, 2	
8	4	2
9, 10, 11	1, 2, 3	3
12	4	3
13, 14, 15	1, 2, 3	4
16	4	4
17	1, 2, 3	1, 2
18	1, 2, 3	3, 4
19	1	1, 2, 3, 4

Table 2. Parameter values used for each of the four problems

Parameter	Origin problem	Rastringin’s function	Schewefel’s function	Timetable problem
Particle count	30	30	30	30
M	0.5	0.9	0.9	0.9
G	0.5	0.9	0.9	0.3
L	0.5	0.5	0.5	0.7
Neighbourhood	Particle & 3 closest	Particle & 3 closest	Particle & 3 closest	Particle & 3 closest
Search scale	–	–	500	2
SRF^{factor}	–	–	5000	500
SRF^{power}	–	–	3.5	3.5

Two quantized variables were associated with each class, the time it is to be scheduled and the room it is to occur in. Each of these is mapped to a different axis in problem space. A total of 38 axes were therefore required to schedule these 19 classes. The number of possible quantized values these axes contain varies from 1 to 6.

8.2 Experimental Details

The values used for the parameters for each of these four problems are shown in Table 2.

9 Sample Results

All the figures below contain multiple plots, each corresponding to a different threshold. The concept of a threshold is really only meaningful when using some fitness estimation, but setting the threshold to one has the effect of not

allowing any fitness estimation – the ‘traditional’ PSO. For thresholds below one, the lower the threshold value the higher the ratio of fitness estimations to true fitness evaluations.

9.1 Minimizing the Distance to the Origin in 100 Dimensions

The fitness values reported at a particular iteration are the average distance from the origin of all 30 particles in 100 independent repeats of the experiment.

There is some evidence on all the plots in Fig. 4 of two phases of activity, in the first of which fast progress is made. In the second phase (from about 1000 iterations onwards) progress is slower as the algorithm finds it harder to make a move that has a net beneficial effect on the fitness over all 100 dimensions. It could be argued that PSO (like other algorithms that simultaneously update all dimensions) is not a very suitable algorithm for this second phase.

Having a threshold of either 0.75 or 0.5 has little effect on the average best fitness per iteration compared to a threshold of one, despite the fact that the first two thresholds correspond to a mixture of fitness estimation and true fitness evaluation, and the last to only using true fitness evaluation. When the threshold is as low as 0.25, the average fitness falls more slowly. The fact that the estimated fitness can never be lower (or higher) than the lowest (highest) fitness of the two reference points from which it is derived makes it even harder for the algorithm to find points whose estimated fitness is better than the current \bar{G}_{best} and \bar{L}_{best} . This may, at first sight, suggest that the fitness

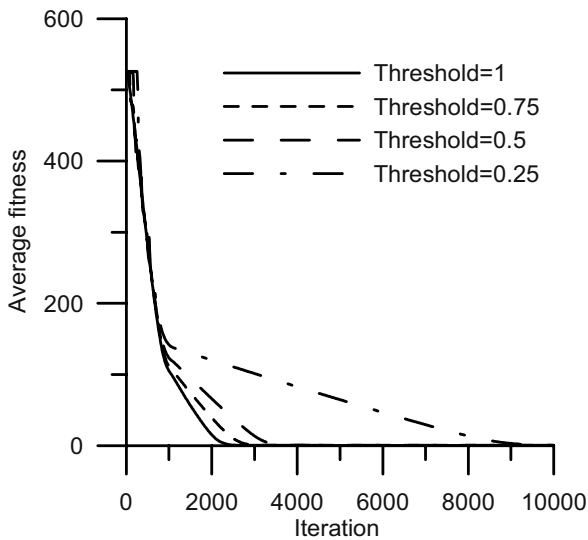


Fig. 4. Finding the distance to the origin in 100 dimensions as a function of the iteration

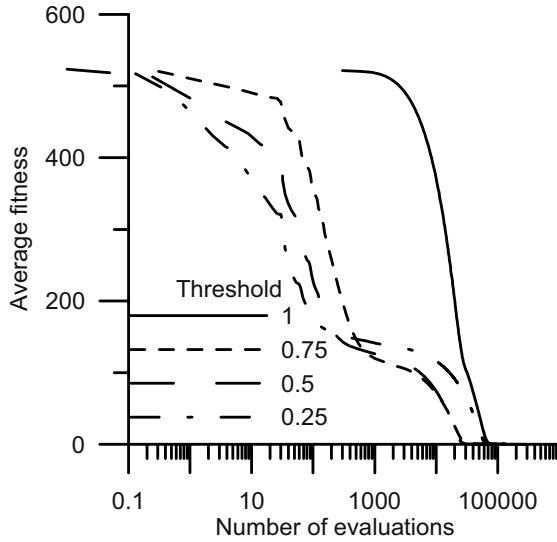


Fig. 5. Finding the distance to the origin in 100 dimensions as a function of the number of true evaluations as opposed to fitness estimates made (the fitness values reported at a particular iteration are the average)

estimation PSO algorithm is not particularly suited to problems containing regions of problem space that are smooth changes.

However, when the average fitness is plotted against the number of true evaluations as in Fig. 5, it becomes clear that for this problem it will take less true evaluations to achieve a given performance using fitness estimation than when not using it (if only marginally for a threshold of 0.25). The quality of the final solution is comparable in all cases. Had the time taken to do a true evaluation been significantly greater than the time taken to estimate the fitness (which is not the case for this particular simple demonstration problem), the overall result would have been less computing for results of comparable quality.

9.2 Rastrigin’s Function in 100 Dimensions

Consider the bold line in Fig. 6 which shows the average best known fitness (averaged over 100 independent repeats) for a PSO solving Rastrigin’s Function in 100 dimensions using only true fitness evaluation. While the gradient of the graph varies, progress is almost continuous, with only short periods of apparent stagnation. A genetic algorithm would be likely to show longer periods of apparent stagnation. Like a GA, the solid curve tends to plateau out in the vicinity of, but not actually at, the global optimum of zero. Once

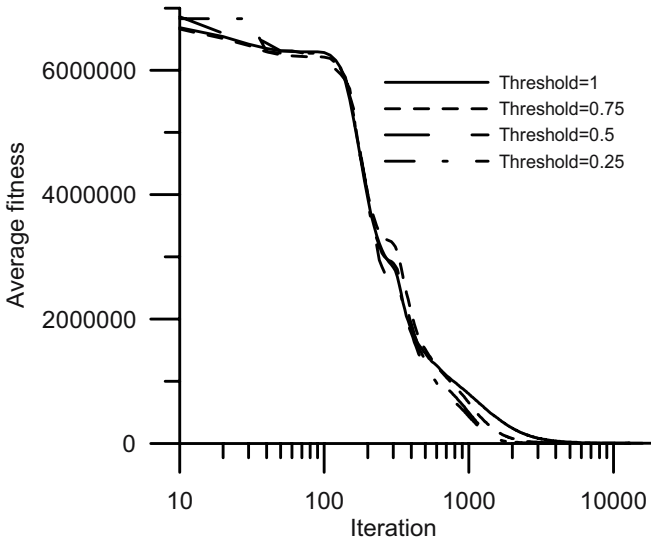


Fig. 6. Best known fitness per iteration (averaged over 100 independent repeats) for Rastrigin's function in 100 dimensions

Table 3. Final statistics for Rastrigin's function (averaged over 100 independent repeats)

Threshold	1	0.75	0.5	0.25
Average fitness	1553	1511	1525	1619
Standard deviation	221	180	121	153
Maximum	2544	2510	1899	2074
Minimum	1241	1166	1262	1314

in the vicinity of an optimum it is often better to switch to using a simple gradient descent algorithm for the local search.

Figure 6 shows that it would be hard to pick whether fitness estimation was being used (and, if it were, what value was being used for the threshold) if one just has the average best known fitness at each iteration (an iteration is all particles making one position update). The values presented here are for 30 particles and the average is over 100 independent repeats.

However, Fig. 7 once again shows that the performance as a function of the number of true evaluations differs substantially with the four threshold values. The performance with a threshold of one (no fitness estimation) is poorer than any of those that do allow fitness estimation.

Table 3 shows the average final fitness and the standard deviation for the four tested threshold values, along with the maximum and minimum final values found (a total of 100 independent repeats were done for each threshold

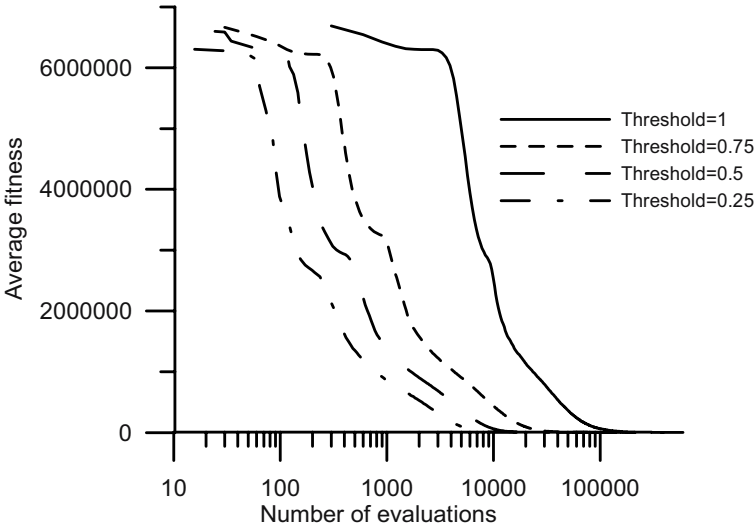


Fig. 7. The best known fitness per true evaluation (averaged over 100 independent repeats) for Rastrigin’s function in 100 dimensions

value). As Rastrigin’s Function is a minimization function (with a global best value of zero) the final values show that the exact optimum had not been located, although compared to the initial values of over 6,000,000 the particles had made significant progress in the number of iterations that they had been allowed (20,000). Again PSO, with or without fitness estimation, is not efficient in the final stages of converging to an optimum. Table 3 also shows that the threshold in use could not be identified from these end results in a blind test.

Figure 8 shows the ratio of the cumulative totals of the number of fitness evaluations to the number of fitness estimations for Rastrigin’s function as a function of the iteration number. Note that whenever fitness estimation is being used this ratio is asymptotic to a number less than one. This means that the number of true fitness evaluations is always less, often significantly less, than one half of the number of true fitness evaluations that would be required by a PSO algorithm that does not use fitness estimation.

9.3 Schwefel’s Function in 30 Dimensions

This function, with its many local optima, is highly problematic for a traditional PSO algorithm. However, it can be solved by successive exploration of optima using the WoSP variant of the PSO algorithm [14] The results presented here only show relatively early stages of this exploration (only the first 10,000 iterations, by which stage the best known result is well within the top thousandth of one percent of all results). However, given 200,000 iterations, WoSP has a 41% chance of finding the global optimum [14].

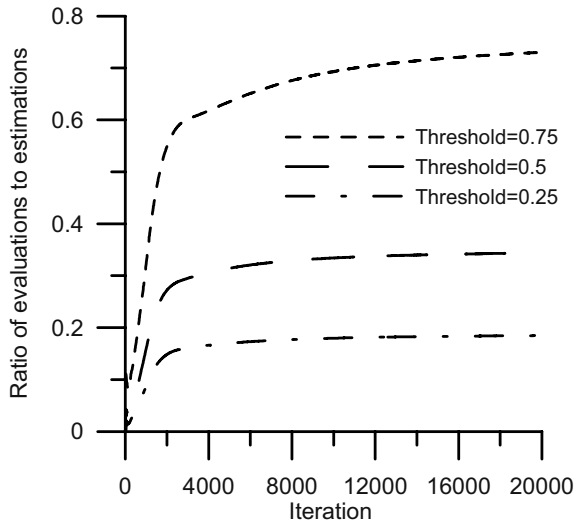


Fig. 8. The ratio of the total number of true fitness evaluations to total number of fitness estimations for Ratrigin’s function plotted per iteration

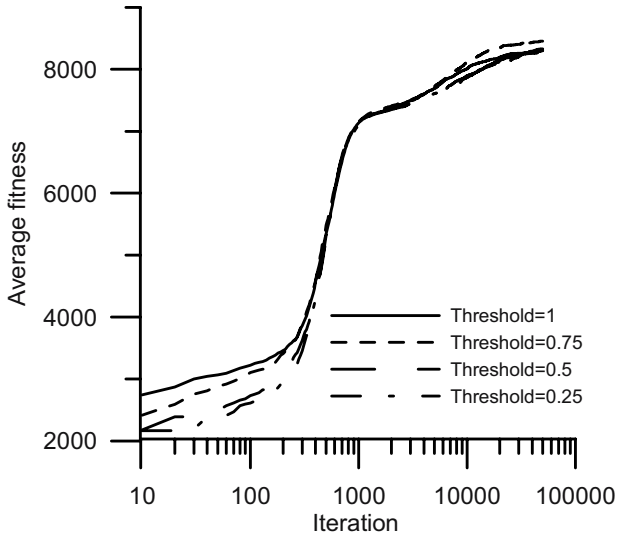


Fig. 9. The best known fitness per iteration (averaged over 100 independent repeats) for Schwefel’s function in 30 dimensions

Although there is some difference in the early stages and a slight difference in the late stages, the plots of the average best known fitness versus iteration shown in Fig. 9 are very similar, especially during the time that the swarm is making good progress. Since this is a plot of the best optimum known, many

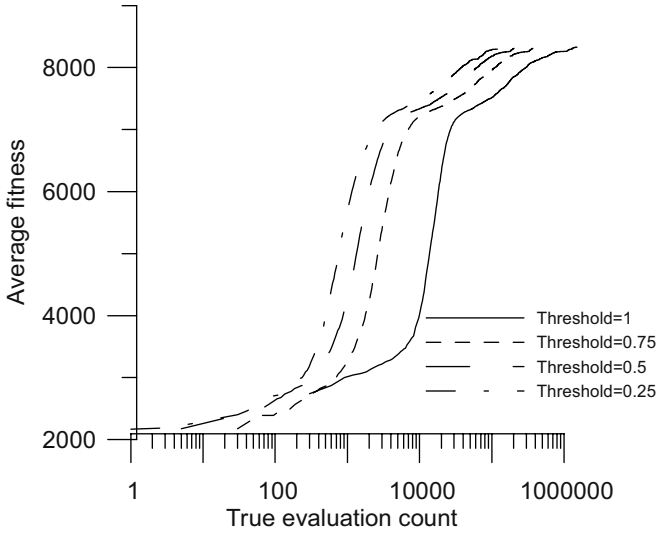


Fig. 10. Best known fitness per true evaluation (averaged over 100 independent repeats) for Schwefel’s function in 30 dimensions

optima are explored without affecting the plot. Nevertheless progress is fairly continuous with few regions of apparent stagnation.

When plotted as the average best known fitness versus the number of true evaluations (Fig. 10), the advantage of using fitness estimation becomes clear. If there was only enough time to perform 10,000 true fitness evaluations, the best location found by the conventional WoSP PSO (threshold=1) would have, on average, a fitness of some 3500. Using a threshold of 0.25, the best location the WoSP PSO with fitness estimation would have found, on average, would have a fitness of about twice this.

The particular significance of this result is that it is achieved in a system that frequently moves particles far from the region(s) of problem space that have been explored. This underlies the importance of the reliability value associated with the fitness, and the way that this decreases, not only with the reliability associated with the two reference positions in use, but also with the distance of the new position from these two reference positions. This clearly demonstrates that the fitness estimation algorithm not only decreases the number of fitness evaluations required but also is reasonably efficient at deciding when true evaluation is really required.

Results from a Simple Quantized Problem Space

Because of the large number of ways that the classes can be arranged, attempts to solve it with the traditional PSO algorithm were, as expected, highly ineffectual (as will be seen from Table 5). However, the WoSP variant of the PSO

was able to solve the problem. Details of the basic parameter values used are shown in Table 2. It is not claimed that the values used were optimal but they do follow the general guidelines given in [15]. Each swarm was randomly initialized and then allowed 1,500,000 evaluations after which the best solutions found were recorded.

A simple problem specific local heuristic was used that took a solution with one or more constraint violations and repeated the following set of steps until either the number of constraint violations was reduced to zero or a user specified number of attempts had been made. A list of all classes that were involved in these violations was made. One class was chosen at random from this list and a second list made of all the other room time combinations to which it could be moved. One possible move was chosen from this second list and the change this possible move would make in the constraint violation count was calculated. Only if this was positive was the move actually made. A new list of clashing classes was then made and the process repeated until either no clash occurred or a total of fifty tries had been made. The algorithm described in the above paragraph is very greedy and can easily result in a local optimum being reached that does not meet all constraints. As a result the original quantized position passed to the local heuristic was saved and the algorithm described above was tried twenty times, with the original quantized position being restored at the start of each time. The best result found in any of these twenty tries was the result actually used. The extra computational load was insignificant as the local heuristic was only run when a wave died – typically a few hundred times per run. The results of running the conventional and WoSP variant of the PSO on this problem are shown in Table 4.

Table 4 shows that in terms of the best performing repeat out of each group of 100 there is a steady improvement as progressive enhancements are made to the classical particle swarm algorithm (from left to right in the table). Interestingly, adding waves alone produces a greater positive effect than adding

Table 4. An overview of the performance of 100 independent repeat runs for each of the four possible combinations of waves and local heuristic

	No	No	Yes	Yes
Using waves	No	Yes	No	Yes
Local heuristic	No	Yes	No	Yes
Number of times one or more solutions found that satisfied all constraints	0	0	8	80
Average number of constraint violations per run	5.72	3.81	2.25	0.2
Number of constraint violations in best solution found	1	1	0	0
Number of constraint violations in worst solution found	8	8	3	1

Table 5. The number of constraint violations for 100 repeats of each of all combinations of waves and local heuristic

Number of constraint violations in best solutions found	Waves		Local heuristic		Yes		Yes	
	No	No	No	Yes	No	Yes	No	Yes
0					8		80	
1				6	26		20	
2			1	10	32			
3			1	32	17			
4			15	22	10			
5			24	13	5			
6			29	14				
7			27	2	2			
8			3	1				

Table 6. The average reduction in constraint violations obtained using the local heuristic for 100 repeats with and without waves

	Average	Minimum	Maximum
Without waves	1.9	0	5
With waves	2.1	0	7

the local heuristic alone but the best results are obtained when both of these are used. While Table 4 shows only the best result for each repeat, Tables 5 and 6 show statistics derived from all the repeats for all combinations.

Comparing the columns in Table 5 that do not involve the local heuristic, it is clear that the addition of waves consistently and substantially improves the performance. Again it can be observed that the performance with waves alone is better than the performance with the local heuristic alone.

Table 6 shows that the improvement made by the use of the local heuristic was essentially independent of the use of waves.

The best performing combination by far is when both waves and the local heuristic are used and these results have been examined in more detail.

During the 100 independent repeats, each run reported on average 653 (max 671, min 634) candidate solutions that were passed to the local heuristic. A candidate solution corresponds to the best position found by a wave during its existence and subsequently refined using the local heuristic. The range of the number of constraint violations in all these candidate solutions is from 0 to 12. During the runs a grand total of 1242 solutions that satisfied all constraints (absolute solutions) were found. Twenty runs produced no absolute solutions; the other 80 runs produced between 1 and 51 solutions each, with an average of 15.7 absolute solutions per run. However, even though each particle maintained an individual list of promotion points, some optima were

explored by more than one wave during a run. On average, the runs that found absolute solutions found 5.8 different absolute solutions each (the maximum for any wave being 21, the minimum 1). Overall, the 100 repeats found a grand total of 446 different absolute solutions out of the approximately 2500 that exist.

10 Concluding Remarks

The particle swarm optimization algorithm has proved to be efficient, fast and flexible. It shows promise as an effective algorithm for a wide range of optimization problems. A number of variations have been suggested to better suit it to particular classes of problems and some of these, together with the basic algorithm, have been discussed in this Chapter. No algorithm is ideal for all situations and it has been noted the PSO, like other algorithms such as the genetic algorithm, is really a coarse search algorithm that becomes inefficient in the final stages of homing in on an optimum. But, when coupled with an appropriate local search technique, PSO and its many variants deserve a prominent place in the armory of everyone seriously involved with optimization in the real world.

References

1. Blackwell T, Branke J (2004) Multi-swarms optimization in dynamic environments. In: *Lecture Notes in Computer Science*, 3005. Springer-Verlag, Berlin: 489–500.
2. Blackwell TM, Bentley PJ (2002) Dynamic search with charged swarms. In: Langdon WB et al. (eds.) *Proc. Genetic and Evolutionary Computation Conf. – GECCO-2002*, 9–13 July, New York, NY. Morgan Kaufmann, San Francisco, CA: 19–26.
3. Braendler D, Hendtlass T (2002) The suitability of particle swarm optimisation for training neural hardware. In: Hendtlass T, Ali M (eds.) *Lecture Notes in Artificial Intelligence*, 2358, Springer-Verlag, Berlin: 190–199.
4. Brits R (2002) Niching strategies for particle swarm optimization. *Master's Thesis*. Department of Computer Science, University of Pretoria, South Africa.
5. Brits R, Englebrecht A, van der Bergh F (2002) A niching particle swarm optimiser. In: *Proc. 4th Asia-Pacific Conf. Simulated Evolution and Learning (SEAL'2002)*, 18–22 November, Singapore: 692–696.
6. Carlisle A, Dozier G (2000) Adapting particle swarm optimization to dynamic environments. In: Arabnia HR (ed.) *Proc. Intl. Conf. Artificial Intelligence*, 26–29 June, Las Vegas, NV: 429–433.
7. Carlisle A, Dozier G (2002) Tracking changing extrema with adaptive particle swarm optimizer. In: Jamshidi M, Hata Y, Fathi M, Homalfar A, Jamshidi JS (eds.) *Proc. World Automation Congress (Intl. Symp. Soft Computing in Industry – ISSCI'2002)* 9–13 June, Orlando FL, TSI Press, Albuquerque, NM: 265–270.
8. Clerc M (1998) Some math about particle swarm optimization. (available online at <http://clerc.maurice.free.fr/PSO/PSOmathstuff/PSOmathstuff.htm> – last accessed January 2007).

9. Clerc M (1999) The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *Proc. Congress Evolutionary Computation (CEC1999)*, 6–9 July, Washington, DC. IEEE Press, Piscataway, NJ, 3: 1957.
10. Digalakis J, Margaritis K (2000) An experimental study of benchmarking functions for genetic algorithms. *Intl. J. Computer Mathematics*, 79: 403–416.
11. Eberhart R, Shi Y (2000) Comparing inertia weights and constriction factors in particle swarm optimisation. *Proc. 2000 Congress Evolutionary Computation*, 16–19 July, La Jolla, CA. IEEE Press, Piscataway, NJ: 84–88.
12. Hendtlass T, (2007) Fitness Estimation and the Particle Swarm Optimisation Algorithm, *Proc Congress on Evolutionary Computing (CEC2007)*. IEEE Computer Society Press, Piscataway, NJ: 4266–4272.
13. Hendtlass T (2006) A particle swarm algorithm for complex quantised problem spaces. *Proc. Congress Evolutionary Computation (CEC2006)*, 16–21 July, Vancouver, Canada. IEEE Computer Society Press, New York, NY: 3760–3765.
14. Hendtlass T (2005) WoSP: a multi-optima particle swarm algorithm. *Proc. Congress Evolutionary Computing (CEC2005)* 2–5 September, Edinburgh, UK. IEEE Press, Piscataway, NJ: 727–734.
15. Hendtlass T (2004) A particle swarm algorithm for high dimensional problem spaces. *Proc. IEEE Swarm Workshop*, 9–11 May, Ann Arbor, MI (available online at <http://www.cscs.umich.edu/swarmfest04/Program/Abstracts/abstracts.html#HendtlassT> – last accessed 22 May 2007).
16. Hu X, Eberhart R (2002) Adaptive particle swarm optimisation: detection and response to dynamic systems. In: *Proc. Congress Evolutionary Computation (CEC2002)*, 12–17 May, Honolulu, Hawaii. IEEE Press, Piscataway, NJ: 1666–1670.
17. Janson S, Middendorf M (2003) A hierarchical particle swarm optimizer. *Proc. Congress Evolutionary Computing (CEC2003)*, 9–12 December, Canberra, Australia. IEEE Press, Piscataway, NJ: 1666–1670.
18. Janson S, Middendorf M (2004) A hierarchical particle swarm optimizer for dynamic optimization problems. In: Raidl GR, Cagnoni S, Branke J, Corne DW, Drechsler R, Jin Y, Johnson CG, Machado P, Marchiori E, Rothlauf F, Smith GD, Squillero G (eds.) *Proc. EvoWorkshop 2004*, 20–24 June, Toulouse, France. Lecture Notes in Computer Science 3005, Springer-Verlag, Berlin: 513–524.
19. Kennedy J, Eberhart RC (1995) Particle swarm optimization. *Proc. IEEE Conf. Neural Networks (ICNN95)*, November, Perth, West Australia. IEEE Press, Piscataway, NJ: 1942–1947.
20. Paquet U, Engelbrecht AP (2006) Particle swarms for equality-constrained optimization. *Fundamenta Informaticae*, 76: 1–24.
21. Parrot D, Li X (2004) A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In: *Proc. 2004 Congress Evolutionary Computation (CEC2004)*, 20–23 June, Portland, OR. IEEE Press, Piscataway, NJ: 98–103.
22. Randall M (2005) A dynamic optimisation approach for ant colony optimisation using the multidimensional knapsack problem: recent advances in artificial life. *Advances in Natural Computation*, 3: 215–226.
23. Salami M, Hendtlass T (2002) A fast evaluation strategy for evolutionary algorithms. *J. Soft Computing*, 2(3): 156–173.
24. Schwefel HP (1981) *Numerical Optimization of Computer Models*. Wiley, Chichester, UK.

Resources

1 Key Books

Engelbrecht AP (2005) *Fundamentals of Computational Swarm Intelligence*. Wiley, London, UK.

Kennedy J, Eberhart RC, Shi Y (2001) *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA.

Bonabeau M, Dorigo M, Theraulaz G (1999) *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, UK.

2 Organisations, Societies, Special Interest Groups, Journals

IEEE Computational Intelligence Society
<http://www.ieee-cis.org>

3 Key International Conferences/Workshops

IEEE Congress on Evolutionary Computing – CEC (IEEE)

Genetic and Evolutionary Computation Conference – GECCO (ACM SIGEVO)

Swarmfest
<http://www.swarm.org>

4 (Open Source) Software

Cilib – a public domain framework and library for CI algorithms

<http://cilib.sourceforge.net>

Optimization Algorithm Toolkit (OAT) ‘A workbench and toolkit for developing, evaluating, and playing with classical and state-of-the-art optimization algorithms on standard benchmark problem domains; including reference algorithm implementations, graphing, visualizations and much more.’

<http://optalgtoolkit.sourceforge.net/>

DNA and Immunity-Based Computing

DNA Computing and its Application

Junzo Watada

Graduate School of Information, Production and Systems, Waseda University,
Kitakyushu, Japan 808-0135, junzow@osb.att.ne.jp

1 Introduction

The objectives of this Chapter are twofold: firstly to introduce DNA computation, and secondly to demonstrate how DNA computing can be applied to solve large, complex combinatorial problems, such as the optimal scheduling of a group of elevators servicing a number of floors in a multi-storey building.

Recently, molecular (or wet) computing has been widely researched not only within the context of solving NP-complete/NP-hard problems – which are the most difficult problems in NP – but also implementation by way of digital (silicon-based) computers [23]. We commence with a description of the basic concepts of ‘wet computation’, then present recent results for the efficient management of a group of elevators.

2 DNA Computing

The main idea behind DNA computing is to adopt a biological (wet) technique as an efficient computing vehicle, where data are represented using strands of DNA. Even though a DNA reaction is much slower than the cycle time of a silicon-based computer, the inherently parallel processing offered by the DNA process plays an important role. This massive parallelism of DNA processing is of particular interest in solving NP-complete or NP-hard problems.

It is not uncommon to encounter molecular biological experiments which involve $6 \times 10^{16}/ml$ of DNA molecules. This means that we can effectively realize 60,000 TeraBytes of memory, assuming that each string of a DNA molecule expresses one character. The total execution speed of a DNA computer can outshine that of a conventional electronic computer, even though the execution time of a single DNA molecule reaction is relatively slow. A DNA computer is thus suited to problems such as the analysis of genome information, and the functional design of molecules (where molecules constitute the input data).

DNA consists of four bases of molecular structure, named adenine (*A*), guanine (*G*), cytosine (*C*) and thymine (*T*). Furthermore, constraints apply to connections between these bases: more specifically, *A* can connect *only* with *T*, and *G* *only* with *C* – this connecting rule is referred to as ‘Watson-Crick complementarity’. This property is essential to realize the *separate* operation (discussed later). In other words, it is possible to separate a partial string of characters ‘ad’ so that a DNA sequence complementary to the DNA denoting ‘ad’ is marked, input into a test tube, hybridized to form a double-strand helix of DNA, then abstracted. Further, this property enables us to randomly create a set of character strings according to some rule.

Since Adleman described a method for solving a directed Hamiltonian path problem with seven cities using DNA molecules [1], researchers have pursued theoretical studies to realize *general* computation using DNA molecules – for example, [31]. Adleman has developed a computational model to realize, via experimental treatment of DNA molecules, operations on multiple sets of character strings, following the encoding of finite alphabet characters onto DNA molecules [2].

As previously mentioned, DNA molecules can be used as information storage media. Usually, DNA sequences of around 8–20 base-pairs are used to represent bits, and numerous methods have been developed to manipulate and evaluate these. In order to manipulate a wet technology to perform computations, one or more of the following techniques are used as computational operators for copying, sorting, splitting or concatenating the information contained within DNA molecules:

- ligation,
- hybridization,
- polymerase chain reaction (PCR),
- gel electrophoresis, and
- enzyme reaction.

In the following Section we briefly describe the specific bio-chemical process which serves as the basis of *our* DNA computing approach.

A DNA computer performs wet computation based on the high ability of special molecule recognition executed in reactions among DNA molecules. Molecular computation was first reported in [1], where it was found that a DNA polymerase – which incorporates an enzyme function for copying DNA – is very similar in function to that of a Turing Machine. DNA polymerase composes its complementary DNA molecule using a single-strand helix of a DNA molecule as a mold. On the basis of this characteristic, if a large amount of DNA molecules is mixed in a test tube, then reactions among them occur simultaneously. Therefore, when a DNA molecule representing data or code reacts with other DNA molecules, this corresponds to super-parallel processing and/or a huge amount of memory in comparison with a conventional (electronic) computer.

2.1 Encoding Scheme

In any DNA computational procedure, the main challenge is to encode each object of interest into a DNA sequence. A correct design is essential in order to ensure optimal results; an incorrect design could result in wrong sequences following the ligation process.

Ligation and Hybridization

When DNA sequences are dropped in a test tube using a micro pipettor (Fig.1), the DNA sequences recombine with each other by means of some enzyme reaction, this process being referred to as 'ligation'. All DNA sequences to be used in the experiment – along with their complements – are mixed together in a single test tube. Normally the oligonucleotide or DNA mixture is heated to 95° centigrade (celsius) and cooled to 20°C at 1°C per minute for hybridization, as indicated in Fig. 1. The reaction is then subjected to a ligation. At the end of this process, a certain DNA sequence will ligate together with another DNA sequence in order to produce a new sequence.

Polymerase Chain Reaction (PCR)

Polymerases perform several functions, including the repair and duplication of DNA. Polymerase Chain Reaction (PCR) is a process that quickly amplifies the amount of specific DNA molecules in a given solution, using primer extension by polymerase. Each cycle of the reaction doubles the quantity of



Fig. 1. Droppers for spoiding and hybridizing

this molecule, leading to an exponential growth in the number of sequences. It consists of the following key processes:

1. *Initialization*: a mix solution of template, primer, deoxynucleotide-triphosphate (dNTP) and enzyme is heated to 94–98°C for 1–9 minutes to ensure that most of the DNA template and primers are denatured;
2. *Denaturation*: heat the solution to 94–98°C for 20–30 seconds for separation of DNA duplexes;
3. *Annealing*: lower the temperature enough (usually between 50–64°C) for 20–40 seconds for primers to anneal specifically to the single-strand DNA (ssDNA) template;
4. *Elongation/Extension*: raise temperature to optimal elongation temperature of *Taq* or similar DNA polymerase (70–74°C) for the polymerase adds dNTP's from the direction of 5' to 3' that are complementary to the template;
5. *Final Elongation/Extension*: after the last cycle, a 5–15 minute elongation may be performed to ensure that any remaining ssDNA is fully extended.

Steps 2 through 4 are repeated 20–35 times; less cycles result in less product, too many cycles increases the proportion of incomplete and erroneous products. PCR is a routine job in the laboratory that can be performed by an apparatus called a *thermal cycler*.

Denaturation Temperature Gradient PCR

Denaturation temperature gradient PCR (DTG-PCR) is a modified PCR method in which the denaturation temperature changes with cycle [22]. In DTG-PCR, conventional PCR is performed where the temperature of the denaturation step (step 2 of the aforementioned PCR procedure) is gradually increased.

Quantitative PCR

Quantitative PCR (Q-PCR) is a modification of the PCR used to rapidly measure the quantity of DNA, complementary DNA (cDNA) or Ribonucleic Acid (RNA) present in a sample. It may be used to determine if a DNA sequence is present in a sample, as well as the number of copies produced in PCR.

Affinity Separation

The objective of the affinity separation process is to verify whether each datum includes a certain sequence. This process permits single strands containing a given subsequence v to be filtered out from a heterogeneous pool of other sequences. After synthesizing strands complementary to v and attaching them



Fig. 2. Electrophoresis

to magnetic beads, the heterogeneous solution is passed over the latter. Those strands containing v anneal to the complementary sequence and are retained; those strands not containing v pass through and are discarded.

Normally, in this process a double-stranded DNA is incubated with the Watson-Crick complement of data that is conjugated to magnetic beads. A bead is attached to a fragment complementary to a sub-string, then a magnetic field used to extract all the DNA fragments containing such a sequence. The process is then repeated.

Gel Electrophoresis

Gel electrophoresis is an important technique for sorting DNA strands by their size [4]. Electrophoresis enables charged molecules to move in an electric field, as illustrated in Fig. 2. Basically, DNA molecules carry negative charge. Thus, when we place them in an electrical field, they tend to migrate towards a positive pole. Since DNA molecules have the same charge per unit length, they all migrate with the same force in an electrophoresis process. Smaller molecules therefore migrate faster through the gel, and can be sorted according to size (usually agarose gel is used as the medium here). At the end of this process the resultant DNA is photographed, as indicated in Fig. 3.

3 Comparison with Conventional Computing

Now DNA computing employs completely different tactics when allocating an independent letter code (such as ATCG, GTAC or CAAC) to each sample. Next, DNA sequences corresponding to the number of possible combinations are prepared. After they are hybridized in super-parallel fashion, the remaining DNA fragments are amplified to obtain an answer sequence – note that this procedure is carried out only once [17].



Fig. 3. Camera

The main benefit of using DNA computation to solve complex problems is that all possible solutions are created *concurrently* – in other words, it offers massively parallel processing. By contrast, humans – as well as most electronic computers – solve problems in a step-by-step manner (in other words, sequentially). DNA provides other benefits, including low cost, and energy efficiency [2].

The main steps in DNA computing are:

1. *Separate* (T, s): this operation separates a given set T into the set $+(T, s)$ of characters, including character string s and the set $-(T, s)$ of character strings that do not contain character string s . This operation corresponds to abstract experimentation on DNA molecules in a test tube.

2. *Mix*: this operation mixes sets T_1 and T_2 into the union set $T_1 \cup T_2$. This operation corresponds to mixing test tubes T_1 and T_2 .
3. *Detect* (T): this operation returns ‘YES’ if the test tube T is not empty, and ‘NO’ if it is empty. The operation corresponds to an experimental treatment procedure that detects the existence of DNA molecules by the electrophoretical fluorescent method.
4. *Amplify* (T): this operation corresponds to creating multiple sets T_1 and T_2 with the same contents as the given set T . This corresponds to an experimental treatment that amplifies the amount of molecules using polymerase chain reaction (PCR).

Now from the perspective of DNA computing, the most important characteristic of a DNA molecule is its Watson-Crick complementarity.

In Adleman’s model, a set of character strings – computed using hybridization – is computed according to the four steps described above. Using this computation, an NP-complete problem can be solved using an algorithm based on production-detection PCR. DNA computers can be used to solve real-world problems using this method.

4 Applications of DNA Computing

The theory of DNA computing is discussed in [2, 27, 29]. DNA computing has been applied to various fields, including nanotechnology, combinatorial optimization [25, 26], Boolean logic circuit development [27], and of particular relevance to the present Chapter, scheduling [18, 20, 23, 29, 33].

5 Approaches to Optimization and Scheduling

We have a long history of mathematical approaches for solving optimization problems. However there are limitations with such methods, resulting in many problems remaining unsolved. Beyond such mathematical methods, ‘problem solving’ in general attempts to mimic human or empirical approaches as a ‘rule-of-thumb’. This became prevalent after the von Neuman computer was invented in 1945. Subsequently, this led to a ‘golden era’ in artificial intelligence (AI) during the late 1970s and early 1980s, even though logic approaches (including production systems, predicate logic, semantic networks, and frame systems), present their own challenges, and can lead to combinatorial explosion. On the other hand, in order to mimic human thinking, many other methods have been proposed, including fuzzy systems, genetic algorithms (GAs), chaotic systems, artificial neural networks (ANNs), and so on – which are collectively referred to as ‘soft computing’. If we intend to solve large-scale, real-world problems, we need to overcome this issue of combinatorial explosion. In short, NP-complete problems cannot be solved using present-day silicon-based computers.

Genetic Algorithm (GA)

A genetic algorithm (GA) is a soft computing technique modeled on genetic mechanisms within biological organisms, and which searches for optimal values, obeying a number of simple constraints in each successive generation. GAs have been successfully applied to elevator group management [8, 15]. The setting of parameters in such elevator group control systems is difficult to achieve manually; [10] demonstrated that GAs are able to provide good parameter settings.

Artificial Neural Network (ANN)

In artificial neural networks (ANNs), optimization of an evaluation function can be implemented using backpropagation (BP) learning based on past transactions [28, 32].

Fuzzy Logic and Other Soft Computing Approaches

In the fuzzy logic approach, the ease of update rules comes as a welcome relief in comparison with GAs and/or ANNs [16, 19]. Other soft computing methods that have been successfully applied to elevator group control optimization include reinforcement learning agents [11], evolutionary strategies [7], and genetic network programming [12–14].

New Demands

Recently, new generation elevator systems – such as the elevator group supervisory control system (EGSCS) [5, 6] – have been developed to satisfy the various needs of users and to enable the ‘verticalization’ of buildings [3]. These new types of elevator system place additional constraints on group management. When searching a large number of alternatives, they require fast processing and large computer overhead. Accordingly, group management systems have been intensively studied in order to improve elevator transportation efficiency and convenience. Usage conditions of an elevator system are changed depending on time and customers. Business people don’t like to wait very long, whereas hotel guests do not like crowded elevators, even if they *are* transported quickly. Elevator systems are required to fulfill passengers’ different preferences.

6 Elevator Management System

Multiple elevators are commonly used in high-rise buildings (‘skyscrapers’). Effective control of such multiple elevators is essential. The overall aim in controlling a group of elevators is to satisfy the time constraints of all passengers,

at the same time providing the most efficient system. The basic problem is to decide *which* elevator should stop at a particular floor where passengers are waiting to go up(down).

Even in peak (rush) hours, it is possible to find *all* elevators moving in the same direction, or alternatively all elevators arriving simultaneously at the same floor. In order to resolve such situations, all elevators need to be optimally assigned to passengers, regardless of the latter's changing arrival times at the various floors in the multi-storey building.

The group control system selects elevator movement patterns according to random changes in traffic volumes and/or driving management, or in the eventuality of an accident. Such group control realizes comfortable, safe and economical management of elevators.

Suppose that a building has N floors and m elevators. Table 1 shows the current position of each elevator and the destinations of passengers in each elevator, together with the intended travel direction of passengers waiting on each floor. Figure 4 illustrates this situation graphically: the left-hand graph shows an upward movement, and the graph on the right-hand side a downward movement, respectively. Elevators A , B and M stop at floors 2, $N - 1$, and 3, respectively. On the other hand, there are people on floor 1 who desire to go up, and people on floors $N - 1$ and N who desire to go down. At time t , Elevator A at floor 2 has passengers who are going to floors 3, 4 and $N - 1$. Furthermore, there are passengers on floors N , $N - 1$ and 1 who are going down, down, and up, respectively.

Accordingly, in this research, the input information to the elevator management system is as follows:

1. The present position of each elevator;
2. The destination floors of each elevator (the required floor numbers where passengers in each elevator are going are input to the system);
3. The floors from which each elevator have been called (people on each floor can indicate their direction but they cannot input their destination floor).

Table 1. Elevator information at time- t

Floor	Queuing	Elevator-A	Elevator-B	...	Elevator- m
N	↓				
$N - 1$	↓		(1,3,5)		
⋮					
3					(4, $N - 1, N$)
2		(3,4, $N - 1$)			
1	↑				

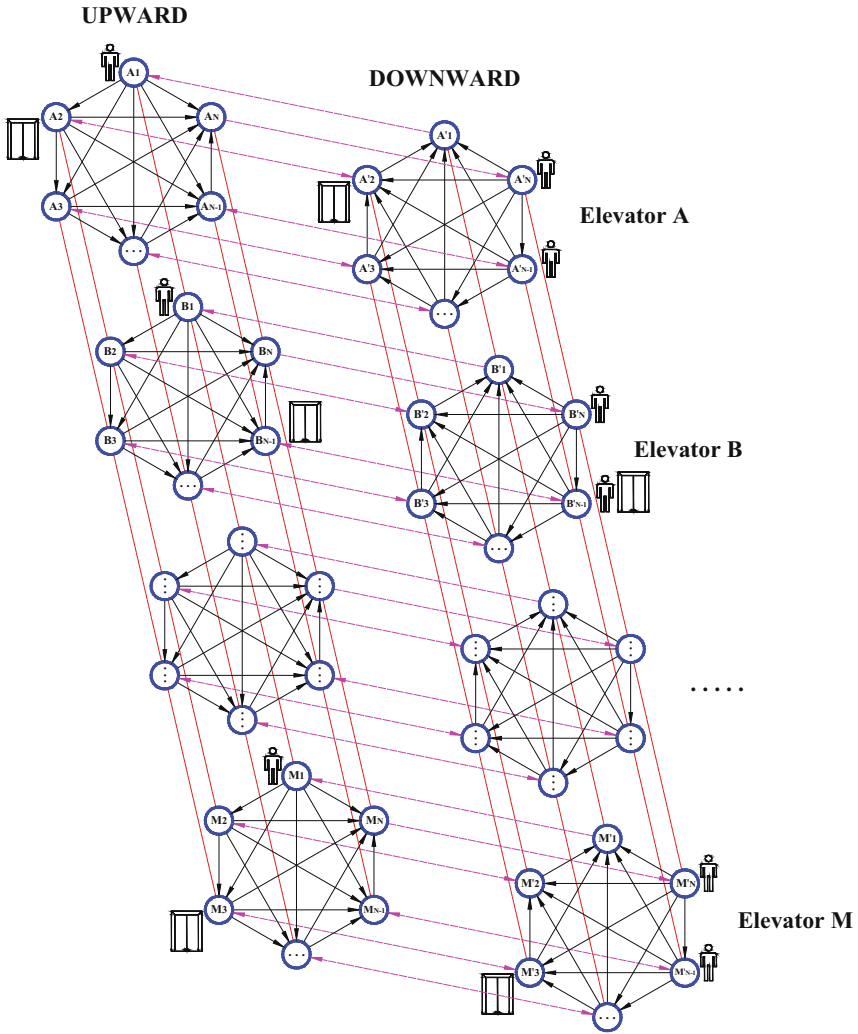


Fig. 4. Whole paths of M elevators

As previously mentioned, Table 1 shows that Elevator- A is at Floor 2 and has passengers who are going to floors 3, 4 and $N - 1$; likewise Elevator- B is at Floor $N - 1$ and has passengers travelling down to Floors 1, 3 and 5. There are people who are going up on Floor 1, down on Floor $N - 1$, and down on Floor N . Based on this information, the challenge is to efficiently manage *all* elevators.

6.1 Restrictions on Elevator Movements

The problem is to determine the optimal scheduling of *all* m elevators – in other words to provide the shortest overall wait time – given the wait queue

and the initial position of each elevator at any time t . Let us denote the following variables:

- $|d(m) - o(m)|$ is the total number of floors moved
- C_t is the time cost of an elevator traveling between adjacent floors
- C_s is the time cost of an elevator stopping at a floor

The elevator moves between floors – denoted by m – must be consecutive, in other words,

$$d(m_i) = o(m_{i+1}) \vee 1 \leq i < N \tag{1}$$

where $o(m)$ is the original floor, and $d(m)$ is the destination floor

The traveling time T between floors can be represented as

$$T(|d(m) - o(m)|) = \begin{cases} [|d(m) - o(m)|]C_t + C_s & m = \text{destination,} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The output of the graph G given by the sum of the costs thus represents the total travel time of elevator- E , that is

$$G(E) = \sum_{m=1}^N T(|d(m) - o(m)|) \tag{3}$$

For a building with M elevators, the graph of single elevator movements shown in Table 2 can be duplicated M times in representing the whole paths of elevator travel. The total traveling time of M elevators can thus be calculated by summing the traveling time of each single elevator as

$$G(E_1, E_2, \dots, E_{M-1}, E_M) = \sum_{k=1}^M G(E_k) \tag{4}$$

The optimal travel route – denoted O – is thus given by the minimum total traveling time of all elevators with all initial conditions and requirements satisfied, namely

$$O = \min\{G(E_1, E_2, \dots, E_{M-1}, E_M)\} \tag{5}$$

Table 2. Elevator management information

Floor	Calling	Elevator-A	Elevator-B
6			(2,3)
5	↓		
4	↑		
3	↓		
2			
1		(3,5)	

6.2 Elevator Scheduling

Now since we have m elevators, we can duplicate m graphs and connect between all vertices in the one graph. Figure 5 shows the case where $m = 2$.

The left-hand side of Fig.5 illustrates all paths which are going up, and the right-hand side all paths going down. Elevator-A at floor 1 can move to all floors from 2 to n . However when Elevator-A is at floor j , it can move to all floors from $j + 1$ to n , but cannot move down, for example to floors 1 through i . The right-hand side graph shows the same situation concerning the downward movement of the same Elevator- i . The connections between both graphs show the changing of direction from down to up (or from up to down). These changes of direction happen on all floors. The connections between elevators on the same floor show that passengers can move from one elevator to another (but these movements are not considered in the computations because they depend on passenger preferences).

It is therefore sufficient that one of the m elevators may reach the calling floor on the line of graph A, B, \dots, m . Let us construct a graph for each case where elevator-A or elevator-B, \dots , and elevator- m reach a floor where the button has been pushed.

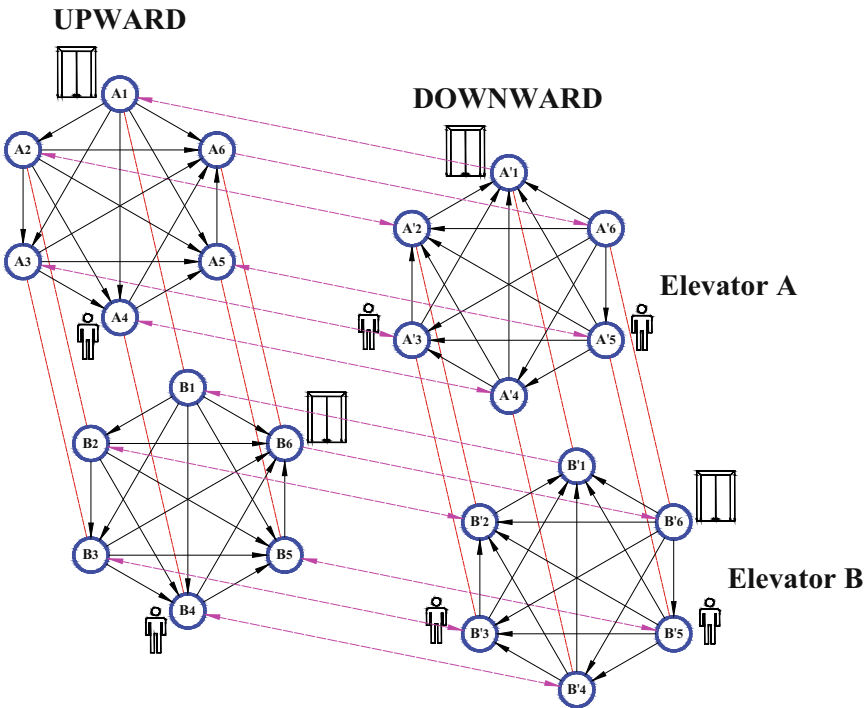


Fig. 5. Whole paths of two elevators

Considering all combinations, let us calculate the shortest path of each graph A, B, \dots, m . Suppose that $f(1, 2, \dots, m)$ denotes the largest value of graphs A, B, \dots, m . By calculating all combinations $f_x(A, B, \dots, m)$, we can obtain the optimal allocation of elevators by selecting the minimum value of $f_x(A, B, \dots, m)$, where x denotes the number of combinations. For example, when the number of elevators is 2 and the number of calling floors is 3, the number of combinations is 2^3 .

It is possible to represent elevator- B by a graph, just as we did in the case of elevator- A . Figure 5 illustrates the graph of all paths of the two elevators A and B .

Now, elevator- A is currently at floor-1 and its destination floors are 3 and 5. The destination floors of elevator- B at floor-6 are 2 and 3. There are also upward calls. Figure 5 shows the floors with a symbol where one or both elevators should stop.

In this case, if the destination floor has been decided for elevator A or B , then the other one is automatically assigned to the other destination floor. As a result, it is necessary to calculate the optimal paths for *two* kinds of graphs for elevators A and B . The larger value for both elevators A and B is denoted by $f_x(A, B)$.

The problem here is to select the smallest value – $\min(f(A, B), (x = 1, 2, \dots, 8))$. The obtained schedule which gives the smallest value is the optimal solution for elevators A and B . There are 16 kinds of graph which can be obtained from 8 combinations, as shown in Figs. 4 and 5. The next Section shows how to calculate the shortest path schedule for both elevators.

7 Bio-Soft Computing Based on DNA Length

In DNA computation, each base sequence is assigned to a floor, as shown in Table 3.

Let us denote the connection between two base sequences corresponding to each floor as the movement between two floors. Further, let us assign an

Table 3. Correspondence between floors and base sequences

1	2	3	4	5	6
AAAA	CCCC	TTTT	ATAT	GAGA	GGGG
1'	2'	3'	4'	5'	6'
CACA	TCTC	TGTG	GCGC	CAGT	GATC

appropriate length of DNA sequence to the edge weight:

$$\begin{aligned}
 rll\psi_k &= f(k) + T_E \\
 \psi_1 &= f(1) + T_E \text{ string of two random characters}(ZZ) \\
 &\vdots \\
 \psi_5 &= f(5) + T_E \text{ string of ten random characters} \\
 &\quad (YYYYYYYYYYYYYYYYYYYY) \tag{6}
 \end{aligned}$$

$$\begin{aligned}
 f(A, B) &= \max(4 + 4 + 4 + 4 + 4, 4 + 2 + 4 + 4 + 4 + 2 + 4 + 4 + 4 + 4) \\
 &= \max\{20, 36\} = 36 \tag{7}
 \end{aligned}$$

These values show the time spent moving between two floors, which are combined in the base sequence.

In this problem all movements between floors comprise 40 roots. Let us produce DNA sequences corresponding to these roots in Table 4.

Let us produce DNA fragments corresponding to a floor in Table 3 and DNA fragments corresponding to a root in Table 4. Next, place these DNA fragments and combining polymerase in the same test tube and store the test tube at an appropriate temperature; all combinations will be automatically created.

Various DNA sequences are automatically created by combining fragments shown as each floor, and filaments shown as each movement root, respectively, in Table 4. These DNA sequences correspond to combinations of feasible solutions. In order to solve the graph shown in Fig. 5, DNA sequences which have ‘AA’ (the former two characters upward at the first floor) at the start, and ‘GA’ (the latter two characters upward at the fifth floor) at the end are detected, out of the many DNA sequences, using various polymerase.

As we know the floors where elevators should stop, only DNA sequences with $AA * TTTT * GA$ are selected – in other words those where the DNA sequences start with ‘AA’, pass through ‘TTTT’, and terminate at ‘GA’. Then, the shortest DNA sequence shows the optimal solution which starts at the 1st floor, stops at the 3rd floor, and reaches the 6th floor. This procedure can be abstracted by the weight of a DNA sequence, since long DNA sequences are ‘heavy’ and short DNA sequences are ‘light’. At the end we check the DNA sequence and convert it to a floor number. The shortest roots for graphs 5 through 12 contain the following DNA sequences, and the length of these DNA sequences can be calculated.

The shortest sequence is the schedule where elevator-*A* stops at the 4th floor, and elevator-*B* stops at the 3rd and 5th floors. Therefore, the optimal schedule for elevators-*A* and *B* is obtained for the present state of elevators and calling floors. If this computation is pursued to obtain the optimal schedule whenever buttons are pushed at a calling floor, the schedule with the shortest wait time will always be obtained.

Table 4. Representation of roots by DNA sequence (edge DNA oligonucleotides)

1 → 2	<i>AZZCC</i> <i>TTEEGG</i>	6' → 5'	<i>GAZZGT</i> <i>CTEECA</i>
1 → 3	<i>AAWWWTT</i> <i>TFFFFAA</i>	6' → 4'	<i>GAWWWWCG</i> <i>CTFFFFGC</i>
1 → 4	<i>AAVVVVVAT</i> <i>TTHHHHHHTA</i>	6' → 3'	<i>GAVVVVVVTG</i> <i>CTHHHHHHAG</i>
1 → 5	<i>AAXXXXXXXXXGA</i> <i>TTIIIIIIICT</i>	6' → 2'	<i>GAXXXXXXXXXXTC</i> <i>CTIIIIIIIIAG</i>
1 → 6	<i>AAYYYYYYYYYGG</i> <i>TTJJJJJJJJCC</i>	6' → 1'	<i>GAYYYYYYYYYYCA</i> <i>CTJJJJJJJJGT</i>
2 → 2'	<i>CCTC</i> <i>GGAG</i>	5' → 5	<i>CAZZGA</i> <i>GTEECT</i>
2 → 3	<i>CCZZTT</i> <i>GGEEAA</i>	5' → 4'	<i>CAZZCG</i> <i>GTEEGC</i>
2 → 4	<i>CCWWWWAT</i> <i>GGFFFFTA</i>	5' → 3'	<i>CAWWWWTG</i> <i>GTFFFFAC</i>
2 → 5	<i>CCVVVVVGA</i> <i>GGHHHHHHCT</i>	5' → 2'	<i>CAVVVVVVTC</i> <i>GTHHHHHHAG</i>
2 → 6	<i>CCXXXXXXXXXGG</i> <i>GGIIIIIIICC</i>	5' → 1'	<i>CAXXXXXXXXXXCA</i> <i>GTIIIIIIIGT</i>
3 → 3'	<i>TTTG</i> <i>AAAC</i>	4' → 4	<i>GCAT</i> <i>CGTA</i>
3 → 4	<i>TTZZAT</i> <i>AAEETA</i>	4' → 3'	<i>GCZZTG</i> <i>CGEEAC</i>
3 → 5	<i>TTWWWWGA</i> <i>AAFFFFCT</i>	4' → 2'	<i>GCWWWWTC</i> <i>CGFFFFAG</i>
3 → 6	<i>TTVVVVVGG</i> <i>AAHHHHHHCC</i>	4' → 1'	<i>GCVVVVVVCA</i> <i>CGHHHHHHGT</i>
4 → 4'	<i>ATCG</i> <i>TAGC</i>	3' → 3	<i>TGTT</i> <i>ACAA</i>
4 → 5	<i>ATZZGA</i> <i>TAEECT</i>	3' → 2'	<i>TGZZTC</i> <i>ACEEAG</i>
4 → 6	<i>ATWWWWGG</i> <i>TAFFFFCC</i>	3' → 1'	<i>TGWWWWCA</i> <i>ACFFFFGT</i>
5 → 5'	<i>GACA</i> <i>CTGT</i>	2' → 2	<i>TGCC</i> <i>CTGG</i>
5 → 6	<i>ATZZGG</i> <i>TAEECC</i>	2' → 1'	<i>TGZZCA</i> <i>CTEEGT</i>
6 → 6'	<i>GGGA</i> <i>CCCT</i>	1' → 1	<i>CAAA</i> <i>GTTT</i>

8 Bio-Soft Computing with Fixed-Length DNA

For the elevator dispatching problem, an N -story building equipped with M identical elevator cars given by up calls, down calls, and car calls. The optimal route is given by $\min\{G(1, 2, \dots, N)\}$.

The key factor to cost sequence design is the T_m of a DNA strand. The concept is to design the DNA sequences that have heavier weights with higher T_m than those lighter weights. DNA amplification and detection techniques often depend on oligonucleotide T_m . The T_m of a DNA duplex is defined as the temperature where one-half of the nucleotides are paired and one-half are unpaired [34]. The T_m indicates the transition from double helical to random coil formation and is related to the DNA GC base content [24]. Usually expressed as a percentage, it is the proportion of GC-base pairs in the DNA molecule or genome sequence being investigated. GC-pairs in the DNA are connected with three hydrogen bonds instead of two in the AT-pairs, which makes the GC-pair stronger and more resistant to denaturation by high temperatures. In our encoding scheme, the DNA sequences that represent floor nodes are fixed length, and costs are distinguished by T_m of the given DNA strands. This design makes an oligonucleotide with lighter weight, which means that more economical path tend to have a lower T_m .

All the possible solutions are randomly generated by DNA hybridization and ligation with the oligonucleotides representing floors, edges, and costs. To satisfy the conditions of an elevator dispatching problem, the route must begin and end at a specified node, and the route must pass by each consecutive floor until it reaches the final destination.

The PCR can be applied to test the former requirement, which is a technique for amplifying DNA that rapidly synthesizes many copies of a specific DNA segment by providing specific complementary sequences (primers) and enzymes (DNA polymerases – for example, *Pfu* and *Taq*). The DNA strands corresponding to the original floor and the complement of the final destination floor are used as two primers in two successive PCRs to reproduce the routes.

To test the latter requirement, agarose gel electrophoresis is applied. Agarose gel electrophoresis is a method to separate DNA strands by size, and to determine the size of the separated strands by comparison with strands of known length. All PCR products are sieved by agarose gel electrophoresis, and unreasonable lengths are excluded. To verify the DNA strands pass by every consecutive floor, the product from the above step is affinity-purified with a biotin-avidin magnetic bead system.

To solve the elevator routing problem with our molecular algorithm, note that all possible end paths of elevator- i are joined with the start path of elevator- $(i + 1)$, so that the total output of the graph $G(1, 2, \dots, N)$ – representing the travel route of all elevators – can be calculated.

DTG-PCR is a specified PCR protocol that modifies the denaturation temperature profile. If the denaturation temperature is decreased to a certain level in PCR, the DNA strands with denaturation temperatures lower than that temperature will be denatured and amplified. As the denaturation temperature is increased cycle-by-cycle in PCR, other DNA strands with higher denaturation temperature will also be amplified. However, the economical

paths that have lighter weights will be amplified more and will occupy the major part of the solution and hence can be easily detected [22]. Based on the electrophoretic mobility of DNA strands in different T_m , the temperature gradient gel electrophoresis (TGGE) is applied to detect the the most economical route among other possibilities resulting from DTG-PCR.

The proposed bio-soft computing algorithm for solving the elevator dispatching problem is summarized as follows:

Algorithm 1 Bio-soft Elevator Dispatch Algorithm

0. Design the fixed-length DNA sequences with thermodynamic control of weight;
 1. Generate a random pool of solutions by hybridization and ligation;
 2. Retrieve the strand that satisfies the conditions of the elevator dispatching problem by PCR and gel electrophoresis;
 3. Verify that the strands pass by every consecutive floor by affinity purification;
 4. Join the strands of elevators by ligation;
 5. Sieve the economical path by DTG-PCR;
 6. Detect the most economical path by TGGE;
 7. Read out the most optimal route by DNA sequencing.
-

8.1 Empirical Study

A six-story building equipped with two identical elevator cars, A and B , is considered in this empirical study. As illustrated in Table 2, elevator- A is currently at the 1st floor, moving upwards to answer calls on the 3rd and 5th floors; elevator- B is at the 6th floor, moving down to answer calls on the 2nd and 3rd floors. In addition, calls have been requested on the 3rd, 4th, and 5th floors to go down, up, and down, respectively. The objective is to find the optimal route for all elevators that fulfills all initial conditions and subsequent requirements. The optimal route will be given by $\min\{G(A, B)\}$.

As shown in Table 5, each floor node is randomly associated with a 20-mer sequence of ssDNA, denoted by F_i , which has a similar melting temperature due to node sequences contributing equally to the thermal stability of the paths. Weight sequences are designed to have different T_m depending on the weights; the lighter the weight, the lower the T_m . In other words, a more economical path has lower T_m . The edge between consecutive floors is generated by a partial beginning node, a cost sequence, and a partial ending node. For each floor movement (edge) $i \rightarrow j$ in the graph, an oligonucleotide $F_{i \rightarrow j}$ is created that is the 3' 10-mer complement of F_i followed by the cost sequence of path length, and then the 5' 10-mer complement of F_j . The edge from floor 1 to floor 2, for example, the 3' 10-mer complement of F_1 : 'CATGACAACG' is followed by the cost sequence of C_t : 'ATCTTGGATTATTACCAAG', then the 5' 10-mer complement of F_2 : 'TGGCTACATG', as illustrated in Fig. 6.

Table 5. DNA sequences associated with each floor node and cost for the six-floor elevator routing problem

	DNA Sequence (5' → 3')	T_m (°C)	GC Content (%)
Floor Nodes			
F_1	TCCTCGTTAGGTACTGTTGC	46.02	50
F_2	ACCGATGTACCTCTCAATGC	46.40	50
F_3	TGGTCAGCTAATGACGTGAG	46.42	50
F_4	GCGGTTCTAAATCCGTCAC	46.51	50
F_5	ATTGGACCCAGATGCAAAGG	46.92	50
F_6	GTTAGACCTCGCGTTGCTAT	46.97	50
F'_1	GCGTAATCGTATCCGTGAGA	46.58	50
F'_2	TAGCCTTACGTACCGGCTTA	46.84	50
F'_3	CCGTAACGTATAGCGATGGA	46.22	50
F'_4	GACGGTATTGCGTAATTCGG	46.48	50
F'_5	ATCGGAATCGATCCGTATGC	46.88	50
F'_6	AGCTGGGATAAGGCATACCA	46.76	50
Costs			
C_t	ATCTTGGATTATTACCAAG	36.98	30
C_s	GAGCCGACCAGCGACACCCA	55.84	70

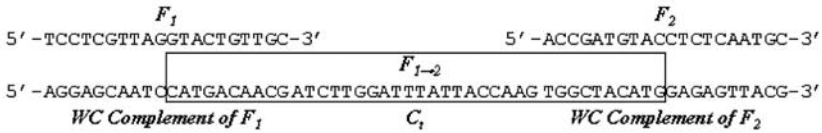


Fig. 6. Encoding scheme example: an oligonucleotide $V_{1 \rightarrow 2}$ is created that is the 3' 10-mer of V_1 , followed by the weight sequence of path length, and then the 5' 10-mer of V_2

In this study, the nearest-neighbor (N-N) model is applied to calculate the T_m , which is the most accurate method for predicting the T_m of oligonucleotide DNA through interactions between neighboring bases. The enthalpy (ΔH) and entropy (ΔS) of adjacent bases is considered in the formula [30]. The T_m in this study was calculated with the initial concentration of $1nM$ oligonucleotide and $50mM$ salt.

The proposed fixed-length DNA-based algorithm for solving the elevator dispatching problem began from generating a random pool of possible routes by the hybridization of DNA strands that represent the floors and edges. All possible paths of the elevator dispatching problem were generated simultaneously under the massive parallelism of DNA molecules. Each F_i and edge $i \rightarrow j$ were mixed in a single ligation reaction. As per [22], the added amount of an edge was varied according to weight – namely the weight increased, the amount was decreased. The oligonucleotide mixture was heated to $95^\circ C$ and

cooled to 20°C at 2°C/min for hybridization. The reaction mixture was then subjected to a ligation.

Conventional gel electrophoresis excluded the unreasonable length of DNA strands from the candidate pool. Then, the DNA strands that did not pass by every floor nodes between origin and destination were excluded by affinity separation. The complement of F_1 was conjugated to magnetic beads so that only those ssDNA molecules which contained the sequence F_1 annealed to the bound were retained. This process was repeated until each floor node was verified.

In DTG-PCR, the denaturation temperature started low (70°C) in the beginning cycles of PCR, lower than the T_m of the template strands. Then, the denaturation temperature was gradually increased until it reached 95°C and maintained at the same temperature for the remaining cycle. After this process, one main band was observed in the gel which contained two different DNA strands of the possible routes, as shown in Table 6. These strands, however, were of the same length and cannot be separated by conventional gel electrophoresis.

Nevertheless, from the algorithm design, the weights have distinct behaviors in T_m and thus the more economical path would have a lower T_m . Thus,

Table 6. Result from DTG-PCR; two different DNA strands represent the possible routes of the same length which cannot be separated by conventional gel electrophoresis

DNA Sequence (5' → 3')	Oligo Length (Bases)
TCCTCGTTAGTACTGTTGCATCTTGGATTATTACCAAG ACCGATGTACCTCTCAATGCGAGCCGACCAGCGACACCCA TGGTCAGCTAATGACGTGAGGAGCCGACCAGCGACACCCA GCGGTTCTAAATTCGGTCACGAGCCGACCAGCGACACCCA ATTGGACCCAGATGCAAAGGAGCTGGGATAAGGCATACCA GAGCCGACCAGCGACACCCAATCGGAATCGATCCGTATGC ATCTTGGATTATTACCAAGGACGGTATTGCGTAATTCGG GAGCCGACCAGCGACACCCACCGTAACGTATAGCGATGGA GAGCCGACCAGCGACACCCATAGCCTTACGTACCGGCTTA	360
TCCTCGTTAGTACTGTTGCATCTTGGATTATTACCAAG ACCGATGTACCTCTCAATGCGAGCCGACCAGCGACACCCA TGGTCAGCTAATGACGTGAGGAGCCGACCAGCGACACCCA GCGGTTCTAAATTCGGTCACGAGCCGACCAGCGACACCCA ATTGGACCCAGATGCAAAGGAGCTGGGATAAGGCATACCA ATCTTGGATTATTACCAAGATCGGAATCGATCCGTATGC ATCTTGGATTATTACCAAGGACGGTATTGCGTAATTCGG GAGCCGACCAGCGACACCCACCGTAACGTATAGCGATGGA GAGCCGACCAGCGACACCCATAGCCTTACGTACCGGCTTA	360

Elevator A		Elevator B		T_m ($^{\circ}\text{C}$)	GC Content(%)
1	→ 2 → 3 → \triangle → 5	6	→ ∇ → 4 → ∇ → 2	82.93	54.44
1	→ 2 → 3 → \triangle → ∇	6	→ 5 → 4 → ∇ → 2	81.92	52.22

Fig. 7. The distinct behavior in melting temperature among possible optimal solutions. \circ : car call; \triangle : up hall call; ∇ : down hall call

TGGE can be used to filter the DNA strands that have lowest T_m from other strands of the same length. Based on the correlation of the melting characteristic of a DNA strand to its electromigration, the DNA strand of the most economical route would travel fastest in gel; hence, it can be distinguished from other possible routes. The T_m and its GC content among those possible routes are shown in Fig. 7. More specifically, the DNA strands corresponding to the route for elevator A: ‘1 → 2 → 3 → 4 → 5’ that answers the hall call at the 4th floor and 5th floor for up and down, respectively, and elevator B: ‘6 → 5 → 4 → 3 → 2’ that answers the hall call at the 3rd floor for down – in other words, the optimal solution.

9 Conclusion

The following is the famous *uncertainty principle* discovered by Heisenberg in 1927:

“It is not the world that attracts attention now and that a usual physical law sways in the minute (nano) world but the world of a quantum-mechanics-law.”

At present computers are built on the Deterministic Turing Machine (DTM) model; they have yet to be realized modeled on the newer Quantum Turing Machine (QTM) concept. Although this type of computer has yet to be produced in the real world, present-day (von Neumann, semiconductor type) face a wall of combinatorial problems.

There remain several problems with DNA computing however, these being:

- ‘Preparation’ and ‘extraction’ take too much time, and
- errors occur in copying DNA.

Despite such problems, we fully expect that at some time in the future, DNA computers will replace present-day (silicon-based) ones.

Acknowledgement

The author would like to express his gratitude to Professor John Fulcher for his assistance in preparing this Chapter.

References

1. Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science*, 266: 1021–1024.
2. Adleman LM (1998) Computing with DNA. *Scientific American*, 279(2): 54–61.
3. Amano M, Yamasaki M, Ikejima H (1995) The latest elevator group control system. In: Barary GC (ed.) *Elevator Technology 6 – Proc. ELEVCON'95*, March, Hong Kong. Intl. Association Elevator Engineers, Essex, UK: 88–95.
4. Amos M, Paun G, Rozenberg G, Salomaa A (2002) Topics in the Theory of DNA Computing. *J. Theoretical Computer Science*, 287: 3–38.
5. Barney G, dos Santos S (1985) *Elevator Traffic Analysis, Design and Control (2nd ed)*. Peter Peregrinus, London, UK.
6. Barney G (2003) *Elevator Traffic Handbook*, Spon Press, London, UK.
7. Beielstein T, Ewald C-P, Markon S (2003) Optimal elevator group control by evolution strategies. In: Cantú-Paz E *et al.* (eds.) *Proc. Genetic and Evolutionary Computation Conf. (GECCO'03)*, 12–16 Julu, Chichago, IL: 1963–1974.
8. Bi X, Zhu C, Ye Q (2004) A GA-based approach to the multi-objective optimization problem in elevator group control system. *Elevator World*, June: 58–63.
9. Binti R, Bakar A, Watada J, Pedrycz W (2006) A DNA computing approach to data clustering based on mutual distance order. In: Watada J (ed.) *Proc. 9th Czech-Japan Seminar on Data Analysis and Decision Making Under Uncertainty*, 18–22 August, Kitakyusyu and Nagasakidate, Japan: 139–145.
10. Cortes P, Larraneta J, Onieva L (2004) Genetic algorithm for controllers in elevator groups: analysis and simulation during lunchpeak traffic. *Applied Soft Computing*, 4: 159–174.
11. Crites R, Barto A (1998) Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33: 235–262.
12. Eguchi T, Hirasawas K, Hu J, Markon S (2004) Elevator group supervisory control system using genetic network programming. In: *Proc. IEEE Congress Evolutionary Computation (CEC'04)*, 19–23 June, Portland, OR. IEEE Press, Piscataway, NJ. 2: 1661–1667.
13. Eguchi T, Hirasawas K, Hu J, Markon S (2006) Elevator group supervisory control system using genetic network programming with functional localization. *J. Advanced Computational Intelligence and Intelligent Informatics*, 10(3): 243–244.
14. Eguchi T (2006) Study on optimization of elevator group supervisory control system using genetic network programming, *PhD Dissertation*, Graduate School of Information, Production and Systems, Waseda University, Kitakyushu, Japan.
15. Fujino A, Tobita T, Segawa K, Yoneda K, Togawa A (1997) An elevator group control system with floor-attribute control method and systems optimization using genetic algorithms. *IEEE Trans. Industrial Electronics*, 44(4): 546–552.
16. Gudwin R, Gomide F, Netto M (1998) A fuzzy elevator group controller with linear context adaptation. In: *Proc. Fuzzy-IEEE98, WCCI'98-IEEE – World Congress Computational Intelligence*, 4–9 May, Anchorage, AL. IEEE Press, Piscataway, NJ: 481–486.
17. Ito Y, Fukusaki E (2004) DNA as a ‘nanomaterial’. *J. Molecular Catalysis B: Enzymatic*, 28: 155–166.

18. Jeng D J-F, Watada J, Kim I (2007) Solving a real time scheduling problem based on DNA computing. *Soft Computing J.* (in press).
19. Kim C, Seong K, Lee-Kwang H, Kim JO (1998) Design and implementation of a fuzzy elevator group control system. *IEEE Trans. System, Man and Cybernetics – PART-A*, 28(3): 277–287.
20. Kim I, Jeng D J-F, Watada J (2006) Redesigning subgroups in a personnel network based on DNA computing. *Int. J. Innovative Computing, Information and Control*, 2(4): 885–896.
21. Lee JY, Zhang B-T, Park TH (2003) Effectiveness of denaturation temperature gradient-polymerase chain reaction for biased DNA algorithms. *Pre-Proc. 9th Intl. Meeting on DNA Based Computers*, Madison: 208.
22. Lee JY, Shin S-Y, Park TH, Zhang B-T (2004) Solving traveling salesman problems with DNA molecules encoding numerical values. *Biosystems*, 78(1): 39–47.
23. Lipton RJ (1995) DNA Solution of Hard Computational Problems. *Science*, 268: 542–545.
24. Marmur J, Doty P (1962) Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature. *J. Molecular Biology*, 5: 109–118.
25. Ouyang Q, Kaplan PD, Liu S, Libchaber A (1997) DNA solution of the maximal clique problem. *Science* 278: 446–449.
26. Owenson GG, Amos M, Hodgson DA, Gibbons A (2001) DNA-based logic. *Soft Computing*, 5(2): 102–105.
27. Paun GH, Rozenberg G, Salomaa A (1999) *DNA Computing: New Computing Paradigms*. Translated by Yokomori T (Japanese ed.) Springer-Verlag, Tokyo, Japan.
28. Powell BA, Sirag DJ, Withehall BL (2000) Artificial neural networks in elevator dispatching. *Lift Report*, 27(2): 44–57.
29. Rohani BAB, Watada J, Pedrycz W (2006) A DNA computing approach to data clustering based on mutual distance order. In: Watada J (ed.) *Proc. 9th Czech-Japan Seminar on Data Analysis and Decision Making Under Uncertainty*, 18–22 August, Kitakyusyu and Nagasaki, Japan: 139–145.
30. SantaLucia JJr (1998) A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. National Academy of Sciences*, 95: 1460–1465.
31. van Noort D (2004) Towards a re-programmable DNA computer. In: Chen J, Reif JH (eds.) *Proc. 9th Intl. Workshop DNA Based Computers (DNA9)*, Lecture Notes in Computer Science 2943, Springer-Verlag, Berlin: 190–196.
32. Wan H, Liu C, Liu H (2003) NN elevator group-control method. *Elevator World*, 2: 148–154.
33. Watada J, Kojima S, Ueda S, Ono O (2006) A DNA computing approach to optimal decision problem. *Int. J. Innovative Computing, Information and Control*, 2(1): 273–282.
34. Wetmur JG (1991) DNA probes: applications of the principles of nucleic acid hybridization. *Critical Reviews in Biochemistry and Molecular Biology*, 26(3): 227–259.
35. Winfree OE, Lin F, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693): 539–549.

Resources

1 Key Books

1.1 DNA Computing

Amos M (2005) *Theoretical and Experimental DNA Computation*. Springer-Verlag, Berlin.

Calude CA, Paun G (2001) *Computing With Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing*. Taylor and Francis, London, UK.

Deng Z, Chen Y, Tian Y, Mao C (2006) A fresh look at DNA nanotechnology. In: Chen J, Jonoska N, Rozenberg G (eds.) *Nanotechnology: Science and Computation*. Springer-Verlag, Berlin.

Madrona E (2000) *Global Distributed Applications With Windows DNA*. Artech House, Norwood, MA.

Paun GH, Rozenberg G, Salomaa A (1999) *DNA Computing: New Computing Paradigms*. Springer-Verlag, Berlin.

1.2 Elevator Management

Barney G, dos Santos SM (1985) *Elevator Traffic Analysis, Design and Control (2nd ed)*. IEEE Computer Society Press, Los Alamitos.

Barney G (2003) *Elevator Traffic Handbook*, Spon Press, London, UK.

2 Key Survey/Review Articles

Adleman LM (1998) Computing with DNA. *Scientific American*, 279(2): 54–61.

Lipton RJ (1995) DNA Solution of Hard Computational Problems. *Science*, 268: 542–545.

Mao C, LaBean TH, Reif JH (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407: 493–495.

Reif JH, LaBean TH (2007) Autonomous programmable biomolecular devices using self-assembled DNA nanostructures: Surveying recent developments in bio-DNA computing. *Communications ACM*, 50(9): 46–53.

Shapiro E, Benenson Y (2006) Bringing DNA computers to life. *Scientific American*, May: 45–51.

Bibliography of Molecular and Splicing Systems

<http://www.dcs.ex.ac.uk/~pf201/dna.html>

3 International Organization

International Society for Nanoscale Science, Computation and Engineering

<http://www.isnsce.org/>

4 Discussion Groups, Forums

P System Internet Forum

<http://www.cs.us.es/gcn/foro.htm>

5 Research Groups

EMCC European Molecular Computing Consortium

<http://openit.disco.unimib.it/emcc/?screen=1024x768x7>

Computational Biomodelling Laboratory, Finland

<http://combio.abo.fi/>

Japanese Molecular Project

<http://hagi.is.s.u-tokyo.ac.jp/MCP/>

Leiden Center for Natural Computation

<http://www.wi.leidenuniv.nl/home/lcnc/>

The P Systems Molecular Computing Consortium

<http://psystems.disco.unimib.it/European>

Hardware Implementation of P Systems

http://www.teuscher.ch/psystems*

The Molecular X-Machines Project (Department of Computer Science,
University of Sheffield, UK)

<http://www.dcs.shef.ac.uk/~bernardf/molxm/index.htm>

6 Key International Conferences and Workshops

<http://www.liacs.nl/home/pier/webPagesDNA/index.html>

7 Web Resource

Institute of Electronics, Information and Communication Engineers, Japan
elevator scheduling competition (in Japanese):

Multiple cars in multiple shafts (therefore each car can become an obstacle
for others, leading to potential deadlock). The challenge is to minimize the
sum of times from persons calling cars to persons reaching destination floors.

<http://www.ieice.org/~cst/compe07/>

The Next Generation of Immunity-Based Systems: From Specific Recognition to Computational Intelligence

Yoshiteru Ishida

Department of Knowledge-Based Information Engineering, Intelligent Sensing System Research Center, Research Center for Future Vehicle, Toyohashi University of Technology, Tempaku, Toyohashi 441-8580, Japan, ishida@tutkie.tut.ac.jp

1 Introduction

The post-genome era proved that DNA sequence data [11, 26] with structural and functional analysis on genes archived in many data bases can support in developing new bio-engineering technologies and can drive systemic views for biological systems. However, the post-genome era also proved that sequence data alone is not sufficient, but revealed that higher knowledge of the function of proteins is indispensable. Personalized medicine required not only sequence data, but further knowledge such as SNP (single nucleotide polymorphism) and of functioning of proteins and its deployment to interacting systems such as gene networks, giving birth of a new territory called proteome.

Considering such trends in the post-genome era, we propose possible directions for immunity-based systems (IMBS). One such approach is a constructive systems approach, taking fundamental properties of the component and trying to construct a fundamental function. The synthetic approach has been extensively studied [3, 5, 18, 24], to mention but a few). A constructive approach that assumes an intrinsic character of the components (such as antibodies), and constructs the fundamental function of the immune system from the component. Although it should not be limited to two, another possible direction of next generation immunity-based systems is to extend and enhance models and simulations to be operational: that is, involving medications as a control to the systems with the immune system and pathogen interactions. This would be made possible by using post-genome genetic data. The operational models and simulations allow, for example, involving the immune system in personalized medicine. Information of disease agents, medicine, and host agents are required for personalized medicine.

When functions are focused and more pathways are revealed, biological systems will be studied as a system of interacting components and processes.

Restricting discussions on immunity-based systems, the post-genome era naturally proceeds to study immune systems focusing not only on discovery of genes related to the immune system such as MHC (major histocompatibility complex), but also its systemic organization and the knowledge of the organism [9]. While on the other hand, studies on the immune system are indispensable for personalized medicine, since the immune systems is one important component for personalization of medicine, and such personal differences are integrated in the immune system. The other two components for personalized medicine are: pathogens and medications.

Too close mimicking and superficial analogy could be misleading, not only for biologically inspired systems but also for computing (circuits) that use biological system components. Biological systems can be neither simple nor optimal. One reason for apparently complex and roundabout implementations is that biological systems have large-scale interactions in a spatio-temporal sense. In space, they interact with an environment that includes not only nonself but self. In time, they undergo an adaptation within an individual time scale, as well as evolution in a species time scale. Thus, it is suggested that a superficial analogy could be misleading in mimicking biological systems; biological mimicking should not be done at a phenomenological level, but instead on a principle level.

Another reason for the complexity and intangibility of biological systems seems largely due to the feature of the material they comprise – namely, proteins. This would suggest that a constructive systems approach to biological mimicking systems can be not only an alternative to modeling and simulations but also a complementary tool supporting and guiding the modeling and simulation. The huge information available in post genome era allows a systems approach to biology, and this trend is accelerated for immunology as well. Next generation immunity-based systems may depend not only on a modeling/simulation approach, but also on a constructive approach that might bridge between the material and experimentally-based immunology and model/simulation-based informatics on bio-systems.

In summary, next generation immunity-based systems should focus on the following:

- A constructive systems approach to computational intelligence and artificial systems by assuming material similar to the real biological systems
- Extension and enhancement of models and simulations so that several operations are possible, involving genome data in the post-genome era, and targeting bioinformatics incorporating the immune system (such as personalized medicine involving the immune system).

This Chapter explores the first issue – that is, we consider next generation immunity-based systems by first revisiting conventional immunity-based systems (focusing on recognition capability), and next by extending them by

restricting antibodies (or peptides in general) as a base material for a constructive systems approach to immunity-based systems. Another Chapter in this volume explores the models without recognition – that is, all the agents mounting only effectors but without sensors (see Chap. 4). In this Chapter, each agent is assumed to be capable of recognizing the state of other agents.

This Chapter is organized as follows: Sect. 2 focuses on the preliminary problem of whether recognition is indeed needed, focusing on the specific task of abnormal state eradication on a simple network. Section 3 addresses the problem of networked recognition that involve action counterpart, hence agents can not only recognize but also be recognized. Section 4 further introduces adaptation by assuming agents can not only reproduce but also mutate in the receptor counterpart. Section 5 considers arrayed recognition, which is the very first step, even before networked recognition; however, it assumes specific recognition capability of antibody-antigen recognition.

2 Impact of Recognition

It is still controversial whether the immune system actually needs to discriminate ‘self’ and ‘nonself’ in order to eliminate *nonself* [20], however, elimination is actually done, and hence the double-sided property that elimination could be directed not only towards nonself, but also to self. Thus, the immune system is a double-edged sword.

This Section considers the impact of recognition in a simple model. To observe the impact, a simplified problem of network cleaning is considered. In information systems, the repairing units can repair others simply by copying their content, but could have spread contamination when the repairing units themselves are contaminated. We consider the possibility of cleaning up the network by mutual copying. However repair by copying in information systems is also a ‘double-edged sword’, and it needs to be identified when the network can really eradicate abnormal elements from the system.

The self-repairing network consists of units capable of repairing other connected units. We call the connected units as neighbor units based on the terminology of cellular automata (CA). Although mutual repair and other interactions involved may be done in an asynchronous manner, our model considers synchronous interactions for simplicity. Each unit tries to repair the units in its neighborhood, however whether it can really repair or not depends on several factors: the state of the repairing unit and the success rate of the repair.

In a mathematical formulation, the model consists of three elements ($\mathbf{U}, \mathbf{T}, \mathbf{R}$) where \mathbf{U} is a set of units, \mathbf{T} is a topology connecting the units, and \mathbf{R} is a set of rules of the interaction among units. In the simulations to come, a set of units is a finite set with N units, and the topology is restricted

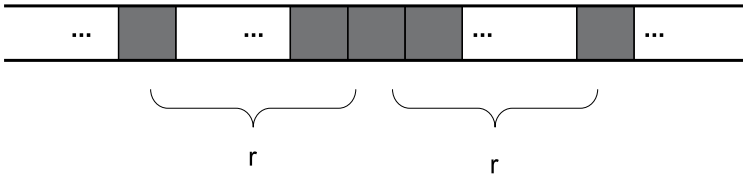


Fig. 1. One-dimensional lattice with the neighborhood radius r ; the next state of the cell will be determined by $2r + 1$ nodes in the neighborhood

to the one-dimensional lattice as shown in Fig. 1. The network structure could be an n -dimensional array, complete graph, random graph, or even a scale-free network. In our one- or two-dimensional lattice, each unit has S neighbors and the lattice with a boundary condition – in other words, the structure of the lattice is a ring with unit 1 adjacent to the unit N in the case of a one-dimensional lattice. Also, we restrict our discussion to cases where each unit has a binary state: normal (0), and abnormal (1).

2.1 An Impact of Recognition is a Double-Edged Sword

Our model also involves recognition of the states (normal or abnormal) of a target node before trying to repair it. For simplicity, frequency of recognition is controlled by a recognition rate γ . When recognition is undertaken (with a probability γ), successful recognition occurs with a recognition success rate γ_0 when performed by normal nodes, and γ_1 by abnormal nodes. If the target node is identified as ‘abnormal’, repair action take place. When recognition does not occur (with a probability $1 - \gamma$), the repair action takes place with the probability μ . Thus, if recognition is completely suppressed ($\gamma = 0$), this new model reverts to the original model. Figure 2 shows the procedure of recognition and repair.

Computer simulations are conducted in a one-dimensional array with a ring structure (periodic boundary condition). The parameters listed in Table 1 are fixed throughout the simulations. Other parameters: $\gamma, \gamma_1, \mu, \text{ and } \alpha_1$ are varied to observe the impact of recognition.

We are concerned with the problem: “Is recognition really necessary?” Moreover, if ‘yes’, then when and how should the recognition should be incorporated? In the following simulations, we pursue the problem of identifying an appropriate level of recognition (namely, γ) when the adverse effect of abnormal units (that is, γ_1 and α_1) is given.

When the rate of successful repair by abnormal nodes (that is, α_1) is given, what is the minimum level of recognition (namely, γ) required for abnormal node eradication? Figure 3 plots the minimum level of γ . As observed and already reported, we do not care about the level of repair and/or recognition when α_1 exceeds a threshold (0.4 in this simulation). However, when α_1 is less

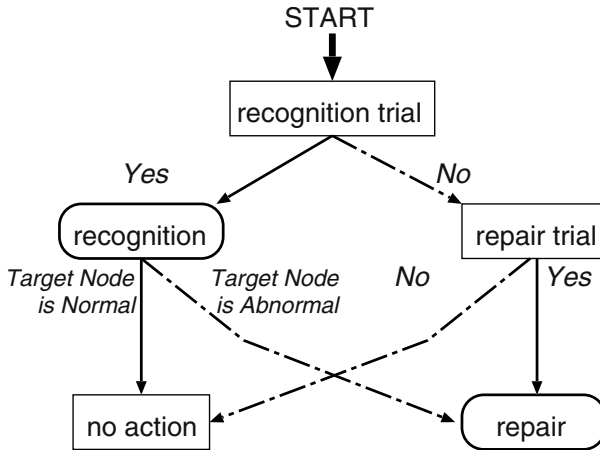


Fig. 2. Recognition carried out prior to repair

Table 1. Parameter list for the simulations

	Description	Value
N	number of nodes	500
$N_f(0)$	initial number of failure nodes	250
r	neighborhood radius	1
T	number of time steps for each trial	5000
N_T	average number of trials	10
α_0	repair success rate by normal nodes	1
γ_0	recognition success rate by normal nodes	1

than the threshold, recognition is needed (γ is positive) to eradicate abnormal nodes. Furthermore, the smaller the level of repair (μ), the smaller the level of recognition (γ) can be.

In this simulation (and with the specific model parameters as indicated), only repair by copying suffices for abnormal node eradication when the rate of successful repair by abnormal nodes exceeds some level. However, recognition before repair is required when the rate does not exceed this level.

3 Immunity-Based Systems: Evolved Recognitions

3.1 Definition of Immunity-Based Systems

Although recognition may not be needed under an optimistic situation in a simple network model, as in the previous Section, immunity-based systems (IMBS) [13] assume each agent mounts receptor counterpart. IBMS as a design

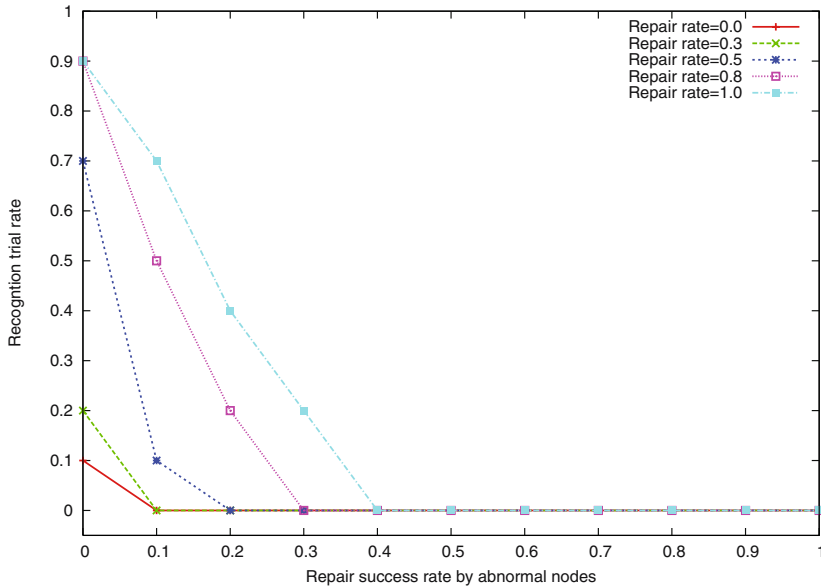


Fig. 3. Minimum level of γ to eradicate abnormal nodes when α_1 is given

paradigm has the following three properties:

1. a *self-maintenance system* with monitoring not only of the nonself but also of the self
2. a *distributed system* with autonomous components capable of mutual evaluation
3. an *adaptive system* with diversity and selection

In the following Sections, networked recognition focuses on the first two, while adaptive recognition involves the third one of these.

3.2 Networked Recognition

[17] proposed the immune network. In network theory, the immune system is not merely a ‘firewall’ but a network of antigen-antibody reactions. That is, when an antigen is administered, it stimulates the related cells and causes them to generate antibodies. However, the generated antibodies themselves are antigens to other cells, and consequently result in another antibody generation. The antigen-antibody reaction percolates like a chain reaction and hence requires a regulation mechanism. An analogy of this problem in engineering design is the ‘alarm problem’ of placing mutually activating and non-activating alarms whose sensitivity must be appropriately set to avoid false negative and false positive.

There is a variety among immune system models, even if we restrict ourselves to those by differential equations. If they were to be described by a single

equation with x_i : the number of recognizing (or recognized) sets (T -cells, B -cells, antibodies, and antigens) and a_{ij} : interactions between type i and type j (positive for stimulation and negative for suppression), the equation would be:

$$\frac{dx_i(t)}{dt} = F(\{x_i(t)\}, \{a_{ij}(s_i(t), s_j(t), aff_{ij}(t))\}) \tag{1}$$

where s_i denotes the state of the type i entity (for example, activated/inactivated, virgin/immune, and so on); and aff_{ij} the affinity between these two types. The dimension of x_i (the number of types) can vary, since a new type can be born, mutated from other types, or just injected in the case of antigens.

So far, this is not much different from the population dynamics of general ecological systems described by the Lotka-Volterra equation, for example. What makes this equation peculiar to the immune system is that interactions a_{ij} vary depending on the states of type i and type j entities, as well as the affinity between them. It is this affinity that models of the immune system devised by several techniques, such as the ‘shape-space’ model [23], where antigens and antibodies are expressed as points in the space, which allows the affinity between them to be measured as a distance between the points. Several spaces such as continuous and discrete ones are considered, hence several distances too (such as Euclidean and Hamming distance).

In such dynamical models, immunological concepts such as immune memory and tolerance are mapped to attractors of the dynamical systems. Within the context of problem solving, attractors of the system are mapped to solutions, thus the perturbed state (nonself) will be attracted to the solution (self), and hence nonself will be eliminated and self will be preserved. Positive and negative regulation will be interpreted as reinforcement and elimination.

Let us consider a credit assignment problem where high credit should be assigned to the self and low credit to nonself. Weighting the vote and propagating the information correctly identifies the abnormal agents. A continuous dynamic network is constructed by associating the time derivative of the state variable with the state variables of other agents connected by the evaluation chain. Further, considering not only the effect from evaluating agents, but also that from evaluated agents leads to the following dynamic network:

$$\frac{dr_i(t)}{dt} = \sum_j T_{ji}R_j + \sum_j T_{ij}R_j - 1/2 \sum_{j \in \{k: T_{ik} \neq 0\}} (T_{ij} + 1) \tag{2}$$

where $R_i(t) = \frac{1}{1 + \exp(-r_j(t))}$ and

$$T_{ij} = \begin{cases} -1 & \text{if evaluating agent } i \text{ is normal and evaluated agent } j \text{ is faulty} \\ 1 & \text{if both agents } i \text{ and } j \text{ are normal} \\ \pm 1 & \text{if evaluating agent } i \text{ itself is faulty} \\ 0 & \text{if there is no evaluation from agent } i \text{ to agent } j \end{cases} \tag{3}$$

In evaluating agents, agent j will stimulate (inhibit) agent i when $T_{ji} = 1(-1)$. We call this model the *black-and-white model*, meaning that the network tries to separate an abnormal agent clearly from a normal agent; namely, the *credibility* (which differs from the probabilistic concept of *reliability*) of an agent tends to be 1 (fully credible) or 0 (not credible), not an intermediate value. Moreover, we have proposed several variants of this dynamic network, such as the *skeptical model* and the *gray model* for different engineering needs. The results presented in this Chapter are generated only from the *black-and-white model*.

Figure 3 shows an example of the evaluation chain of mutual voting. The pattern associated with the evaluation arc shows a case when agents 4 and 5 are faulty. A positive arc from agent i to agent j indicates that agent i voted positively for agent j (in other words, considered ‘normal’), and a negative arc negatively (that is, considered ‘abnormal’). Formally, evaluation results are assumed to give the following pattern:

$$T_{ij} = \begin{cases} -1 & \text{if evaluating agent } i \text{ is normal and evaluated agent } j \text{ is faulty} \\ 1 & \text{if both agents } i \text{ and } j \text{ are normal} \\ \mp 1 & \text{if evaluating agent } i \text{ itself is faulty} \\ 0 & \text{if there is no evaluation from agent } i \text{ to agent } j \end{cases} \tag{4}$$

Simple voting at each agent does not work, since three agents (2, 3, and 5) are all evaluated as ‘faulty’ by two other agents, and hence cannot be ranked in terms of credibility. Since an abnormal agent may give faulty results, these votes should be weighted. Next, let us introduce a binary weight for each agent: 0 (inactive or abnormal) when the sum of votes for the agent is negative, and 1 (active or normal) when the sum of votes for the agent is zero or positive. Starting with all agents active, evaluating the weight would synchronously result in the sequence of credibility vector $(R_1 R_2 R_3 R_4 R_5)$, as shown on the right of Fig. 4.

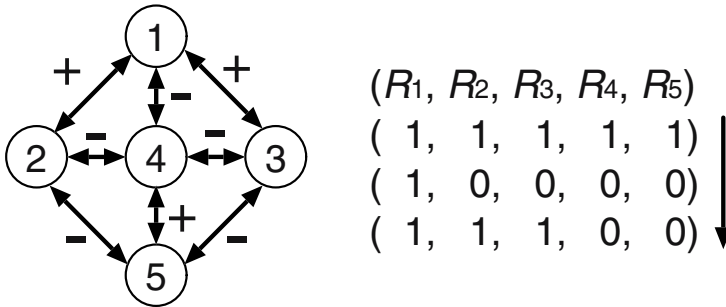


Fig. 4. An example evaluation chain of mutual voting (*left*), and the credibility vector sequence (*right*)

Example: Application to Automobile Engine Sensor Diagnosis [12]

A dynamic relational network can be built in roughly two steps:

1. *Line up candidates of relational arcs:* find causally related sensors by investigating correlation by checking indices such as coefficient of correlation.
2. *Narrow down the above candidates:* remove those arcs from sensor A to B if the test from sensor A to B generates false positives or false negatives.

A time series analysis is carried out for step 1 (using mutual correlation matrix), and/or for step 2 (prediction by the models of time series analysis). As reported below in the case of both the combustion control system of an automobile engine and for a particular fault in an air-flow sensor, a statistical analysis of up to step 1 for building the network suffices. However, time series analysis (with the VAR model) is used to determine the sign of an arc (evaluation from node i to node j) in online diagnosis.

In this Section, a case study with statistical analysis for building the relational network is reported. S_a indicates the data from sensor A . In step 1, arcs between A and B are added if $|\text{coefficient of correlation between } S_a \text{ and } S_b| \geq 0$. Figure 5 shows a network built when $\theta = 0.4$ and only step 1 in the algorithm is used. The network turned out to be complete. Signs are a snapshot of evaluation based on the sensor data. Gray level in the nodes indicates credibility. Dark nodes correspond to high credibility, while light nodes to low credibility (that is, evaluated as ‘faulty’) [14]. The signs of arcs in the network

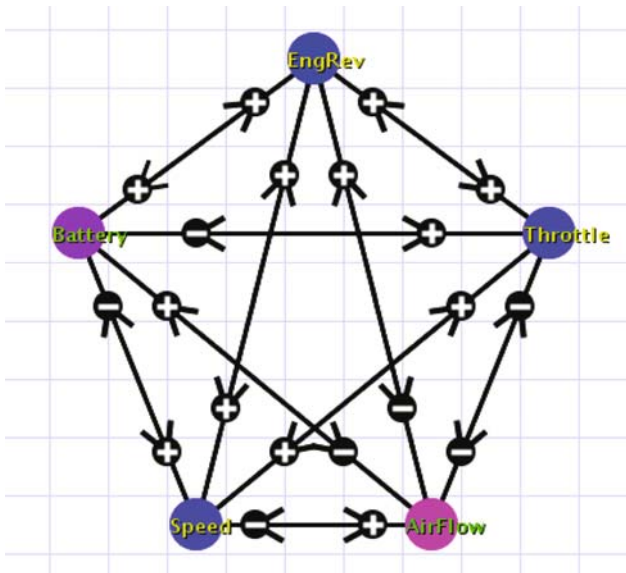


Fig. 5. A network with arcs added when $|\text{coefficient of correlation}| \geq 0.4$ in cruise phase (EngRev: engine revolutions; Battery: battery voltage) [14]

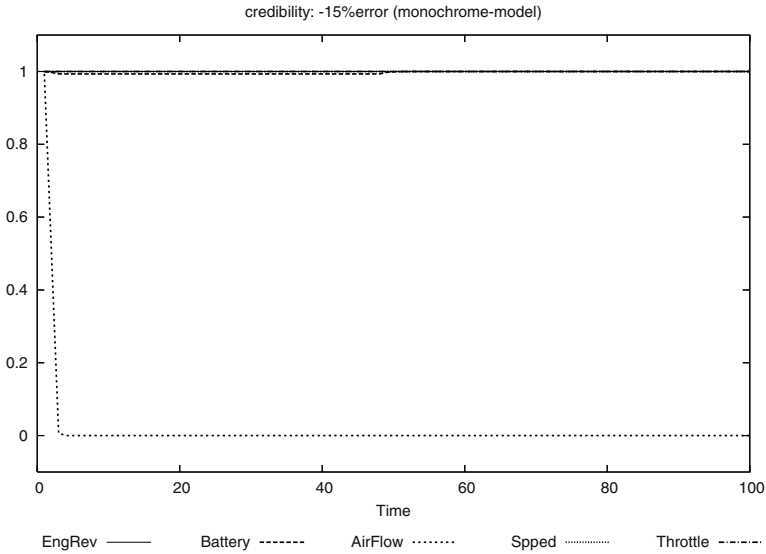


Fig. 6. Diagnosis by the evaluations calculated from the VAR model when the air flow sensor is faulty

change dynamically in online diagnosis; Fig. 5 shows only a snapshot of signs. The network structure does not change during the diagnosis.

It should be noted that the above calculation is done using only normal sensor data. Figure 6 shows the time evolution of credibility calculated by the time series analysis stated above. The dotted line shows the time evolution of sensor credibility. Only the credibility of the faulty sensor (Air Flow) becomes 0, hence the diagnosis is successful.

As heuristics for solving problems by the networked recognition, the following remarks apply:

- Signals from different agents should be related by signal processing models and statistical analysis [14] to map from the signals to evaluations (positive/negative sign of the network as in Fig. 5).
- Interactions among agents should be designed so that attractors of the entire network correspond to solutions to be obtained

Compared with the Bayesian Network [22], the above networked recognition is not able to obtain probabilities of events, however the problem solving mechanism can be directly embedded in the system where many agents are able to relate with each other. When applied to the signal processing domain, as in the above example, networked recognition is able to utilize the information embedded in the relations between the signals, as well as the information embedded in each signal itself – that is, both absolute and relative information in multiple signals can be involved. Networked recognition can be applied

not only to signal processing domain but also to other domains, such as data mining, and search engines, if distributed agents are involved and mutually related.

3.3 Adaptive Recognition

[4] speculated on clonal selection theory based on antibody production. An immune algorithm for a population of agents is proposed based on the clonal selection concept [16]. The most naive immune algorithm has the following three steps carried out in parallel by agents distributed over the system. In the algorithm, agents (corresponding to the immune cells) have not only recognition and communication capabilities, but also reproduction capability with possible mutation.

1. *Generation of diversity*: diverse agents with distinct specificity of the receptor and the effector are generated;
2. *Establishment of self-tolerance*: agents are adjusted to be insensitive to ‘known patterns’ (self) during the developmental phase;
3. *Memory of nonself*: agents are adjusted to be more sensitive to ‘unknown patterns’ (nonself) during the working phase.

Figure 7 shows a process which basically mimics the affinity maturation; affinity will increase by exploring diverse agents with slightly varied receptors. Diversity is generated by recombination of genetic counterpart, which is due to

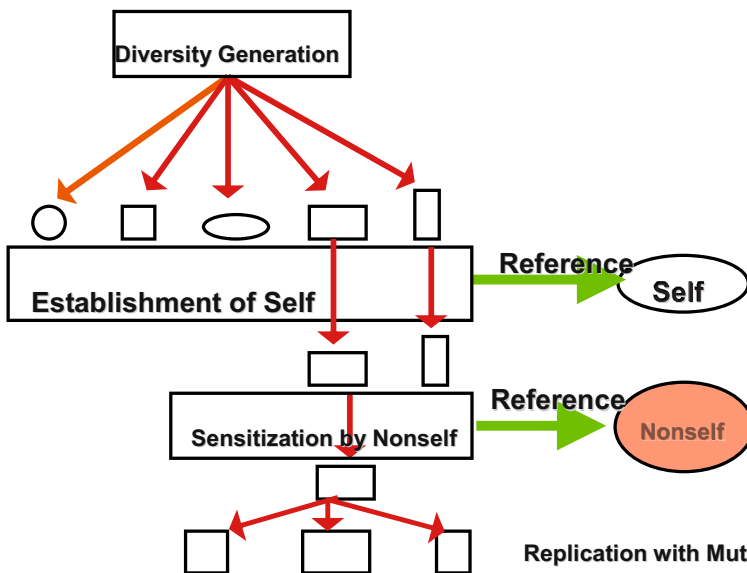


Fig. 7. Utilizing diversity for affinity maturation by agent filtering and agent sensitization [13]

the finding by [25]. In using diversity for exploring further possibility of affinity increase, slight variations of not only structure but also function (affinity) can contribute. For the immune system, the environment with which it must interact is not only the nonself from the outer world, but also the self from the internal world.

Immune algorithms are meant for specific problems where self-nonsel self discrimination and openness to the environment are critical. Further, the immune algorithm assumes ‘agents’ as a primitive to build immunity-based systems. In summary, the significance of the immune system used by the immune algorithm is:

- indirect information transfer from the environment by ‘selection’, as opposed to ‘instruction’;
- adaptive character driven by continuous diversity generation;
- involvement of self-reference as well as nonself-reference.

An outline of the immune algorithm is depicted in Fig. 7. The algorithm is described in a general context – it is for any adaptive system for self-nonsel self.

This action part formalized as an immune algorithm has been used for noise cancelation, where noise corresponds to the nonself and the control signal to the self. Since the signal is not labeled beforehand, agents must discriminate the self signal from the nonself one by the specific features of these signals. Further, the cancelation signal from agents must be discriminated for other agents. Although the noise cancelation can apply even to the unknown noise, it must deal with self-reactive agents (that try to cancel the control signal) as if auto-immune disease could happen to the immune system.

Example: Noise Neutralization by Agents

Agents with diverse receptors are first needed. As a set of gene data for initial agents, primitive ones such as shown in Fig. 8 can be used. Diversity may be provided by genetic operations such as recombination. In the simulation, however, ten different gene data with different base lengths but identical heights are used. Since genes will change by adaptation to the noise in the immune algorithm, the initial set of genes may be arbitrary as long as they have variations. However, primitive genes are required so they can approximate many shapes of disturbance signal. During adaptation in the memory of nonself step, genes mutate and higher affinity is attained.

To observe immunologic memory, a noise is first imposed, then the different noise imposed at 15,000 step. Finally, the first noise is again imposed at 30,000 step. Figure 9 shows the response (output from the system) to this noise imposition. It is known that the neutralizer more efficiently neutralizes the noise in the second encounter, if we compare the responses at the initial and second (after 30,000 step imposition of other disturbances) encounter. This

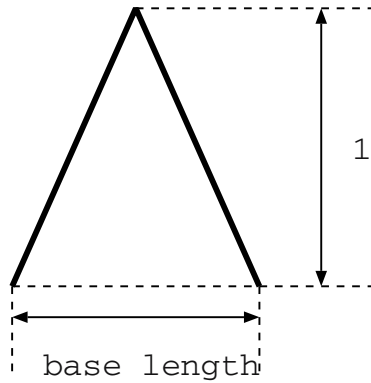


Fig. 8. Initial gene data; ten different base lengths are prepared initially [15]

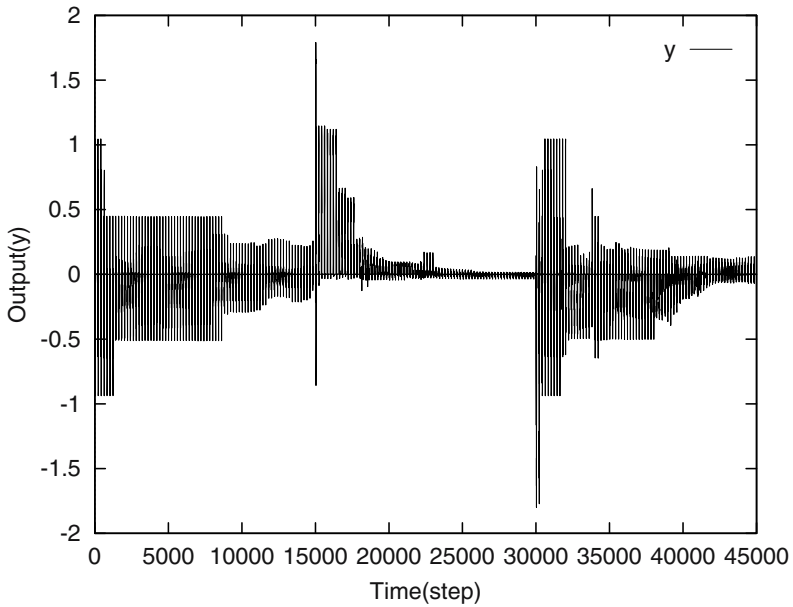


Fig. 9. Time evolution of error when first encounter with the disturbance of a type at step 0 and again at step 30,000, after imposition of a different disturbance from step 15,000 to 30,000 [9]

comes from the adaptation of the agent: memory attained by elongation of the lifespan in this case.

For observing the step of ‘Establishment of Self-Tolerance’ in the immune algorithm, a sine wave is imposed to the reference input, then agents cannot discriminate whether the signal is disturbance or the reference input.

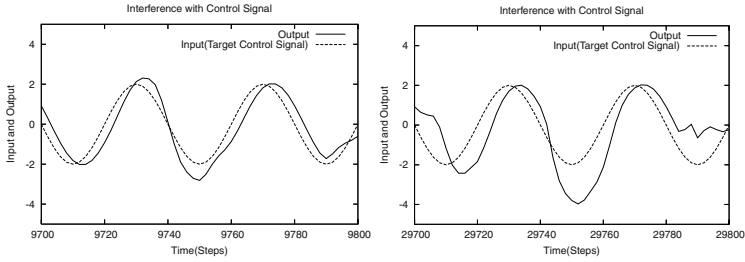


Fig. 10. Response from the system during 100 steps in the early phase (*left*) and the final phase (*right*) in a 30,000-step simulation when the self-reactive agents are not filtered

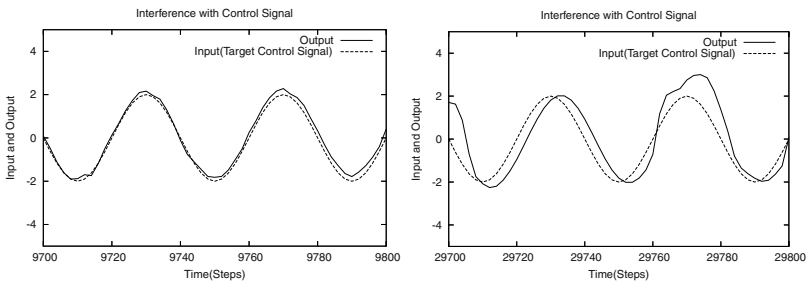


Fig. 11. Response from the system during 100 steps in the early phase (*left*) and the final phase (*right*) in a 30,000-step simulation when the self-reactive agents are filtered

In this and the next simulation, a training phase is added before the noise neutralization.

During the training phase, only the reference input is imposed without noise. In this simulation, agents are not filtered in the training phase. Figure 10 shows the response during 100 steps in the early phase (left: from 9,700 to 9,800 steps) and that in the final phase (right: from 29,700 to 29,800 steps) in a 30,000-step simulation in the noise neutralization phase after training. Noise is not well neutralized due to the self-reactive agents.

In another simulation, the self-reactive agents are removed at the training phase. After this training phase, the neutralizer is placed at the same environment as the previous simulation. Figure 11 shows responses both in the early and final 100, as in Fig. 10. In the final phase (righthand plot of Fig. 11), it is observed that the noise is well canceled while preserving the self (that is, the target signal). We also observe, however, that the self-reactive agents will appear after long time steps, due to the affinity increasing by mutation of existing agents. This would suggest that the self-reactive agents should not only be removed during the training phase, but also memorized at this

phase so that agents similar to the memorized one (hence self-reactive) will be removed whenever they appear by mutation.

In this simulation, one extreme of the IMBS is used – that is an adaptive system (open to the environment) where agents will evolve by adapting to the exogenous nonself. However, agents could form a network by communication and cooperation both in elimination of the disturbance and in memorizing the disturbance pattern.

For example, if the neutralizing signal can affect the other agents, then it would be close to the network model (networked recognition). Further, if the neutralizing signal from agents can be an error signal to other agents, then agents may be connected by signal similarly to Jerne's network [17].

As heuristics for solving problems by adaptive recognition, the following remarks apply:

- Signals (phenotype) should be mapped to gene data (genotype) by decomposing and expressing signals as a primitive signal such as a triangular (Fig. 8) or any other form (such as those found in wavelets), allowing coding of signals where genetic operations are possible;
- Reference to the self as well as nonself should be carefully designed, allowing for the possibility of either or both changing.

Compared with other population-based methods, adaptive recognition can handle not only changing nonself but changing self. This feature also leads to the risk of 'auto-immune disease'. As an application to intrusion detection, adaptive recognition can handle intrusion from both inside as well as outside, by preparing and diversifying profiles for legitimate users as well as illegitimate ones. That is, profiles of legitimate users within the firewall can be taken and processed not only to identify legitimate users (the self) but also to identify illegitimate masqueraders, by diversifying the profiles and even synthesizing the profiles of non-legitimate users (by mutating and recombining available profiles). Here, profiles are any signature that can be obtained by monitoring activities during login. This would provide the possibility of trade-off between internal masqueraders and external masqueraders, other than that between false positive and false negative. Intrusion detection (or equivalently legitimate user identification) becomes more important in the era of ubiquitous computing.

4 Antibody-Based Computing: Arrayed Recognition

[1] demonstrated that Hamiltonian circuits can be achieved by DNA-based computing. Many researchers established that not only DNA but also other macro molecules could have computational capability comparable to DNA. For example, protein-based computing had been proposed by [10] and extended by [2].

Antibody-based computing has a possibility of extension to an immunity-based problem solver that incorporates not only specific recognition of antibodies but adaptive nature supported by diversity generation, selection, and reinforcement of the selected antibodies.

4.1 Definition of Antibody-Based Computing

The immune system is capable of recognizing even artificially synthesized substances. Also, it can further classify substances into the self (those derived from the individual) and oneself. Among those bearing recognition capabilities, antibody is no doubt bearing important component and has been studied in great detail.

Similarly to the DNA-based computing, antibody-based computing utilizes affinity between macro molecules: antibodies. Since the computational capabilities that DNA-based computing could be inherited to antibody-based computing, we rather focused on the difference between them.

Affinity between antigens and antibodies can be measured and their intensities can be ordered (as formatted in an affinity matrix). That is, in contrast to $\mathbf{Matching}(DNA_i, DNA_j) = 1$ (matched) 0 (not matched), $\mathbf{Affinity}(Antigen_i, Antibody_j)$ can vary from 0 (no agglutination) to 1 (highest agglutination). This difference would suggest that antibody-based computing could be more general in expressing and solving problems. Also, error tolerance that could be implemented more directly than the DNA-based computing.

4.2 Solving a Combinatorial Problem: The Stable Marriage Problem

The stable marriage problem (SMP) [8] assumes n men and n women, with each member having preference lists of members of the opposite sex. A pair of a man M_i and a woman W_j is called a *blocking pair* if they are not pair in the current solution, but M_i prefers W_j to their current partner, and W_j prefers M_i to their current partner as well. A matching between men and women with no such blocking pair is called *stable*.

Let us consider the stable marriage problem by antibody-based computing. The stable marriage problem can be mapped to antigen-antibody reaction so that preference order of each person in SMP will be reflected in the affinity level between an antibody and an antigen. It should be remarked that agglutination process could be any agglutination (not necessarily between antibodies and antigens) if their affinity levels are measurable and ordered. After agglutininogen and agglutinin are adequately arranged, the solution of SMP will emerge by observing concentration of the agglutination.

Although obtaining a stable matching shows some computational power, it can be solved in $O(N^2)$ time, where N is the size of men (and women). A well-known algorithm exists for giving stable matching for man-oriented matching or woman-oriented one [7, 12]. By further assuming that the concentration observed at a cross-point can reflect the amount of antibodies imposed, the array is capable of obtaining any stable matching in the array from the man-oriented (man optimal and woman pessimal) matching to the woman-oriented (woman optimal and man pessimal) one. By regulating the quantities of all the antibodies $AbM_i (i = 1..n)$ (or equivalently antigens $AgW_j (j = 1..n)$, from a unit to α , the matching would become close to the man-oriented one. Similarly, increase of $AbW_i (i = 1..n)$ will bias the matching towards the one of woman-oriented one.

4.3 Mapping the Stable Marriage Problem to Antibody-Based Computing

Mapping a combinatorial problem to antibody-based computing can be done by composing antigen-antibody compounds corresponding to a problem entity. As for the stable marriage problem, the entity is an individual corresponding to a man or a woman. Antibodies and antigens for a compound corresponding to a particular individual will be determined by considering her(his) preference list over men(women).

Let us consider a scheme for synthesizing antigen-antibody compounds that realize mapping from given preference lists to the compounds. If the woman W_i prefers the man M_j to other men, the compound corresponding to W_i must contain antibody AbW_i and the compound corresponding to M_j contains antigen AgM_j that satisfies $\mathbf{Aff}(\mathbf{AbW}_i, \mathbf{AgM}_j)$ being highest among other $AgM_j (j = 1..n)$. If M_j is second in the preference list of W_i , then $\mathbf{Aff}(\mathbf{AbW}_i, \mathbf{AgM}_j)$ must be second highest, and so on. AgM_j must realize the order from women W_k other than W_i , hence the affinity $\mathbf{Aff}(\mathbf{AbW}_k, \mathbf{AgM}_j)$ must realize the order accordingly (if AgM_j alone cannot realize the order, then new antigen realizing the order must be added to the corresponding compound). Constraints for selecting antibodies and antigens for a compound corresponding to a person can be summed up as follows:

- $\mathbf{Aff}(\mathbf{AbW}_i, \mathbf{AgM}_j) > \mathbf{Aff}(\mathbf{AbW}_i, \mathbf{AgM}_k)$ if the woman W_i prefers M_j to M_k in her preference list; and
- $\mathbf{Aff}(\mathbf{AbM}_i, \mathbf{AgW}_j) > \mathbf{Aff}(\mathbf{AbM}_i, \mathbf{AgW}_k)$ if the man M_i prefers W_j to W_k in his preference list.

Let us next consider how to solve SMP with an array format. In the array shown in Table 2, row i and column j correspond to the compound for man i (namely, AbM_i and AgM_i), and that for woman j (that is, AbW_j and AgW_j). In other words, at the cross-point ij , two antigen-antibody reactions between AbM_i and AgW_j (reflecting man i 's preference), and between AbW_j and AgM_i (reflecting woman j 's preference) will take place.

Table 2. Arrayed compounds to solve the stable marriage problem: $M_i(W_i)$ stands for the compound for a man i (woman j), the symbol * at the ij cross-point indicates that M_i and W_i is selected as a stable pair due to a high affinity (each row and each column has only one pair [15])

Compounds	M_1	M_2	\dots	M_i	\dots	M_n
W_1						
W_2						
\vdots						
W_j				*		
\vdots						
W_n						

Table 3. Landsteiner’s ABO blood group system [15]

Blood Type	A	B	AB	O
Antigen (agglutinogen)	A	B	A,B	none
Antibody (agglutinin)	β	α	none	α, β

Under the assumption that the concentration observed at each cross-point is proportional to both $\mathbf{Aff}(\mathbf{AbM}_i, \mathbf{AgW}_j)$ and $\mathbf{Aff}(\mathbf{AbW}_j, \mathbf{AgM}_i)$, the array can find a stable matching by selecting one cross-point with highest concentration from each row and column. This matching is certainly stable one, for suppose otherwise there must be a blocking pair M_k and W_l such that $\mathbf{Aff}(\mathbf{AbM}_k, \mathbf{AgW}_l) > \mathbf{Aff}(\mathbf{AbM}_k, \mathbf{AgW}_p(\mathbf{M}_k))$ and $\mathbf{Aff}(\mathbf{AbW}_l, \mathbf{AgM}_k) > \mathbf{Aff}(\mathbf{AbW}_l, \mathbf{AgM}_p(\mathbf{W}_l))$, where $p(M_k)$ denotes a partner of M_k in the current matching. Then both concentration at the cross-point k_l is higher than those of $k_p(M_k)$, and those of $p(W_l)l$ reflecting the affinity level.

Example: A Trivial two-by-two Stable Marriage Problem

Landshteiner’s ABO blood group system [19] may be used as an example of antibody-based computing. His blood type system is based on antigens (as agglutinogen) on red blood cells and antibodies (as agglutinin) in the blood serum. Table 3 shows agglutinogen and agglutinin of each blood type. Affinity between antibody and antigen is shown in Table 4. Table 5 indicates the well-known incompatible transfusion among the blood type A, B, AB, and O.

Table 4. Affinity matrix: a circle indicates that the antibody-antigen reaction would occur if the antibodies in the column meet with antigens in the row

Antigen; Antibody	A	B
$\alpha(\text{anti-A})$	○	
$\beta(\text{anti-B})$		○

Table 5. Agglutination when the blood type in the column is transfused with the blood type to the row: a circle indicates that the blood type of the column when transfused to that of the column would agglutinate (a double circle indicates an agglutination higher than circles)

Blood Type	A	B	AB	O
A		○		○
B	○			○
AB	○	○		⊙
O				

Table 6. A trivial preference list for the two-by-two stable marriage problem

	M_1	M_2		W_1	W_2
W_1	1	2	M_1	1	2
W_2	2	1	M_2	2	1

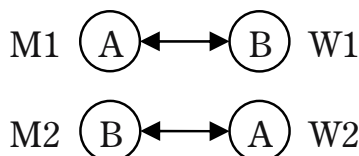
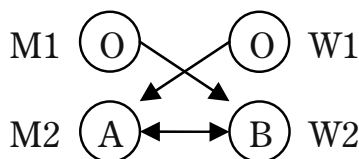
In this example, we map the relation the woman W_i (the man M_i) prefers the man M_j (the woman W_j) to other to the relation that if the blood of W_i (M_i) would be agglutinate when the blood of M_j (W_j) were transfused. That is, if the woman W_i prefers the man M_j most, the blood type should be so assigned that the type for W_i comprises of antibody AbW_i and antigen AgW_i ; and that for M_j of antibody AbM_j and antigen AgM_j and the affinity $\text{Aff}(\text{Ab}W_i, \text{Ag}M_j)$ are highest.

For the trivial case when the preference lists of men and women are as per Table 6, simple assignment would suffice: a man to type A and another man to type B; for the woman who prefers a man with type A to type B, and for another woman type A (Fig. 11). It should be noted that assignment to A for two men and to B for two women would not work, since the assignment does not reflect the preference of men and women.

In the nontrivial preference list shown in Table 7, one assignment would be type O to both M_1 and W_1 , type A to M_2 , and type B to W_2 (Fig. 13).

Table 7. A non-trivial preference list for the two-by-two stable marriage problem [23]

	M_1	M_2	W_1	W_2
W_1	2	1	M_1	2
W_2	2	1	M_2	2

**Fig. 12.** A blood type assignment reflecting the preference [15]**Fig. 13.** A blood type assignment reflecting the preference of Table 7 [23]

For other two preference lists (with a different graph topology than that of Figs. 12 and 13), it is not possible to map the blood type with the above correspondence, and other compounds should be synthesized for realizing the preference lists.

We have shown that antibodies, a macro molecule of the immune system, with specific recognition capability could be used for computation as DNA are used for DNA computing. We suggest a possibility of extending antibody-based computation to the solver, rather than showing its computational capability or universality. The application aims not at replacing current electronic computers, but rather at being a supportive tool for bioinformatics.

5 Toward a General Problem Solver: Immunity-Based Problem Solver

Problem solving by Means-Ends Analysis (MEA) [21] organizes a search in a dynamically constructed search space of problem-subproblem decomposition. It embodies and simulates human problem solving by the recognition-action cycle shown in Fig. 14, where solid arcs are recognitions and white arcs are actions. An important feature of MEA is that application of operators is not

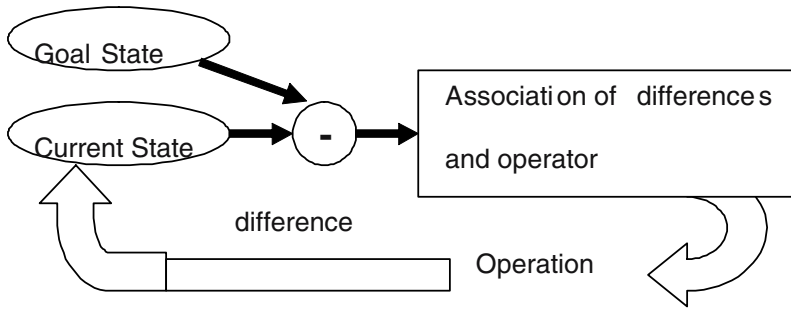


Fig. 14. Problem solving by Means-Ends Analysis (MEA) [15]

very rigid: if an operator selected in the heuristics part is not directly applicable to the current problem, the problem will be divided into subproblems. This flexibility allows a certain degree of freedom in identifying the heuristics, and further contributes to generality in the problems that MEA can handle. In fact, MEA had been implemented as General Problem Solver (GPS) that can deal with many well-known puzzles [6].

As an intermediate stepping stone from general problem solving by MEA to immunity-based problem solving, let us briefly investigate more general biological problem solving. It should be first emphasized that the following discussions and Fig. 15 is for bridging purposes between MEA and immunity-based problem solving, and hence may be rough and approximate. Here solid arcs are recognitions, and white arcs actions; recognition action cycles are iterated until there is no difference between the goal state and the current state. One difficulty is that a unit of biological system such as DNA, cells, individuals, and species do not constitute a usual hierarchical system understood in component-system relation found in most artificial systems. Another difficulty, hence making the biological problem solving remarkable, is that biological systems use concepts distinct from those used in artificial systems to realize robustness (robustness is a solution implemented and embedded in the system by a biological problem solving to challenges to system survival). Biological problem solving utilizes the variations for implementing robustness, although artificial problem solving considers the variations as disturbance and trying to prevent them from occurring and minimizing the effect. Thus, the 'difference' in Fig. 15 of biological problem solving can be quite different from that shown in Fig. 14.

The third difficulty that makes biological problem solving more complicated and entangled is that the given problem is a challenge to the survival of the problem solver (the biological system) itself. This self-referential aspect must be paid attention in capturing the immunity-based problem solving shown in Fig. 16.

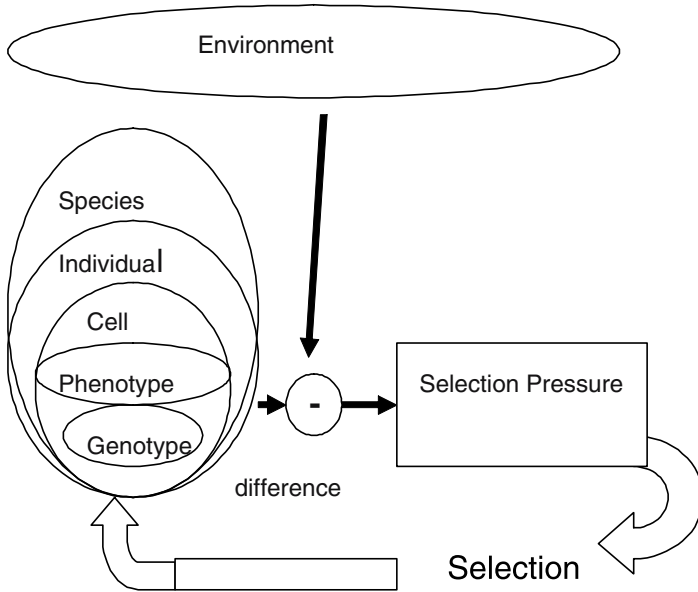


Fig. 15. Problem solving by Means-Ends Analysis (MEA)

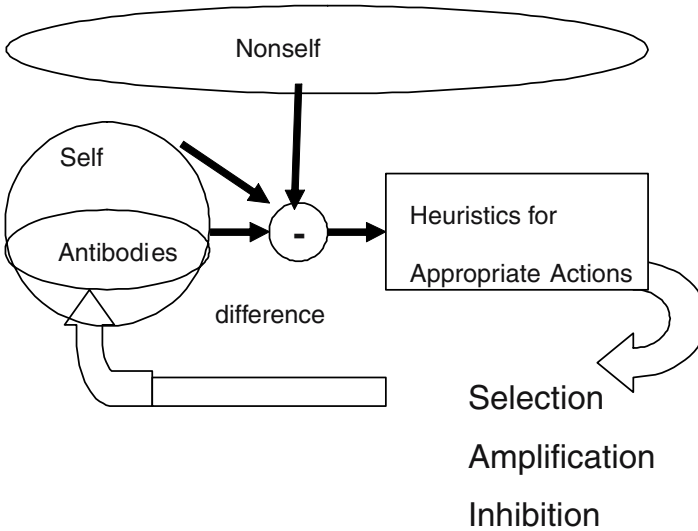


Fig. 16. The immune system as a problem solver: only antibodies are focused

Throughout Fig. 14 (MEA), Fig. 15 (biological problem solving) and Fig. 16, the framework for problem solving is to recognize differences and deploy actions based on these differences. However, actions are oriented toward the system itself for both biological and current immunity-based problem solv-

ing, hence the process is intrinsically adaptation (or in Fig. 15, evolution). Figure 16 and the following discussion focuses only on the immune system, involving antibodies, hence that of adaptive immunity.

In means-ends analysis on the one hand, the problem solving process constitutes an intrinsic part of the solution. That is, the *order* of operator applied is a critical part of the solution of a given puzzle. Thus, the problem is fixed throughout the problem solving; hence, the solver deals with a static problem.

On the other hand, in immunity-based problem solving, the problem itself undergoes changes, because the environment, including the nonself, is changing and the solver involving the self must change accordingly. Therefore, there is no complete solution and there will always be a gap between the current solution and the current problem. However, the current solution can be used for the next problem when the next problem (the change) also evolved from the current problem. Problem solving does not have a beginning and an end. The current solution is not good for the current environment because the latter is ever changing; therefore the gap between these two must be compensated for the next solution. However, the next solution is not built from scratch but rather from the current solution. The solution must always chase the environment, which is an online and dynamical adaptation to the dynamical environment. In immunity-based solving, the typical environmental change is either a challenge from the outside (for example, bacteria and viruses) or from inside (say, cancer). To deal with these challenges, the solver (a collection of agents) must prepare a diverse set for being selected by these problems (challenges) and the selected agents must be further increased. Since there is always a difference from the current solution and the current environment, there must be a diversification of agents.

6 Conclusion

This Chapter has explored the possibility of antibody-based computing that use antibodies or peptides in general. We first investigate several types of recognitions by revisiting immunity-based systems. The very primitive form of arrayed recognition is shown to have a computational capability comparable of DNA based computation by taking an example of a combinatorial problem: the Stable Marriage Problem. This would suggest that more sophisticated forms of recognition such as networked or selected recognition will have computational capabilities not only in a static context but in a more dynamic context as seen in the environments which the immune systems face. Thus, main feature of the antibody-based computing is that allows extension to a problem solver, since the immune system is a problem solver embedded in individuals, taking care of challenges to the individuals.

Acknowledgements

I am grateful to Prof. Fulcher, not only for giving me the opportunity for presenting this work, but also for his great assistance in editing and clarifying the chapter. I am also grateful to the anonymous reviewers, whose comments were quite helpful in improving the chapter. Mr. Sasajima, Mr. Mori, Mr. Oohashi, Mr. Sugawara, and Mr. Hiraizumi assisted with the numerical simulations. This work was supported in part by a Grant-in-Aid for Scientific Research (B) 16300067, 2004. This work was also partly supported by the 21st Century COE Program ‘Intelligent Human Sensing’ of the Ministry of Education, Culture, Sports, Science and Technology of Japan. I am also grateful to Mr. Abe, Mr. Hattori, and Mr. Toyofuku of Design Department No. 21, Electronics Engineering Division II, Vehicle Engineering Group, Toyota Motor Corporation, for incisive discussions on this project. Lastly, I am indebted to Dr. Watanabe who supported the project launch.

References

1. Adleman LM (2004) Molecular computation of solutions to combinatorial problems. *Science*, 266(11): 1021–1024.
2. Balan MS, Krithivasan K (2004) Parallel computation of simple arithmetic using peptide-antibody interactions. *Biosystems*, 76(1–3): 30–307.
3. Basu S, Gerchman Y, Collins CH, Arnold FH, Weiss R (2005) A synthetic multicellular system for programmed pattern formation. *Nature*, 434: 1130–1134.
4. Burnet FM (1957) A modification of Jerne’s theory of antibody production using the concept of clonal selection. *Australian J. Science*, 20: 67–69.
5. Elowitz MB, Leibler SA (2000) Synthetic oscillatory network of transcriptional regulators. *Nature*, 403: 335–338.
6. Ernst G, Newell A (1969) *GPS: A Case Study in Generality and Problem Solving*. Academic Press, New York, NY.
7. Gale D, Shapley L (1962) College admissions and the stability of marriage. *American Mathematical Monthly*, 69: 9–15.
8. Gusfield D, Irving RW (1989) *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA.
9. Hood L, Galas D (2003) The digital code of DNA. *Nature*, 421: 444–448.
10. Hug H, Schuler R (2001) Strategies for the development of a peptide computer. *Bioinformatics*, 17(4): 364–368.
11. International Human Genome Sequencing Consortium (2001) Initial sequencing and analysis of the human genome. *Nature*, 409: 860–921.
12. Irving RW, Leather P, Gusfield D (1987) An efficient algorithm for the optimal stable marriage. *J. ACM*, 34(3): 532–543.
13. Ishida Y (2004) *Immunity-Based Systems: a Design Perspective* Springer-Verlag, Berlin.
14. Ishida Y (2006) Designing an immunity-based sensor network for sensor-based diagnosis of automobile engines. In: Gabrys B, Howlett RJ, Jain LC (eds.) *Proc. Knowledge-Based Intelligent Informaiton and Engineering Systems Conf.*

- (*KES'2006*). Lecture Notes in Computer Science LNCS 4252. Springer-Verlag, Berlin: 146–153.
15. Ishida Y (2007) A constructive systems approach to understanding the immune system as a biological problem solver. In: *Proc. 1st IEEE Symp. Foundations of Computational Intelligence (FOCI'07)*, 1–5 April, Honolulu, Hawaii. IEEE/Omnipress, Madison, WI: 646–650 (CD-ROM).
 16. Ishida Y, Adachi N (1996) Active noise control by an immune algorithm: adaptation in immune system as an evolution. In: *Proc. 3rd IEEE Intl. Conf. Evolutionary Computation (ICEC'96)*, 20–22 May, Nagoya, Japan. IEEE Press, Piscataway, NJ: 150–153.
 17. Jerne NK (1973) The immune system. *Scientific American*, 229(1): 52–60.
 18. Kollmann M, Lovdok L, Bartholome K, Timmer J, Sourjik V (2005) Design principles of a bacterial signalling network. *Nature*, 438(24): 504–507.
 19. Landsteiner K (1900) Zur kenntnis der antifermentativen, lytischen und agglutinierenden wirkungen des blutserums und der lympe. *Zentralblatt Bakteriologie*, 27: 357–362.
 20. Langman RE, Cohn M (eds.) *Seminars in Immunology* (available online at: <http://www.ingentaconnect.com/content/ap/si/2000/00000012/00000003> – last accessed: April 2007).
 21. Newell A, Simon H (1972) *Human Problem Solving*. Prentice Hall, Englewood Cliffs, NJ.
 22. Pearl JJ (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (Revised 2nd Printing)*. Morgan Kaufmann, San Mateo, CA.
 23. Perelson AS, Oster GF (1979) Theoretical studies of clonal selection: minimal antibody repertoire size and reliability of self-non-self discrimination. *J. Theoretical Biology*, 81: 645–670.
 24. Sprinzak D, Elowitz MB (2005) Reconstruction of genetic circuits. *Nature*, 438(7067): 438–424.
 25. Tonegawa S (1983) Somatic generation of antibody diversity. *Nature*, 302: 575–581.
 26. Venter JC, Adams MD, Myers EW, et al. (2001) The sequence of the human genome. *Science*, 291: 1304–1351.
 27. Volterra V (1931) Variations and fluctuations of the number of individuals in animal species living together. In: Chapman RN (ed.) *Animal Ecology*. McGraw-Hill, New York, NY.

Resources

1 Key Books

Burnet FM (1959) *The Clonal Selection Theory of Immunity*. Cambridge University Press, London, UK.

Dawkins R (1976) *The Selfish Gene*. Oxford University Press, Oxford, UK.

Dennett DC (1995) *Darwin's Dangerous Idea: Evolution and the Meaning of Life*. Simon & Schuster, New York, NY.

Edelman GM (1992) *Bright Air Brilliant Fire: On the Matter of the Mind*. Basic Books, New York, NY.

Goldberg DE (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

Holland J (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Ishida Y (2004) *Immunity-Based Systems: A Design Perspective*. Springer-Verlag, Berlin.

Janeway CA, Travers P, Walport M, Shlomichik M (2002) *Immunobiology: The Immune System in Health and Disease*. (5th ed.). Garland Publishing, New York, NY.

Maturana H, Varela F (1980) *Autopoiesis and Cognition: The Realization of The Living*. D. Reidel, Dordrecht, Germany.

Paul WE, et al. (1999) *Fundamental Immunology (4th ed.)*. Lippincott-Raven, Philadelphia, PA.

Perelson AS (ed.) (1988) *Theoretical Immunology, Part I and II*. Addison Wesley, Reading MA.

Segel LA, Cohen IR (eds.) (2001) *Design Principles for the Immune System and Other Distributed Autonomous Systems*. Oxford University Press, New York, NY.

Tauber AI (1997) *The Immune Self*. Cambridge University Press, Cambridge, UK.

Varela FJ (1979) *Principles of Biological Autonomy*. Elsevier/North-Holland, New York, NY.

von Neumann JJ (1966) *Theory of Self-Reproducing Automata*. In: Burks AW (ed.) University of Illinois Press, Urbana, IL.

von Neumann JJ (1956) Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Shannon, C.E., McCarthy, J. (eds.) *Automata Studies*. Princeton University Press, Princeton, NJ: 43–98.

2 Key Survey/Review Articles

Farmer JD, Packard NH, Perelson AS (1986) The immune systems: adaptation and machine learning, *Physica D*, 22: 187–204.

Forrest S (1993) Genetic algorithm: principles of natural selection applied to computation. *Science*, 261: 872–878.

Jerne NK (1973) The immune system. *Scientific American*, 229(1): 52–60.

Jerne NK (1974) Clonal selection in a lymphocyte network. *Society General Physiologists Services*, 29: 39–48.

Jerne NK (1985) The generative grammar of the immune system. *The EMBO J.*, 4(4): 847–852.

Perelson AS, Weisbuch G (1997) Immunology for Physicists. *Review of Modern Physics*, 69: 1219–1267.

Tonegawa S (1985) The molecules of the immune system. *Scientific American*, 253(4): 122–131.

3 Organisations, Societies, Special Interest Groups

Santa Fe Institute

<http://www.santafe.edu/>

4 Research Groups

Artificial Immune Systems at University of York

<http://www-users.cs.york.ac.uk/jtimmis/>

Artificial Immune Systems at the Center for Information Assurance and Intelligent Security Systems Research Lab

<http://www.cs.memphis.edu/~dasgupta/>

Immunity-Based Systems at TUT Systems Sciences Lab

<http://www.sys.tutkie.tut.ac.jp/~ishida/en/index.html>

5 Key International Conferences/Workshops

FOCI 2007: The 1st IEEE Symposium on the Foundations of Computational Intelligence

<http://events.cs.bham.ac.uk/foci07/index.php>

ICARIS 2007: International Conference on Artificial Immune Systems

<http://lsin.unisantos.br/icaris2007/>

IMBS-KES 2007: Special Session on Immunity-Based Systems

<http://www.sys.tutkie.tut.ac.jp/~ishida/IMBS-KES07CFP.php.htm>

Knowledge-Based and Intelligent Engineering Systems Conf. (KES2007)/ XVIIth Italian Workshop on Neural Networks (WIRN2007)

<http://ra.crema.unimi.it/AIS2007/>

Index

- α -cuts, 606, 607
- λ -calculus, 9
- k -Nearest Neighbour rule, 99
- .NET, 420, 424

- abnormal agent, 164, 168, 169, 1097, 1098
- abnormal node, 1094, 1095
- abnormal node eradication, 1095
- absurd type, 359
- acceleration, swarm particle, 1033
- accuracy, classifier, 641, 646, 662, 663, 666, 672
- accuracy-complexity tradeoff, 641, 643, 670, 672–674
- action potential, 766, 767, 772, 776
- action, agent, 158
- activation function, 698, 804, 809
- activation, neuron, 804
- active sensor network, 488
- adaptation, 719, 1113
- adaptation gain, 725
- adaptation, neural system, 769, 770
- adaptive agent, 546
- adaptive agent, artificial, 545
- adaptive clustering algorithm, 490
- adaptive economic agent, 561
- adaptive immunity, 1113
- adaptive learning, 767
- adaptive linear element (ADALINE), 27
- adaptive market hypothesis, 967
- adaptive model, 570
- adaptive recognition, 1096, 1105

- adaptive resonance theory (ART), 46, 114, 693
- adaptive subspace self-organizing map (ASSOM), 731
- adaptive system, 851, 1105
- adaptive system, complex, 485
- adaptive-network fuzzy inference system (ANFIS), 45
- address-event communication, 781, 786
- adenine, 1066
- affect support agent, 201
- affective computing, 187, 206, 216
- affective embodied agent, 207, 210–212
- affinity, 1097, 1101, 1102, 1104, 1106, 1108
- affinity matrix, 1106, 1109
- affinity purification, 1081
- affinity separation, 1083
- affinity separation process, 1068
- agarose gel, 1069, 1080
- agent, 156, 351, 364, 381, 386, 397, 440, 453, 485, 1102, 1113
 - computerized, 967
 - evolving, 966
- agent action, 158
- agent behavior, 489
- agent boundary formation, 163
- agent cluster, 502
- agent communication, 486
- agent communication language, 42
- agent credibility, 1098, 1100
- agent diversification, 1113
- agent emotion, 216

- agent engineering, 563, 564, 567, 568, 570–572, 577
- agent forecasting rule, 415
- agent implementation language, 419, 431
- agent interaction, 409, 447, 450, 1100
- agent interaction protocol, 453
- agent network, 158, 426
- agent population, 156, 1101
- agent probe, 426
- agent reliability, 1098
- agent society, 412
- agent state, 158
- agent technology, 387, 403
- agent, abnormal, 164, 168, 169, 1097, 1098
- agent, adaptive, 546
- agent, adaptive economic, 561
- agent, affect support, 201
- agent, affective embodied, 207, 210–212
- agent, analysis, 696
- agent, animated, 209
- agent, artificial, 518, 578
- agent, artificial adaptive, 543, 545
- agent, autonomous, 387, 517, 579
- agent, beliefs-desires-intentions (BDI), 10, 23, 42
- agent, computational, 212
- agent, computing, 463
- agent, context-aware, 388
- agent, cooperative, 156
- agent, cooperative selfish, 175
- agent, disease, 1091
- agent, distributed, 1101
- agent, distributed analysis, 694
- agent, economic, 411, 541–544, 579
- agent, embodied, 206, 211
- agent, empathic, 210
- agent, financial, 563, 565
- agent, formula, 574
- agent, gambling, 577
- agent, global positioning system (GPS), 398, 400
- agent, GP-based, 555
- agent, heterogeneous, 158, 413
- agent, host, 1091
- agent, human, 518, 545, 578
- agent, independent, 156
- agent, intelligent, 21, 42, 386, 446, 543
- agent, interface, 201, 206
- agent, jellyfish, 426
- agent, k-means, 535
- agent, KNN, 535
- agent, long-horizon, 575
- agent, Lucasian economic, 541–544
- agent, management, 450, 463
- agent, mobile, 389, 395, 397, 398, 401, 402
- agent, multi-, 382
- agent, negotiation, 159
- agent, neighbor, 156, 158, 164, 170, 173
- agent, networked, 156
- agent, networked selfish, 158
- agent, normal, 164, 166, 168, 170, 172, 1098
- agent, offspring, 412
- agent, ontology-based, 381
- agent, profiling, 450, 463, 471
- agent, real estate (REA), 203, 207
- agent, repaired, 164, 168
- agent, scheduling, 450, 459
- agent, search, 381
- agent, self-reactive, 1102, 1104
- agent, selfish, 155, 156, 162, 163, 167, 168, 173, 175
- agent, software, 41, 386, 398, 409, 411, 578
- agent, task allocation, 156
- agent, teaching, 204
- agent, text-based, 210
- agent, zero intelligent (ZI), 568
- agent, ZI Plus (ZIP), 568
- agent-based artificial stock market, 563, 565–567
- agent-based computational economics (ACE), 517, 518, 549, 567, 568, 572, 574, 577, 579, 580
- agent-based economic model, 571
- agent-based macroeconomic model, 556
- agent-based macroeconomics, 573
- agent-based model (ABM), 409, 411, 418, 419, 421, 422, 426, 428, 431, 517, 549, 550, 560, 561, 568–572, 574–579
 - environment, 416, 431
- agent-based modeling (ABM)
 - framework, 421, 423, 424, 431

- agent-based modeling (ABM) platform, 410, 419, 428, 429, 431
- agent-based modeling (ABM) system, 420, 421
- agent-based paradigm, 419
- agent-based social sciences, 517
- agent-based stock market, 564
- agent-based system, 488
- agent-based tourist guidance system, 382
- agent-oriented programming (AOP), 41
- agglutination, 1106
- agglutinin, 1106, 1108
- agglutinin, 1106, 1108
- aging algorithm, 460
- AI-ECON artificial stock market, 563
- air operations officer (OPSO), 370, 372
- air-flow sensor, 1099
- aircraft, fighter, 370
- aircraft, strike, 370
- aircraft, threat, 371
- algae data set, 117, 119, 123, 143, 146
- algorithm
 - evolutionary (EA), 928
 - quantum, evolution, 963
- algorithm, aging, 460
- algorithm, backpropagation (BP), 13, 526, 781, 854, 1072
- algorithm, best-match, 453
- algorithm, branch-and-bound, 97
- algorithm, bucket-brigade, 543
- algorithm, clustering, 141
- algorithm, combinatorial, 352
- algorithm, evolutionary (EA), 23, 37, 47, 537, 834, 851, 862, 881, 883
- algorithm, expectation-maximization (EM), 121–123, 130, 141, 736, 747
- algorithm, first fit decreasing weight (FFD), 899, 914, 916, 917
- algorithm, fitness estimation PSO, 1051
- algorithm, fuzzy C-means (FCM), 617, 620
- algorithm, genetic (GA), 20, 23, 32, 37, 45, 47, 413, 415, 425, 542–544, 546, 550, 555, 556, 798, 803–806, 830, 831, 837, 838, 854, 856, 883, 902, 1036, 1041, 1049, 1052, 1071
- algorithm, gradient descent, 1053
- algorithm, greedy, 358, 899, 1032, 1057
- algorithm, harmonic k-means clustering, 141
- algorithm, HS, 735
- algorithm, hybrid grouping genetic (HGGA), 916
- algorithm, immune, 1101–1103
- algorithm, iterated greedy (IG), 916, 919
- algorithm, k-means clustering, 112, 141
- algorithm, memetic, 919
- algorithm, metaheuristic, 881
- algorithm, MTP, 916
- algorithm, nearest trajectory, 532
- algorithm, optimization, 1030, 1032, 1040, 1048, 1049
- algorithm, particle swarm optimization (PSO), 1029, 1048, 1054, 1056, 1059
- algorithm, RLS, 856, 858
- algorithm, scheduling, 440
- algorithm, storage-reduction, 535
- algorithmic mechanism design, distributed, 158
- aliasing, 1039, 1040
- allocation, resource, 157, 173, 443
- ambiguity problem, 311
- amplify operation, 1071
- analog circuit, 804
- analog hardware, 803
- analogue circuit evolution, 963
- analogue electronics, 768, 782
- analogue neural model, 782
- analogue neural system, 768
- analysis agent, 696
- analysis agent, distributed, 694
- analysis tree, 314
- analysis, mean field, 166
- analysis, means-ends (MEA), 1110, 1113
- analysis, pattern, 708
- analysis, statistical, 1100
- analysis, time series, 1099
- animal brain, 799
- animated agent, 209
- annealing, 1068
- annealing, genetic simulated (GSA), 906
- annealing, simulated (SA), 909
- annotated atomic formula (atom), 236

- annotated literal, 238
- annotated literal, vector, 243
- annotated logic, 233, 235
- annotated logic program with strong negation (ALPSN), 233, 241
- annotated logic program, paraconsistent, 238
- annotated logic programming, 233
- annotated logic, paraconsistent, 235, 300
- annotated logic, propositional paraconsistent, 235
- annotated sentence, 309
- annotation, 233, 235, 271–274
- annotation, bf-, 285–292, 294
- annotation, HPSG, 331
- annotation, LFG, 331
- annotation, paraconsistent vector, 300
- annotation, vector, 243, 271–274, 284, 293–295, 299
- annotator, human, 309
- ant colony optimization (ACO), 39
- ant system, 883
- antecedent fuzzy set, 644, 649
- anti-Hebbian learning, 777
- antibody, 159, 1091, 1093, 1096, 1101, 1106–1108, 1110, 1113
- antibody-based computing, 1106–1108, 1110, 1113
- antigen, 159, 1096, 1097, 1106–1108
- antigen-antibody compound, 1107
- antigen-antibody reaction, 1096, 1106, 1107
- anytime algorithm, 98
- application programming interface (API), 399
- application-specific integrated circuit (ASIC), 807
- applications
 - genetic programming, 958
- appraisal theory, 197
- approach, constructive systems, 1091
- approach, synthetic, 1091
- approach, top-down, 833
- approximate reasoning, 25
- approximation error, 597
- approximation, stochastic, 725
- approximator, universal, 521
- arc, 353, 453
- arc, evaluation, 1098
- architectural design, 365, 367
- architecture, artificial neural network (ANN), 851, 853
- architecture, brain, 764
- architecture, client-server (C-S), 440, 442, 444
- architecture, grid, 440
- architecture, neural, 719
- architecture, peer-to-peer (P2P), 440, 447, 462
- architecture, server-based, 447
- architecture, service-oriented (SOA), 446, 451
- architecture, subsumption, 49
- architecture-altering operator, 928, 949
- archive, digital, 388
- ARM processor, 786
- array, sensor, 488
- art
 - computer, 969
 - evolutionary, 970
- ART, fuzzy, 46
- artificial adaptive agent, 543, 545
- artificial agent, 518, 578
- artificial brain (A-Brain), 518, 797, 799, 800, 802, 804, 805, 808, 811, 812, 814, 816, 823, 825, 830, 832, 833, 835–838
- artificial immune system (AIS), 40
- artificial intelligence
 - human competitive, 962
- artificial intelligence (AI), 3, 4, 7, 9, 26, 311, 351, 381, 567, 690, 927, 1071
- artificial intelligence (AI), distributed, 41, 157
- artificial intelligence (AI), symbolic, 718
- artificial life (Alife), 17, 39, 156, 174, 409, 411, 417, 488, 546, 883
- artificial neural network
 - evolution, 957, 965
- artificial neural network (ANN), 4, 5, 9, 17, 20, 23, 26, 41, 45, 47, 111, 120, 403, 425, 518, 519, 532, 536, 579, 641, 643, 691, 692, 715, 733, 749, 772, 778, 781, 782, 798, 803, 831, 837, 851, 860, 872, 883, 1071
- artificial neural network (ANN)
 - architecture, 851, 853

- artificial neural network (ANN) bias, 521
- artificial neural network (ANN) classifier, 99
- artificial neural network (ANN) ensemble, 852, 858, 860, 870, 872
- artificial neural network (ANN) evolution, 803
- artificial neural network (ANN) group, 31
- artificial neural network (ANN) learning rule, 851
- artificial neural network (ANN) module, 797, 798, 804, 805, 807–809, 813, 826, 827, 830, 834, 836–838
- artificial neural network (ANN) weight, 23, 27, 28, 521, 851, 853
- artificial neural network (ANN), evolution, 859
- artificial neural network (ANN), modular, 859
- artificial neural network (ANN), pulsed, 27
- artificial neural network (ANN), recurrent, 851
- artificial neural network, evolutionary (EANN), 851, 855
- artificial neural network, memetic Pareto (MPANN), 870
- artificial neural system, 769, 781, 782
- artificial neuron, 801, 804, 811
- artificial problem solving, 1111
- artificial stock market, 413, 545, 562, 571
- artificial stock market, agent-based, 563, 565–567
- artificial stock market, AI-ECON, 563
- artificial stock market, SFI, 413, 545, 563, 564, 575
- artificial system, 18, 49, 767, 768, 1092, 1111
- ARTMAP, fuzzy, 47
- assertion, defeasible, 249, 262
- asset pricing model, 563
- association rule, 22
- associative memory (AM), 715, 724, 726, 782
- associative network, feedforward, 734
- associative search, 782
- assumption-based truth maintenance system (ATMS), 233
- asymmetric membership function, 662
- asynchronous inter-agent communication, 489
- ATIS corpus, 334
- atlas, digital, 801
- atom (annotated atomic formula), 236
- atomic formula, 235
- atomic formula, annotated (atom), 236
- attack, random, 491
- attack, targeted, 491
- attention, 193
- attraction point, 1032, 1033, 1038, 1046
- attraction, short-range, 1039
- attractor, 175, 561, 1097, 1100
- attribute, 366
- auction, 157
- auction scheme, 567
- auction, double (DA), 574, 578
- audio streaming, 690
- augmented genetic algorithm (GA), 550
- AURA system, 782
- Australian credit card data set, 856, 858, 864, 868, 871
- auto-associative neural network (AANN), 525
- auto-immune disease, 1102, 1105
- auto-regression, fuzzy multivariate, 48
- autoepistemic logic, 233
- automata, cellular (CA), 158
- automata, probabilistic cellular (PCA), 159
- automated reasoning, 319, 374
- automated theorem proving, 3, 8
- automatic modularization, 859
- automatic speech recognition (ASR), 3
- automatic statistical summary of elementary speech structures (ASSESS), 200
- automatically defined function (ADF), 948
- automobile engine, 1099
- automobile engine combustion control system, 1099
- automobile engine sensor diagnosis, 1099
- autonomous agent, 387, 517, 579

- autonomous nanotechnology swarm (ANTS), 37
- autonomous robot, 797, 836
- autoregressive (AR) model, 523
- autoregressive moving-average (ARMA) model, 520, 523
- available bandwidth, 690
- average, harmonic, 141
- averaging, 863, 865, 871
- Avida, 410
- axon, 770, 777, 780, 781

- backbone, grid, 468
- backgammon, 49, 964, 970
- backpropagation (BP) algorithm, 13, 27, 28, 39, 526, 699, 781, 854, 861, 1072
- backpropagation (BP), fuzzy, 46
- backtracking, 10, 16, 882, 919
- backward chaining, 10, 23
- backward chaining genetic programming, 975
- bacteria, 1113
- bagging, 44
- balance, exploration-exploitation, 1031
- balancing, load, 450, 463
- Baldi, 202
- bandwidth allocation, dynamic, 691
- bandwidth prediction, 690, 691
- bandwidth, available, 690
- bandwidth, consumption, 691
- bandwidth, network, 691
- base pair, DNA, 1066, 1080
- base sequence, DNA, 1077
- base, DNA, 1066
- base, Herbrand, 239
- basic emotions, 187, 202
- basis function, 121, 122, 129, 131, 133
- basis function, canonical, 354
- basis function, Gaussian, 130
- basis function, radial (RBF), 135
- basis vector, Gaussian, 136
- basis, canonical, 359, 361
- batch, 422, 424, 425, 429, 430
- Battiti's MI-based feature selection method (MIFS), 98
- Bayesian classifier, 11
- Bayesian discriminant, 96
- Bayesian discriminant feature selection (BDFS), 96
- Bayesian information criterion, 748, 751
- Bayesian learning rule, 575
- Bayesian network, 5, 10, 21, 23, 580, 1100
- Bayesian self-organizing map, 740
- Bayesian vector quantization (VQ), 726
- Bayesian, naïve, 44
- bead, magnetic, 1069, 1080, 1083
- before-after (bf) relation, 234, 284–288, 290, 293–295
- before/after, disjoint, 285
- before/after, f-included, 288
- before/after, immediate, 286
- before/after, included, 287
- before/after, joint, 286
- before/after, paraconsistent, 288
- before/after, s-included, 287
- behavior change, 212
- behavior change model, 213
- behavior control module, 814
- behavior, agent, 489
- behavior, niching, 1045
- behavioral finance, 577
- belief network, 5, 10, 23
- beliefs-desires-intentions (BDI) agent, 10, 23, 42
- beliefs-desires-intentions (BDI) model, 42
- benchmark, discrete mathematics and theoretical computer science (DIMACS), 894, 912, 913, 919
- benchmark, natural language processing (NLP), 314
- Beowulf cluster, 783, 979
- best position (pbest), 1031–1034, 1038, 1045
- best, global (gbest), 1030, 1031, 1033, 1036–1038, 1040, 1045, 1046, 1051
- best, local (lbest), 1030, 1033, 1034, 1036, 1037, 1051
- best-match algorithm, 453
- bf (before-after) EVALPSN, 234, 291, 294, 299
- bf-annotation, 285–292, 294
- bf-EVALP clause, 285–289, 291, 293, 294, 296, 298, 299
- bf-EVALP literal, 290, 294

- bf-EVALPSN clause, 285
- bf-EVALPSN process order safety verification system, 295
- bf-EVALPSN safety verification, 297
- bf-literal, 285, 293, 294
- bf-measure, 290
- Bhattacharryya divergence, 95
- bi-lattice, 236, 290, 292
- bias, artificial neural network (ANN), 521
- biased mutation, 664, 665
- bidirectional associative memory (BAM), 693
- bin packing heuristic, 914
- bin packing problem (BPP), 882, 887, 892, 894, 899, 905, 914, 917, 969
- binary tournament selection, 656, 661
- bio-engineering, 1091
- bio-soft computing, 1081
- biochemical process, 1066
- bioinformatics, 968, 1092, 1110
- biological brain, 763, 765
- biological inspiration, 1029
- biological mimicking, 1092
- biological neural network, 769
- biological neural system, 769, 780, 784, 786
- biological neuron, 26, 766, 767, 773, 775
- biological neuron IC (BIONIC), 784
- biological plausibility, 782, 1030, 1044
- biological problem solving, 1111
- biological system, 18, 157, 767–769, 779, 781, 788, 968, 1091, 1092, 1111
- biology, 175, 766
- biology, molecular, 1065
- bird flock, 1029
- bit string, chromosome, 804, 806, 809
- bit string, genetic algorithm (GA), 903
- bit-flip mutation, 657, 664
- black box model, 641
- black-and-white model, 1098
- black-box, 541
- Black-Scholes model, 548
- bloat, 982, 986
 - control
 - genetic programming, 987
 - genetic programming theory, 986
 - software, 48
- blocking pair, 1106, 1108
- blood group, 1108
- blood serum, 1108
- blood transfusion, 1108
- BLT, 423, 424
- Blue Brain, 783, 789, 790
- Blue Gene, 800
- Bluetooth, 398, 401
- book, code, 727
- Boolean logic, 23, 41, 766
- Boolean logic circuit, 1071
- boosting, 44
- boredom control circuit, 824
- boredom meter module, 824
- bound, greatest lower (GLB), 356, 358, 363
- bound, least upper (LUB), 358, 367
- bound, lower, 357, 360
- boundary formation, agent, 163
- bounded rationality, 517
- Box-Jenkins, 533
- Bragg diffraction, 745
- brain, 14, 20, 716, 769, 777, 789
- brain architect (BA), 797, 809, 811, 813, 814, 820, 824, 826, 829, 832, 834, 836
- brain architecture, 764
- brain building, 799, 800, 802, 806, 807, 814, 834, 835
- brain function, 764, 766, 769
- brain modeling, 769
- brain theory, 799
- brain, animal, 799
- brain, artificial (A-Brain), 518, 797, 799, 800, 802, 804, 805, 808, 811, 812, 814, 816, 823, 825, 830, 832, 833, 835–838
- brain, biological, 763, 765
- brain, human, 767, 768, 799–801
- brain, mammalian, 773, 836
- branch-and-bound algorithm, 97
- breadth-first search, 10, 16
- breast cancer data set, 868
- brewery pipeline network, 267
- brewery pipeline process, 284
- brewery pipeline valve control, 265
- bucket-brigade algorithm, 543
- building design ontology, 368
- building elevator, 1065, 1072

- building, high-rise, 1072
- building, multi-storey, 1065, 1073
- C, 410, 418, 421, 423, 807
- C measure, 730
- C++, 410, 418, 420, 421, 424, 426, 427, 430
- C, Handel-, 798, 803, 805, 830, 832
- C, Objective, 420
- C-language Inference Production System (CLIPS), 8, 24
- C#, 420, 424
- cache
 - data, 976
 - fitness, 977, 982
- calculus, λ -, 9
- calculus, predicate, 9
- calibration, fuzzy set, 598, 599, 620, 621
- camera language, 813
- camera, CCD, 813
- camera, CMU-CAM2, 813
- cancelation, noise, 1102
- cancer, 1113
- cancer classification, 101
- cancer data set, 101
- candidate solution, 883
- canon, 353, 354, 358, 359
- canonical basis, 359, 361
- canonical basis function, 354
- canonical formation rule, 351, 353, 356, 358, 361, 366, 374
- canonical graph, 356
- capacity, network, 691
- capital asset pricing model (CAPM), 574
- Cartesian genetic programming, 958
- cascade correlation, 29
- case-based reasoning (CBR), 21, 333
- Cauchy mutation, 863
- cell phone, 397
- cell, immune, 159
- cell, Voronoi, 727
- cellular automata (CA), 158, 411, 576, 1093
- cellular automata, probabilistic (PCA), 159, 164, 165, 168, 169
- cellular automaton evolution, 958, 963
- Celoxica, 797, 805, 806, 809, 814, 830, 834, 837
- central limit theorem, 727
- centralized scheduler, 440
- centroid, cluster, 531
- cerebral cortex, 721
- certainty grade, 647
- CFX, 419
- chaining, 459
- chaining mechanism, 462
- chaining operation, 456
- chaining policy, 453
- chaining, backward, 10, 23
- chaining, forward, 10, 23
- channel noise model, 728
- chaos, 7, 155
- chaotic dynamic system, 532
- chaotic system, 1071
- chaotic time series, 532
- character string, 1066, 1070
- charge-coupled device (CCD) camera, 813
- chart, process schedule, 269, 279
- chart, process time, 286–288
- checkers, 970
- checkpoint, 452
- checkpointing, 421, 426
- chemistry, computational, 968
- chess, 964, 970
- child, 885
- China Brain project, 836
- China test, 813
- Chisholm example, 258
- chromosome, 538, 539, 542, 543, 883, 904, 919
- chromosome bit string, 804, 806, 809
- chromosome, elite, 808
- chromosome, test, 808
- circuit, analogue evolution, 963
- circuit, analog, 804
- circuit, boredom control, 824
- circuit, digital, 804
- circuit, dominator, 821, 828
- circuit, Hamiltonian, 1105
- circuit, winner-takes-all (WTA), 820, 822, 823
- circular self-organizing map, 748
- class, 360
- class, pattern, 20
- Classdesc, 421, 426

- classification, 83, 112, 518, 529, 535, 536, 642, 644, 647, 648, 651, 662, 664, 715, 726, 833
 - genetic programming, 958
- classification boundary, 649, 653, 654
- classification data, 17
- classification error, 29, 641
- classification, cancer, 101
- classification, hierarchy, 386
- classification, ionosphere, 100
- classification, pattern, 10, 16, 26, 28, 642, 644, 647, 648, 651, 662, 664
- classification, sonar, 100
- classifier, 81, 543–545, 641, 860
- classifier accuracy, 641, 646, 662, 663, 666, 672
- classifier complexity, 662
- classifier ensemble, 44
- classifier error rate, 641
- classifier fusion, 44
- classifier interpretability, 672
- classifier selection, 44
- classifier system, genetic-fuzzy (GFCS), 571
- classifier, artificial neural network (ANN), 99
- classifier, Bayesian, 11
- classifier, fuzzy rule-based, 641, 644, 647, 648, 650, 652, 653, 655, 661–663, 665, 667, 669, 672, 674
- classifier, interval rule-based, 644, 653
- classifier, pattern, 43
- clause, bf-, 296
- clause, bf-EVALP, 285–289, 291, 293, 294, 298, 299
- clause, bf-EVALPSN, 285
- clause, EVALP, 271–273
- clause, EVALPSN, 270, 279, 282, 295
- clause, generalized Horn (GH), 239
- clause, Horn, 238
- cleaning, network, 1093
- client, 447
- client, iJADE FreeWalker, 397
- client-server (CS) architecture, 387, 395, 440, 442, 444
- client-therapist relationship, 212, 215
- clique, 912, 914
- clock tick, 805, 808, 824, 827–829, 831
- clonal selection theory, 1101
- closure, genetic programming, 938
- cluster, 748
- cluster centroid, 531
- cluster computing, 442, 443
- cluster diameter, 493
- cluster formation algorithm, dynamic, 490
- cluster map, 491, 493
- cluster, agent, 502
- cluster, data, 614, 617
- cluster, PC, 801, 802
- cluster-head, 489, 491
- cluster-information message, 493
- clustering, 112, 136, 614, 715, 726, 732, 749
- clustering algorithm, 141
- clustering algorithm, decentralized, 489
- clustering algorithm, decentralized adaptive, 490
- clustering algorithm, k-means, 112, 141
- clustering algorithm, non-hierarchical, 530
- clustering coefficient, 491
- clustering heuristic, 491
- clustering, decentralized dynamic, 502
- clustering, dynamic data, 489
- clustering, fuzzy, 598, 614, 619
- clustering, fuzzy C-means (FCM), 614, 615
- clustering, k-means, 863
- clustering, temporal gene expression, 747
- clustering-based rule generation, 646
- CMU-CAM2 camera, 813
- co-evolution, 572, 573, 975
- co-evolutionary learning, 859
- coalition, multi-agent, 502
- cobweb model, 544, 549, 552, 571, 573, 574
- code, firing-order, 766
- code, population, 766
- code, rank-order, 780, 782
- code, rate, 766
- codebook, 727, 743
- coding theory, hierarchical noise tolerant, 727
- coefficient of determination, 701
- coefficient, clustering, 491
- coefficient, correlation, 498

- coefficient, fuzzification , 614, 616
- cognition system, 716
- cognitive modeling, 370, 799
- cognitive science, 311
- coil, double helical, 1080
- coil, random, 1080
- collective intelligence, 36
- collective memory, 1032
- colored Petri Net (CPN), 42, 447, 453, 455
- combination method, 855, 863
- combination operation, 321
- combinatorial algorithm, 352
- combinatorial explosion, 17, 1071
- combinatorial optimization, 659, 1071
- combinatorial problem, 881, 1065, 1107, 1113
- combustion control system, automobile engine, 1099
- commerce, electronic (eCommerce), 578
- commodity computing, 444
- commodity price, 157
- common generalization, 357
- common specialization, 357
- communication, 1101
- communication protocol, 387, 440, 447
- communication volume, 494
- communication, address-event, 781, 786
- communication, agent, 486
- communication, asynchronous
 - inter-agent, 489
- communication, inter-agent, 489, 490, 502
- communication, inter-task, 440, 445, 446
- communication, swarm particle, 1030
- communications system,
 - packet-switching, 786
- communications, network-on-chip (NoC), 785, 786
- compact genetic algorithm (cGA), 808, 837
- comparison, pairwise, 598, 608, 634
- compartamental model, 776
- compatibility, 445
- compatibility grade, 650
- competitive learning, 113, 530, 693, 724
- compilation, just-in-time, 420, 431
- compiler, 18
- compiler optimizer, 420
- compiler, hardware, 803–805, 830
- compiler, silicon, 805
- complement, Watson-Crick, 1069
- complementarity, Watson-Crick, 1066, 1071
- complementary DNA (cDNA), 1068
- complementary DNA molecule, 1066
- complete lattice, 235, 245, 271–274
- complex adaptive system, 485
- complex dynamical system, 769
- complex system, 17, 155, 157, 174, 175
- complexity measure, 673
- complexity, classifier, 662
- complexity, computational, 487
- compound search, 98
- computation, evolutionary (EC), 852, 859
- computation, neural, 769, 789
- computational agent, 212
- computational chemistry, 968
- computational complexity, 87, 92, 487, 733, 774, 775
- computational efficiency, 82, 774, 854, 856, 1041
- computational fluid dynamics, 419
- computational grid, 440
- computational intelligence (CI), 4, 16, 17, 20, 42, 155, 158, 174, 517, 549, 570, 571, 579, 1092
- computational intelligence (CI) system,
 - hybrid, 43
- computational learning, 307
- computational load, 1030, 1041, 1046
- computational model, 233
- computational model of emotion, 197
- computational music analysis, 307
- computational overhead, 87
- computational power, 768
- computational science, 442, 444
- computer
 - parallel, 977
- computer art, 969
- computer engineering, 768
- computer game, 970
- computer memory, 1065, 1066
- computer network, 442, 444
- computer personality, 208
- computer program, 18, 546, 547

- evolution of, 928
- computer science (CS), 5
- computer virus, 158, 174
- computer, DNA, 1065
- computer, electronic, 763
- computer, emotionally intelligent, 216
- computer, mainframe, 441
- computer, parallel, 440, 1065
- computer, silicon-based, 1065, 1071
- computer, super-, 20, 439, 767, 768, 789, 800, 801, 979
- computer, von Neumann, 1071, 1084
- computer-aided design (CAD), 369
- computer-mediated therapy, 212
- computerized agent, 967
- computers are social actors (CASA) paradigm, 207
- computing agent, 463
- computing grid, 439, 441, 443, 444
- computing management service, 464, 466, 474
- computing node, 446, 447
- computing paradigm, 156
- computing resource, 441, 444, 447, 471
- computing resource, heterogeneous, 441
- computing, affective, 216
- computing, anti-body, 1113
- computing, antibody-based, 1106–1108, 1110
- computing, bio-soft, 1081
- computing, cluster, 442, 443
- computing, commodity, 444
- computing, distributed, 439, 441, 442
- computing, DNA, 40, 155, 1069, 1071, 1077, 1105, 1106, 1110, 1113
- computing, enterprise, 440
- computing, evolutionary (EC), 4, 31, 39, 883
- computing, granular, 4
- computing, grid, 155, 156, 174, 446, 691
- computing, high-performance (HPC), 443
- computing, immunity-based (IBC), 40, 155
- computing, membrane-based (MBC), 40
- computing, meta-, 442
- computing, molecular, 1065, 1066
- computing, Nature-inspired (NIC), 40
- computing, neuromorphic, 768, 782, 789
- computing, parallel, 41, 442
- computing, parasitic, 155, 156, 174
- computing, peer-to-peer (P2P), 440, 444
- computing, protein-based, 1105
- computing, quantum (QC), 41
- computing, scientific, 420, 421, 444
- computing, soft, 4, 17, 1071
- computing, ubiquitous, 1105
- computing, wet, 1065
- concept, 352, 364
- concept specification, 382
- concept type, 364
- concept type hierarchy, 351, 360
- conceptual entity, 597
- conceptual graph (CG), 351, 352, 357, 358, 363–366, 368–370, 372, 374
- conceptual graph (CG) consistency, 361
- conceptual graph (CG) programming language, 364
- conceptual graph (CG) unification, 362, 364
- conceptual graph (CG) validity, 361
- conceptual graph theory (CGT), 351, 354, 360
- conceptual structure, 352
- concurrency, 763
- conditional probability, 94
- conditional probability, fuzzy, 651
- confidence interval, 606
- confidence level, 625
- conflict resolution, 233
- conformity relation, 354
- conjunction, information, 364
- conjunction, knowledge, 351, 363–369, 372, 374
- connected unit, 1093
- connection, 467
- connection, excitatory, 722
- connection, inhibitory, 722
- connectionism, 4, 17
- connectivity, 471, 766–769, 778, 780
- connectivity netlist, 786
- consciousness, 764, 802
- consequent class, 645, 650, 651
- consistency, 95, 357
- consistency, conceptual graph (CG), 361

- constrained reasoning, 21
- constraint, 364, 370, 386
 - grammar-based genetic programming, 950, 951
- constraint logic programming, 364
- constraint satisfaction problem (CSP), 364
- constraint, problem, 1035, 1049
- constraint, sensibility, 353, 359
- constriction factor, 1034
- construct, 366
- constructive algorithm for training
 - cooperative neural-network ensembles (CNNE), 866, 868
- constructive systems approach, 1091
- consumption bandwidth, 691
- container, 424, 472
- container, grid, 450, 466
- container, microkernel grid, 463
- container, smartGRID, 450, 463
- container, smartGRID2, 463
- content management system, 751
- context-aware agent, 388
- context-aware mobile service, 388
- context-aware system, 388
- context-aware tourist guidance system, 382, 388
- context-free grammar (CFG), 312, 324, 328
- context-free grammar, probabilistic (PCFG), 328
- context-free grammar, stochastic, 331
- context-free rule, 307
- context-preserving crossover
 - theory, 984
- context-preserving crossover, genetic programming, 935
- context-sensitive grammar
 - genetic programming, 952
- continuous problem space, 1046
- control
 - robot
 - evolution, 963
- control state, valve, 276
- control, energy, 501
- control, process, 300
- control, process order, 300
- control, process release, 277
- control, process release safety, 279
- controlled diversification, 1035
- controlled mix, 268, 272, 275
- controlled separate, 268, 272, 275, 276
- controller
 - proportional integrative and derivative (PID), evolution, 963
- controller, fuzzy, 25
- convergence predictor, 503
- convergence, genetic algorithm (GA), 906
- convergence, network, 27, 29
- cooperative agent, 156
- cooperative selfish agent, 175
- coordinates, grid, 491
- copy rule, 355
- copying, mutual, 156, 159, 163, 1093
- copying, repair by, 159, 163, 1095
- Core War, 964
- core, fuzzy set, 610
- Cormas, 413, 428
- corpora, LFG-annotated, 314
- corpus, 307
- corpus fragment, 308
- corpus, parsed, 308
- corpus-based parsing system, 336
- corpus-based, probabilistic parsing, 307
- correlation coefficient, 92, 498
- correlation entropy, 496, 502
- correlation matrix memory (CMM), 782
- correlation matrix, mutual, 1099
- correlation-based feature selection (CFS), 92
- cortex, 764–766, 786
- cortex map, 724
- cortical map, 721
- cortical micro-architecture, 766
- Cosmo, 207
- cost function, 530, 549, 726–728, 737
- counterpropagation network, 693
- covariance matrix, 734
- cox model, penalized, 102
- credibility, agent, 1098, 1100
- credit assignment problem, 1097
- crossover, 898
 - context-preserving
 - theory, 984
 - genetic programming,
 - context-preserving, 935

- genetic programming, headless
 - chicken, 935
- genetic programming, one-point, 935
- genetic programming, size-fair, 935
- genetic programming, subtree, 934
- genetic programming, uniform, 935
- homologous
 - theory, 984
- linear genetic programming,
 - homologous, 955
- one-point
 - theory, 984
- size-fair
 - theory, 984
- crossover operator, 31, 32, 36, 47, 537, 550, 556, 656, 668, 806, 853, 871, 885, 887, 891, 896, 902, 906, 919, 928, 947
- crossover probability, 656
- crossover rate, 891
- crossover, cycle (CX), 888, 889, 903
- crossover, greedy partition (GPX), 897
- crossover, merging (MOX), 888, 890, 903
- crossover, merging independent sets (MIS), 904, 912, 914
- crossover, multi-point, 885
- crossover, one point, 656, 885, 903
- crossover, order (OX), 888, 889, 903
- crossover, partially matched (PMX), 888
- crossover, permutation order based (POP), 903, 912, 914, 916, 918
- crossover, two point, 885
- crossover, uniform, 656, 664
- crossover, uniform order based (UOBX), 903
- crowd behavior, 411
- crowding distance, 660
- cuisine ontology, 393, 394
- Culberson and Luo (CL) heuristics, 898, 902, 904, 919
- cultural theory of emotion, 187
- currency, 558, 562
- curve fitting, 28, 959
- customized tourist information, 403
- cut point, 885, 903, 912
- cybernetics, 4
- cycle crossover (CX), 888, 889, 903
- cycle, safety verification, 296
- cycle, strategy update, 167
- cytosine, 1066
- damage rate, 168, 169, 173
- Darwin IV robot, 802
- Darwin streaming server, 702
- Darwinian evolution, 802
- Darwinism, neural, 802
- data, 14
- data modeling, 959
- data analysis system, multi-agent, 694, 696
- data cache, 976
- data classification, 17
- data cluster, 614, 617
- data clustering, 83
- data compression, 518
- data distribution, 83
- data driven, 146
- data exchange format, 384
- data mining (DM), 3, 4, 16, 29, 386, 549, 674, 675, 690, 699, 801, 968, 974, 1101
- data mining (DM), fuzzy, 651, 674
- data point, 131
- data pre-processing, 16, 19–21, 25, 26, 32, 33, 81, 82, 700, 919
- data projection, 749
- data projection method, 732
- data projection, nonlinear, 732
- data reduction, 82, 85, 101
- data reduction, entropy-based, 85
- data reduction, representative entropy (REDR), 85, 88
- data set, algae, 117, 123, 143, 146
- data set, Australian credit card, 856, 858, 864, 868, 871
- data set, breast cancer, 868
- data set, cancer, 101
- data set, diabetes, 856, 858, 864, 868, 871
- data set, Falkenauer, 916, 919
- data set, glass, 868
- data set, heart disease, 856, 857, 868
- data set, high-dimensional, 112, 146
- data set, ionosphere, 100
- data set, iris, 750
- data set, letter recognition, 868

- data set, microarray gene expression, 102
- data set, reduced, 83
- data set, Scholl and Klein (SK), 914, 916, 919
- data set, sonar, 100
- data set, soybean, 868
- data set, testing, 857, 865
- data set, training, 534, 855, 857, 864
- data set, validation, 855, 857
- data set, wine, 136
- data space, 121, 135
- data structure, 352
- data visualization, 83, 443, 715, 732, 736, 737, 749
- data, DNA sequence, 1091
- data, gene, 1102
- data, genetic, 1091
- data, genome, 1092
- data, meta-, 383, 388
- data-driven fuzzy membership function estimation, 598
- data-driven triangular fuzzy set, 612
- data-oriented parsing (DOP), 307, 336
- database, 443
- database management system (DBMS), 22
- DCOM, 424
- de-fuzzification, output, 25
- deadlock, 159, 175
- decentralized adaptive clustering algorithm, 490
- decentralized clustering algorithm, 489
- decentralized dynamic clustering, 502
- decentralized multi-agent algorithm, 501
- decision module, 814
- decision rule, 541–543, 546, 561, 562, 574
- decision support, 749
- decision tree (DT), 6, 9, 16, 91, 99, 532, 580, 860
- decision tree (DT) pruning, 17
- decoder, greedy, 887, 891, 920
- decomposition, task, 446
- deduction system, 360
- default logic, 233
- default reasoning, 233
- defeasible assertion, 249, 262
- defeasible deontic detachment (DDD⁺), 263
- defeasible deontic logic, 244, 257
- defeasible deontic logic, Nute's, 247, 256
- defeasible deontic reasoning, 275, 299
- defeasible deontic reasoning, EVALPSN, 266, 274
- defeasible deontic theory, 258, 262
- defeasible derivation principle, 249
- defeasible factual detachment (DSD⁺), 258, 262
- defeasible logic, 233, 244, 249, 254
- defeasible reasoning, 233, 243, 244, 247, 254
- defeasible rule, 248, 249, 254–256, 262
- defeasible theory, 248, 254–257, 262
- defeater, 248, 254
- defense domain, 370
- definite clause grammar, 324
- degree distribution, 490
- degree distribution, vertex, 490
- degree, fuzzy membership, 605
- delta learning, 693
- delta learning rule, 27
- delta learning rule, generalized, 27
- deme, 978
- denaturation, 1068, 1080, 1083
- denaturation temperature gradient PCR (DTG-PCR), 1068, 1080, 1081, 1083
- dendrite, 770, 777
- dendritic tree, 771, 773, 776, 784
- denial-of-service, distributed (DDOS), 158
- density estimation, 83
- density histogram, 745
- density matching model, 738
- density-based method, 85
- deontic detachment, defeasible (DDD⁺), 263
- deontic formula, 247
- deontic inheritance, 257
- deontic logic, 244
- deontic logic, defeasible, 244, 257
- deontic logic, Nute's defeasible, 247, 256
- deontic notion, 234, 247
- deontic operator, 256, 262
- deontic reasoning, defeasible, 275, 299

- deontic reasoning, EVALPSN defeasible, 266, 274
- deontic theory, defeasible, 258, 262
- deoxynucleotide-triphosphate (dNTP), 1068
- deoxyribonucleic acid (DNA), 40, 1066
- depth-1 subtree, 307, 312
- depth-first search, 10, 16
- derivation length, 313
- derivation tree, 319, 323
- derivation, language, 320
- derivation, most probable, 329
- derivation, music, 320
- derivation, physics, 320
- derivation, shortest, 323, 329
- derivational problem solving, 320
- Desatur, 913
- descent, gradient, 141
- description logic, 373, 386
- description, task/service (TSD), 451
- descriptor, 597–599
- design, architectural, 365, 367
- design, distributed algorithmic mechanism, 158
- design, engineering, 1096
- design, mechanism, 158
- detachment, defeasible deontic (DDD⁺), 263
- detachment, defeasible factual (DSD⁺), 258, 262
- detect operation, 1071
- detection, intrusion, 1105
- detection, network intrusion, 157
- detector, frequency, 829
- deterministic parsing, 307
- deterministic spatial prisoner's dilemma (SPD), 167
- deterministic Turing Machine (DTM), 1084
- device, handheld, 402, 403
- diabetes data set, 856, 858, 864, 868, 871
- diagram, Hasse, 290
- dialog processing, 308
- difference of Gaussians function, 114
- differential emotion scale (DES), 200
- differential equation, 1096
- differential evolution, 5
- digital archive, 388
- digital atlas, 801
- digital circuit, 764, 804
- digital electronics, 782
- digital hardware, 803
- digital organism, 410
- digital spike communications model, 782
- digital technology, 767
- dilemma, prisoner's (PD), 159
- dimensionality reduction, 28, 525, 715, 732, 749
- Dirac delta function, 772
- direct load management, 501
- directed acyclic graph (DAG), 977
- Directory, Google, 390, 402
- Directory, Open, 390, 402
- disambiguation, 312
- disambiguation, syntactic, 311
- Discipulus, 968
- discourse structure, 311
- discourse, universe of, 598, 603, 605, 610, 625
- discrete event scheduler, 422, 424, 425
- discrete mathematics and theoretical computer science (DIMACS) benchmark, 894, 912, 913, 919
- discretization, fuzzy, 642, 654, 661
- discretization, interval, 643, 654, 655
- discriminant, 11
- discriminant analysis, 719
- discriminant capability, 94
- discriminant, Bayesian, 96
- discrimination, self-nonsel, 1102
- disease agent, 1091
- disease, auto-immune, 1102, 1105
- disjoint before/after, 285
- disjunctive subset, 354
- dislex, 731
- dissimilarity, 732
- distance measure, 114
- distance, Euclidean, 16, 122, 129, 141, 530, 615, 630, 631, 732, 749, 1097
- distance, Hamming, 1097
- distance, Kullback-Leibler information, 738
- distance, Mahalanobis, 95
- distance, Tchebyshev, 616
- distributed agent, 1101

- distributed algorithmic mechanism design, 158
- distributed analysis agent, 694
- distributed artificial intelligence (AI), 41, 157
- distributed computing, 439, 441, 442
- distributed denial-of-service (DDOS), 158
- distributed grid, 690
- distributed heterogeneous grid, 690
- distributed media grid environment, 696
- distributed resources, 691
- distributed system, 387
- divergence, 732
- divergence, Bhattacharryya, 95
- divergence, Kullback-Liebler, 87, 95
- diverse and accurate ensemble learning algorithm (DIVACE), 870, 872
- diversification, agent, 1113
- diversification, controlled, 1035
- diversity, 860, 892, 979, 1101, 1102, 1106
- diversity, population, 906
- divide-and-conquer, 16, 821, 852, 859, 872
- dividend, stock, 413
- DNA base, 1066
- DNA base pair, 1066, 1080
- DNA base sequence, 1077
- DNA computer, 1065
- DNA computing, 40, 155, 1069, 1071, 1077, 1105, 1106, 1110, 1113
- DNA double-strand helix, 1066
- DNA duplex, 1068
- DNA molecule, 1065, 1069, 1070, 1080, 1082
- DNA molecule, complementary, 1066
- DNA molecule, double stranded, 1069
- DNA polymerase, 1066
- DNA process, 1065
- DNA reaction, 1065
- DNA sequence, 1066, 1068, 1069, 1078, 1080
- DNA sequence data, 1091
- DNA single-strand helix, 1066
- DNA strand, 1065, 1069, 1080, 1082–1084
- DNA template, 1068
- DNA, complementary (cDNA), 1068
- DNA, single-strand (ssDNA), 1068, 1081, 1083
- document mining, 726
- domain, 354, 358, 359, 364, 369, 382, 390
- domain expert, 8, 18, 382
- domain information, 367
- domain knowledge, 351, 359, 363, 370, 388, 597, 598, 624
- domain ontology, 382
- domain ontology model, 403
- domain semantics, 359
- domain, tourist, 397
- Domany-Kinzel (DK) model, 165
- dominator circuit, 821, 828
- DOP model, tree-based, 314
- DOP+, 313, 336
- DOP, ML, 315
- DOP, supervised, 334
- DOP, UML, 335
- DOP, unsupervised, 308, 333, 336
- double auction (DA), 564, 568, 574, 578
- double helical coil, 1080
- double-auction (DA) market, 568
- double-strand DNA molecule, 1069
- double-strand helix, DNA, 1066
- draughts, 970
- DSD-proof, 257
- duplex, DNA, 1068
- dynamic bandwidth allocation, 691
- dynamic cluster formation, 488
- dynamic cluster formation algorithm, 490
- dynamic clustering, decentralized, 502
- dynamic data clustering, 489
- dynamic fitness function, 975
- dynamic hierarchy, 489
- dynamic network, 488, 1097
- dynamic offset range, 491
- dynamic optimization problem, 1043
- dynamic problem, 1040, 1046
- dynamic subset selection, 974, 975
- dynamic swarm size, 1041
- dynamical system, 1097
- dynamics, molecular (MD), 410
- dynamics, multi-agent, 489, 503
- dynamics, non-linear, 565
- dynamics, population, 1097
- dynamics, replicator, 167

- eager scheduling, 457
- Echo, 410
- EcoLab, 420–423, 426, 429, 430
- ecological system, 1097
- econometrics, 519, 548, 552, 565, 566, 573, 575
- economic agent, 411, 541, 543, 579
- economic agent, adaptive, 561
- economic agent, Lucasian, 541–544
- economic model, agent-based, 571
- economic modeling, 540, 966
- economic system, 157
- economics, 158, 175, 517, 540, 570, 573, 576, 580, 966
- economics, macro-, 541, 544, 549
- economics, neo-classical, 576
- economics, Santa Fe Institute (SFI), 544
- economy, market, 157
- economy, planned, 157
- edge detection, 766
- effector, 42, 811
- efficiency, computational, 1041
- efficient market hypothesis (EMH), 548, 565, 566, 573, 967
- eigenvalue, 608
- eigenvector, 608, 733
- election operator, 550, 556
- electricity market complex adaptive system (EMCAS), 568
- electricity power grid, 441–443
- electricity supply, 568
- electroencephalogram (EEG), 198
- electromyography (EMG), 199
- electronic commerce (eCommerce), 578
- electronic computer, 763
- electronic document, 750
- electronic engineering, 768
- electronic system level (ESL), 807
- electronics, analogue, 768, 782
- electronics, digital, 782
- electrophoresis, gel, 1080, 1083
- elevator dispatch problem, 1080, 1082
- elevator group, 1065
- elevator group control system, 1072
- elevator group management, 1065, 1072
- elevator group supervisory control system (EGSCS), 1072
- elevator scheduling, 1065
- elevator scheduling, optimal, 1074
- elevator, building, 1065, 1072
- eLibrary, 689
- elite chromosome, 808
- elitism, 661
- elitist EMO algorithm, 659, 661
- elongation, 1068
- Elvis robot, 960
- email, spam, 158, 174
- embedded model, 91
- embedded processor, 786
- embodied agent, 206, 211
- embryonic hardware, 804
- emergence, 409, 486
- emergent property, 573
- EMO algorithm, elitist, 659, 661
- EMO algorithm, memetic, 663
- EMO-based genetics-based machine learning (GBML), 663
- EMO-based rule selection, 663
- emotion, agent, 216
- emotion, computational model of, 197
- emotion, cultural theory of, 187
- emotion, expression of, 191
- emotion, human, 208
- emotion, model of, 197
- emotion, neurological model of, 189
- emotion, simulated, 207, 208
- emotion, simulation of, 216
- emotion, synthetic, 207, 208
- emotional competence framework, 196
- emotional intelligence, 185, 196, 206
- emotional Stroop test, 193
- emotional voice recognition, 197
- emotionally intelligent computer, 197, 216
- emotions, 187
- emotions, basic, 187, 202
- emotions, higher cognitive, 188
- emotions, intentional, 190
- emotions, negative, 205
- emotions, nonintentional, 190
- emotions, secondary, 190
- emotions, tertiary, 190
- emotive speech, 202
- emotive text, 200
- empathic agent, 210
- empathy, 202
- encoder/decoder problem, 527

- energy control, distributed, 501
- energy landscape, 27
- energy management, distributed, 501
- engine monitoring and control, 965
- engine, automobile, 1099
- engine, search, 97, 1101
- engineering design, 1096
- engineering, agent, 563, 564, 567, 568, 570–572, 577
- engineering, bio-, 1091
- engineering, computer, 768
- engineering, electronic, 768
- engineering, evolutionary (EE), 826, 828, 829, 832, 836
- engineering, knowledge (KE), 351
- engineering, neural systems, 768, 783
- engineering, software (SE), 7, 18
- enhanced learning, 1072
- ensemble, artificial neural network (ANN), 852, 858, 860, 870, 872
- ensemble, classifier, 44
- enterprise computing, 440
- entity, logical, 267
- entity, physical, 267
- entropy function, 617
- entropy, correlation, 496, 502
- entropy, fuzzy set, 617
- entropy, Kolmogorov–Sinai, 495
- entropy, metric, 495
- entropy, Rényi, 495, 502
- entropy, representative (RE), 85
- entropy-based data reduction method, 85
- environment, 446, 719
- environment, agent-based modeling (ABM), 416, 431
- environment, open, 439, 440, 444
- environment, runtime, 450, 463
- enzyme, 1068, 1080
- enzyme function, 1066
- enzyme reaction, 1066, 1067
- epistemic negation, 236, 244, 246, 262, 272–274, 289
- EPNet, 853, 856, 859
- epoch, 19, 27, 854, 866, 868
- equal piles problem, 895, 897
- equation, differential, 1096
- equation, Kolmogorov, 160
- equation, position update, 1034
- equation, swarm update, 1046
- equilibrium, Nash, 155–157, 173–175
- equilibrium, Pareto-inferior, 554
- equilibrium, Pareto-superior, 554, 555
- era, post-genome, 1091
- eradication, abnormal node, 1095
- Erben fitness function, 912
- error landscape, 27
- error tolerant vector quantization (VQ), 726
- error, classification, 29
- error, least squares, 856
- error, mean squared (MSE), 16, 701, 733
- error, prediction, 701
- error, quantization, 737
- error, squared, 492
- error, vector quantization (VQE), 83
- error-correction learning, 719
- Essen Associative Code (ESAC), 317
- Essen Folksong Collection (EFC), 312, 316, 330
- estimation, fitness, 1041, 1043, 1050, 1052–1054, 1056
- estimation, fuzzy set membership, 620
- ethics, 203
- ethology, 799
- Euclidean distance, 16, 122, 129, 141, 493, 530, 615, 630, 631, 732, 749, 1097
- EVALP clause, 271–273
- EVALPSN clause, 270, 279, 282, 295
- EVALPSN defeasible deontic reasoning, 266, 274
- EVALPSN pipeline control safety verification, 266
- EVALPSN safety verification, 277, 279, 284
- EVALPSN safety verification framework, 266, 267, 284
- EVALPSN stable model, 265
- EVALPSN, bf (before-after), 234, 291, 294, 299
- evaluation arc, 1098
- evaluation, fitness, 1033, 1034, 1036, 1041, 1043, 1044, 1049, 1052–1054, 1056
- evaluation, rule, 383
- evolution, 883, 1113

- computer program, 928
- grammatical, 952
- recurrent transition network, 957
- evolution finite state automata (FSA), 957
- evolution strategy, 883
- evolution, artificial neural network (ANN), 803, 859
- evolution, co-, 572, 573, 975
- evolution, Darwinian, 802
- evolution, differential, 5
- evolution, generic, 830, 832, 837
- evolution, multi-module, 834
- evolution, multi-test, 828–830
- evolutionarily stable strategy (ESS), 174
- evolutionary algorithm (EA), 23, 37, 47, 537, 540, 568, 834, 851, 862, 881, 883, 928
- evolutionary algorithm (EA), multi-objective, 870
- evolutionary algorithm (EA), speciated, 859
- evolutionary algorithm parallel (EA), 977
- evolutionary algorithm, hybrid (HEA), 881, 913
- evolutionary algorithm, memetic, 881
- evolutionary art, 970
- evolutionary artificial neural network (EANN), 851, 855
- evolutionary computation (EC), 518, 519, 536, 579, 804, 852, 859, 872, 927
- evolutionary computation winter, 32
- evolutionary computing (EC), 4, 31, 39, 883
- evolutionary engineering (EE), 826, 828, 829, 832, 836
- evolutionary ensembles with NCL (EENCL), 862–865
- evolutionary game theory, 573
- evolutionary multiobjective (EMO)
 - fuzzy rule selection, 644, 664–666, 669, 670, 672
- evolutionary multiobjective design, 661, 663
- evolutionary multiobjective fuzzy data mining (DM), 674
- evolutionary multiobjective optimization (EMO), 643, 655, 658
- evolutionary music, 970
- evolutionary process, 766
- evolutionary programming (EP), 32, 36, 536, 540, 853, 883
- evolutionary quantum computer programming, 964
- evolutionary search
 - models, 983
- evolutionary strategy (ES), 32, 536, 538, 1072
- evolubility, 829, 832
- evolvable hardware (EHW), 34, 803, 826, 883
- evolving agent, 966
- examination timetabling, 894, 917
- example, Chisholm, 258
- exchange rate, 12, 544, 558, 560–562, 572, 967
- excitatory connection, 722
- exclusive-or (XOR), 27
- exemplar, 322
- exemplar, training, 26, 27, 29
- exemplary problem solution, 322
- exhaustive search, 10, 16, 881
- expectation operator, 553
- expectation-maximization (EM) algorithm, 121–123, 130, 141, 315, 335, 736, 747
- expert, 121, 603, 605, 607, 608, 620, 630
- expert knowledge, 363, 370
- expert opinion, 598
- expert system (ES), 18, 21, 22, 543
- expert system (ES) shell, 24
- expert system (ES), fuzzy, 45, 47
- expert, domain, 8, 382
- expert, human, 859
- experts, mixture of, 121, 129, 131, 141, 146
- experts, product of, 126, 130, 131, 146
- experts, sum of, 130
- explanation capability, 649
- explanation-based learning, 319
- exploitation, 1031, 1032
- exploitation, social, 1033
- exploration, personal, 1033
- exploration, serial, 1037, 1038

- exploration-exploitation balance, 1031
- explorer particle, 1044
- expression, 191
- expression, facial, 192
- extended vector annotated logic
 - program with strong negation (EVALPSN), 234, 244, 245, 256, 258, 262–265, 284
- extensible markup language (XML), 373, 383, 384, 388
- extension, 1068
- extension, feature structure (FS), 366
- external join, 356
- extraction, feature, 28

- f-included before/after, 288
- facial action coding system (FACS), 192, 199, 202
- facial action parameter (FAP), 192
- facial definition parameter (FDP), 192
- facial electromyography (EMG), 199
- facial expression, 192, 199
- factual detachment, defeasible (DSD⁺), 258, 262
- failure rate, 168–170, 173
- Falkenauer data set, 916, 919
- false negative, 1096, 1099
- false positive, 1096, 1099
- family, fuzzy set, 621
- fan-in, 765
- fan-out, 765
- fascicle processor, 786
- fault tolerance, 445, 448, 450, 457, 463, 715, 763, 764, 767, 769
- feature, 81
- feature dependence, 92
- feature evaluation, 91
- feature extraction, 28, 81, 525, 719
- feature relevance, 92
- feature selection, 81, 90, 101
- feature selection criteria, 91
- feature selection method, MI-raw based (MI-raw-FS), 98
- feature selection method, mutual information-based (MIFS), 98
- feature selection method, quadratic MI-based (QMIFS), 98
- feature selection, Bayesian discriminant (BDFS), 96
- feature space, 112, 740–742
- feature structure (FS), 366
- feature structure (FS) extension, 366
- feature structure (FS) inheritance, 355
- feature structure (FS) intension, 366
- feature structure (FS) type, 355
- feature subset goodness, 94
- features structure (FS) theory, 365
- feedback network, 693
- feedforward associative network, 734
- feedforward network, 693
- feedforward neural network, 522, 526, 719
- field programmable gate array (FPGA), 34, 40, 783, 785, 789, 797, 803, 806–809, 830, 831, 837
- fighter aircraft, 370
- file exchange, multimedia, 444
- filter method, 83, 92
- filter, low pass, 1031
- filtering, noise, 17, 21, 28
- finance, behavioral, 577
- finance, quantitative, 518
- financial agent, 563, 565
- financial market, 561, 577
- financial time series, 519, 531, 532, 548, 552, 563
- financial time series prediction, 966
- finite state automata (FSA) evolution, 957
- finite state automata (FSA), 540
- finite state machine (FSM), 34, 540
- firewall, 1096, 1105
- firing rate, 773, 778, 779
- firing-order code, 766
- first fit decreasing weight (FFD)
 - algorithm, 899, 914, 916, 917
- first-order logic (FOL), 9, 22, 360, 373
- fish shoal, 1029
- fitness, 884, 1032
 - fitness case, genetic programming, 942
 - multi-objective, 942
 - genetic programming, 966
- fitness cache, 977, 982
- fitness case, genetic programming, 942
- fitness estimation, 1041, 1043, 1050, 1052–1054, 1056
- fitness estimation PSO algorithm, 1051

- fitness evaluation, 854, 857, 1033, 1034, 1036, 1041, 1043, 1044, 1049, 1052–1054, 1056
- fitness function, 32, 536, 557, 732, 803, 805, 806, 808, 811, 814, 828, 829, 833, 834, 837, 838, 884, 891, 906, 918, 919, 940, 1047, 1049
 - dynamic, 975
 - genetic programming, 940, 944
- fitness function, Erben, 912
- fitness reliability, 1041, 1042, 1056
- fitness sharing, 859, 862, 982
- fitness-proportionate selection, 934
- fixed-length DNA algorithm, 1082
- fixpoint semantics, 240
- flat graph, 912
- floating search, 98
- floating-point operations per second (FLOPs), 767
- flock of birds, 1029
- Fluent, 419
- food source, swarm, 1031
- forecasting, 117, 518, 524
- forecasting rule, agent, 415
- forecasting rule, market, 413, 415
- forecasting, time series, 28, 535
- foreign exchange rate, 12
- forgetting term, 720, 723
- formal language, 42
- formal logic, 9
- formal safety verification, 234, 266, 299, 300
- formalism, non-monotonic, 244
- format, media content, 699
- formation rule, canonical, 351, 353, 356, 358, 361, 366, 374
- formula agent, 574
- formula, deontic, 247
- Fortran, 418, 423
- forward chaining, 10, 23
- forward message, 492
- Fourier transform, 745
- Fourier transform, quantum
 - evolution, 964
- frame, 9, 22
- frame system, 1071
- framework, 597
- framework, agent-based modeling (ABM), 421, 423, 424, 431
- framework, EVALPSN safety
 - verification, 284
- framework, genetic simulated annealing framework (GSA), 919
- framework, Java, 399
- framework, Jena, 399
- framework, message passing/routing, 462
- framework, order based, 919
- framework, resource management, 462
- Framsticks, 410
- Free Software Foundation (FSF), 417
- frequency assignment problem, 896, 899
- frequency detector, 829
- frequency-based search, 381
- full method, genetic programming, 931
- function approximation, 647, 651, 675
- function optimization, 549
- function optimization problem, 1044
- function, activation, 804
- function, basis, 121, 122, 129, 131, 133
- function, cost, 726–728, 737
- function, enzyme, 1066
- function, fuzzy membership, 47
- function, hyperbolic tangent, 521
- function, kernel, 740, 742, 774
- function, mapping, 740
- function, mexican hat, 724
- function, minimization, 1054
- function, neighbourhood, 725, 727, 728, 731, 737, 740, 742–744
- function, objective, 32, 162, 536, 614, 731, 742, 884, 893, 904, 919, 1043
- function, projection, 733
- function, Rastrigin, 1048, 1052, 1054
- function, Schwefel, 1049, 1054
- function, sigmoid, 521, 723, 804
- function, squashing, 804
- function, Tauber-Wiener, 521
- fusion, classifier, 44
- fuzzification coefficient, 614, 616
- fuzzification, input, 25
- fuzzy associative memory (FAM), 45
- fuzzy backpropagation (BP), 46
- fuzzy C-means (FCM) clustering, 614, 615
- fuzzy C-means (FCM) clustering
 - algorithm, 617, 620
- fuzzy clustering, 598, 614, 619

- fuzzy cognitive map (FCM), 23
- fuzzy conditional probability, 651
- fuzzy controller, 25
- fuzzy data mining (DM), 651, 674
- fuzzy discretization, 642, 654, 661
- fuzzy expert system (ES), 45, 47
- fuzzy feature evaluation index (FFEI), 97
- fuzzy GBML algorithm, 667
- fuzzy inference system (FIS), 25, 45
- fuzzy inference system, adaptive-network (ANFIS), 45
- fuzzy integral, 44
- fuzzy logic, 18, 23, 24, 45, 519, 536, 571, 580, 642, 1072
- fuzzy mapping, 624, 634
- fuzzy membership degree, 605
- fuzzy membership function, 47, 597, 601, 608
- fuzzy membership function estimation, 597
- fuzzy membership function estimation, data-driven, 598
- fuzzy membership function estimation, horizontal, 605
- fuzzy membership function estimation, pairwise, 605, 607
- fuzzy membership function estimation, priority, 607
- fuzzy membership function estimation, vertical, 605, 606
- fuzzy membership function, linear, 613
- fuzzy membership function, piecewise, 623
- fuzzy membership function, unimodal, 610
- fuzzy membership grade, 606
- fuzzy min-max neural network, 47
- fuzzy model, 47
- fuzzy multivariate auto-regression (MAR), 48
- fuzzy neuron, 628
- fuzzy partition, 661
- fuzzy reasoning, 370, 644, 647, 648
- fuzzy relation, 629, 634
- fuzzy rule, 9, 22, 641–644, 646–655, 662–670, 672–674
- fuzzy rule base, 47
- fuzzy rule extraction, 644, 650
- fuzzy rule selection, 663
- fuzzy rule set, 648
- fuzzy rule-based classifier, 641, 644, 647, 648, 650, 652, 653, 655, 661–663, 665, 667, 669, 672, 674
- fuzzy rule-based system, 627, 634, 641, 643, 648, 663, 673–675
- fuzzy search, 403
- fuzzy set, 25, 597–599, 605, 606, 616, 624, 625, 630
- fuzzy set calibration, 598, 599, 620, 621
- fuzzy set core, 610
- fuzzy set entropy, 617
- fuzzy set family, 621
- fuzzy set formation, problem-oriented, 599
- fuzzy set membership, 25
- fuzzy set membership degree, 617
- fuzzy set membership estimation, 620
- fuzzy set membership estimation, user-based, 620
- fuzzy set notation, 599
- fuzzy set operator, 25
- fuzzy set perception, 599, 629, 634
- fuzzy set semantics, 598, 599, 621
- fuzzy set, data-driven triangular, 612
- fuzzy set, Gaussian, 621, 623
- fuzzy set, higher-order, 620
- fuzzy set, hypercube, 47
- fuzzy set, interval-valued, 606
- fuzzy set, referential, 621
- fuzzy set, triangular, 612, 621, 623
- fuzzy set, type-2, 620
- fuzzy subspace, 650
- fuzzy system, 4, 641, 1071
- fuzzy-neuro system, 46
- FuzzyART, 46
- FuzzyARTMAP, 47
- gambling agent, 577
- game
 - computer, 970
 - strategy, 3
- game theoretic framework, 173
- game theory, 8, 155–157, 159, 161, 174, 175, 966
- game theory, evolutionary, 573
- game-playing strategy, 859

- Gandalf, 207
- Gaussian, 30, 122, 128, 725
- Gaussian basis function, 130
- Gaussian basis vector, 136
- Gaussian fuzzy set, 621, 623
- Gaussian kernel, 531, 741, 742, 747
- Gaussian membership function, 662
- Gaussian mixture, 738, 740
- Gaussian mutation, 863
- Gaussian noise, 135
- Gaussian pancake, 128
- gel electrophoresis, 1066, 1069, 1080, 1083
- gel electrophoresis, temperature gradient (TGGE), 1081
- gel, agarose, 1069
- Gelfond-Lifschitz transformation, 242
- gene, 1091
- gene data, 1102
- gene expression clustering, temporal, 747
- gene expression, microarray, 102
- gene network, 1091
- general packet radio services (GPRS), 398, 400
- general-purpose processor, 789
- generalization, 29, 356, 362, 529, 672, 852, 855, 859
- generalization, common, 357
- generalized delta learning rule, 27
- generalized harmony learning, 144
- generalized Horn (GH) clause, 239
- generally Horn program (GHP), 237, 239
- generation, 32, 536, 809
- generative topographic map (GTM), 111, 121, 129, 143, 147
- generic concept, 597
- generic evolution, 830, 832, 837
- generic topographic map (GTM), 732, 736
- genetic program representation, 929
- genetic programming
 - automatically defined function (ADF), 948
- genetic algorithm (GA), 20, 23, 32, 37, 45, 47, 98, 413, 415, 425, 536, 538, 542–544, 546, 550, 555, 556, 561, 563, 567, 571, 572, 643, 655, 798, 803–806, 830, 831, 837, 838, 854, 856, 883, 902, 1036, 1041, 1049, 1052, 1071
- genetic algorithm (GA) bit string, 884, 903
- genetic algorithm (GA) convergence, 906
- genetic algorithm (GA), augmented, 550
- genetic algorithm (GA), order based, 898
- genetic algorithm (GA), steady state, 891
- genetic algorithm, compact (cGA), 808, 837
- genetic algorithm, grouping (GGA), 896, 913
- genetic algorithm, non-dominated sorting (NSGA-II), 5
- genetic data, 1091
- genetic network programming, 1072
- genetic operator, 537, 550, 556, 570, 881, 884, 891, 928, 946, 1102, 1105
- genetic operator rates, 942
- genetic population, 32
- genetic programming
 - applications, 958
 - backward chaining, 975
 - Cartesian, 958
 - developmental, 954
 - geographically distributed, 978
 - grammar-based, 950
 - grammatical evolution, 952
 - initialization, 931
 - initialization, full method, 931
 - initialization, grow method, 931
 - linear representation, 955
 - machine code, 955
 - Markov model, 984
 - master-slave, 978
 - multi-objective, 968
 - neural network, 957
 - parallel, 977
 - parallel distributed, 957
 - Parisian, 966
 - representation
 - graph-based, 957
 - reverse polish, 981

- search spaces
 - theory, 984
- speedup techniques, 973
- strongly-typed
 - theory, 984
- tree-based representation, 931
- Turing complete, 956
- genetic programming theory, 982
- genetic programming (GP), 32, 36, 519, 536, 540, 541, 547, 548, 551, 555, 563, 566, 567, 570, 575, 576, 883, 927
 - multi-objective, 934
 - parallel, 979
- genetic programming closure, 938
- genetic programming problem solver (GPPS), 949
- genetic programming search space, 940
- genetic programming sufficiency, 939
- genetic simulated annealing (GSA), 906, 910, 916, 918, 919
- genetic simulated annealing (GSA) framework, 919
- genetic simulated annealing (GSA), order based, 913, 914, 917
- genetic string, 32
- genetic-fuzzy classifier system (GFCS), 571
- genetically altered penguin, 250
- genetics-based machine learning (GBML), 667
- GenNet, 804
- genome, 1091
- genome data, 1092
- genome sequence, 1080
- genotype, 1105
- geographical information, 402
- geographical information system (GIS), 425
- geographically distributed genetic programming, 978
- glass data set, 868
- global best (gbest), 1030, 1031, 1033, 1036–1038, 1040, 1045, 1046, 1051
- global minimum, 19, 27
- global optimum, 1029, 1036, 1044, 1049, 1052, 1054
- global positioning system (GPS), 382, 388, 397, 401–403
- global positioning system (GPS) agent, 398, 400
- global positioning system (GPS) receiver, 401
- Globus, 443, 444
- GNU C compiler (GCC), 420, 423, 430
- goodness, 94
- Google Directory, 390, 402
- GP-based agent, 555
- GPU speedup factor, 981
- grade, fuzzy membership, 606
- grade, membership, 605
- gradient descent, 27, 141
- gradient descent algorithm, 1053
- gradient descent, stochastic, 728
- gradient method, 733
- gradient-based optimization, 631
- grammar, 307
- grammar, context-free (CFG), 312, 324, 328
- grammar, context-sensitive
 - genetic programming, 952
- grammar, definite clause, 324
- grammar, head-driven phrase structure (HPSG), 314
- grammar, internalized, 309
- grammar, lexical-functional (LFG), 314
- grammar, probabilistic context-free (PCFG), 308, 328, 329
- grammar, stochastic context-free, 331
- grammar, stochastic lexicalized, 332
- grammar-based constraint, 950, 951
- grammar-based genetic programming, 950
- grammatical evolution, 952
- granular computing, 4
- granular mapping, 634
- granularity, 661
- granule, information, 598, 624, 625
- graph, 158, 352, 363
 - directed acyclic (DAG), 977
- graph coloring problem (GCP), 882, 887, 892, 893, 897, 899, 905, 912, 917
- graph matching, 368
- graph projection, 361
- graph theory, conceptual (CGT), 360
- graph unification, 367
- graph, canonical, 356

- graph, conceptual (CG), 351, 352, 357, 358, 363–366, 368–370, 372, 374
- graph, flat, 912
- graph, labeled, 352
- graph, Leighton, 912
- graph, resource description framework (RDF), 399
- graph, scale-free, 490
- graph, tree, 489
- graph, well-formed, 354
- graph-based genetic programming, 957
- Graphcode, 426, 429
- graphical knowledge representation, 352
- graphical user interface (GUI), 397, 422
- graphics processing unit (GPU), 980
- gray model, 1098
- greatest lower bound (GLB), 356, 358, 363
- greedy algorithm, 358, 899, 1032, 1057
- greedy algorithm, iterated (IG), 900
- greedy decoder, 887, 891, 920
- greedy partition crossover (GPX), 897
- grid, 690
- grid architecture, 440
- grid architecture, super-local, 440, 444
- grid backbone, 468
- grid computing, 155, 156, 174, 446, 691
- grid computing, service-oriented, 440
- grid container, 450, 466
- grid container, microkernel, 463
- grid coordinates, 491
- grid information service, 696
- grid inter-operability, 446
- grid management service, 464
- grid middleware, 689, 695
- grid scalability, 446
- grid service, 695
- grid technology, 689, 691
- grid, computational, 440
- grid, computing, 439, 441, 443, 444
- grid, distributed, 690, 696
- grid, electricity power, 441–443
- grid, heterogeneous, 690
- grid, media, 689–691, 695
- grid, power, 502
- grid, sensor, 488, 489
- grid, server-based, 440
- grid, service-oriented, 443
- group, artificial neural network (ANN), 31
- group, blood, 1108
- group, elevator, 1065
- grouping genetic algorithm (GGA), 896, 913
- grouping problem, 892
- grow method, genetic programming, 931
- growing hierarchical-SOM, 751
- guanine, 1066
- guidance system, intelligent tourist, 395
- habituation, 773, 775
- halting
 - program
 - Markov chain model, 985
- Hamiltonian circuit, 1105
- Hamiltonian path problem, 1066
- Hamming distance, 1097
- Handel-C, 798, 803, 805, 830, 832
- handheld device, 397, 398, 402, 403
- hardware compiler, 803–805, 830
- hardware description language (HDL), 803, 807
- hardware, analog, 803
- hardware, digital, 803
- hardware, embryonic, 804
- hardware, evolvable (EHW), 34, 803, 826, 883
- harmonic average, 141
- harmonic k-means clustering algorithm, 141
- harmonic means, K, 146, 147
- harmonic topographic map (HaToM), 121, 140, 142, 143, 147
- harmony learning, generalized, 144
- Hasse diagram, 290
- head-driven phrase structure grammar (HPSG), 314
- head-word, 308
- headless chicken crossover, genetic programming, 935
- health belief model, 213
- health professional, 213
- heart disease data set, 856, 857, 868
- Heatbugs, 416
- Hebbian learning, 113, 693, 716, 720, 723, 724, 734, 777, 782

- Hebrand base, 239
- Hebrand interpretation, 242
- Hebrand universe, 239
- helix, double-strand DNA, 1066
- helix, single-strand DNA, 1066
- Herman the Bug, 203, 207
- heterogeneous agent, 158, 413
- heterogeneous computing resource, 441
- heterogeneous grid, 690
- heuristic, 882, 969
 - hyper-, 969
- heuristic rule evaluation, 664
- heuristic rule extraction, 665
- heuristic search, 10, 16
- heuristic search engine, 97
- heuristic, bin packing, 914
- heuristic, clustering, 491
- heuristic, largest first, 900
- heuristic, local, 1057, 1058
- heuristic, ordering, 899
- heuristics, 7, 919, 1100, 1105, 1111
 - meta-, 969
- heuristics, Culberson and Luo (CL), 898, 902, 904, 919
- hidden layer, 29
- hierarchical noise tolerant coding
 - theory, 727
- hierarchical relationship, 382
- hierarchical temporal memory (HTM), 20
- hierarchy classification, 386
- hierarchy, concept type, 360
- hierarchy, dynamic, 489
- hierarchy, inheritance, 355, 362
- hierarchy, interval type, 371
- hierarchy, specialization, 358
- hierarchy, subsumption type, 356
- hierarchy, taxonomic, 362, 363
- hierarchy, type, 355, 359, 361, 363, 371, 374
- high-dimensional data set, 112, 146
- high-level language (HLL), 3, 6, 8, 18, 418, 798, 805
- high-performance computing (HPC), 443
- high-rise building, 1072
- high-speed networking, 689
- higher cognitive emotions, 188
- higher-order fuzzy set, 620
- higher-order logic (HOL), 9
- higher-order neural network (HONN), 13
- hill climbing, 909, 969
- histogram, 424, 425
- histogram estimator, 93
- Hodgkin-Huxley model, 766, 776
- homologous crossover
 - linear genetic programming, 955
 - theory, 984
- Hong Kong travel, 390, 402
- Hopfield network, 27, 693, 719
- horizontal fuzzy membership function
 - estimation, 605
- Horn clause, 238
- host agent, 1091
- HPSG annotation, 331
- HS algorithm, 735
- hub, 468, 489, 490
- human agent, 518, 545, 578
- human annotator, 309
- human brain, 767, 768, 799–801
- human competitive artificial intelligence, 962
- human competitiveness, 962
- human emotion, 208
- human expert, 859
- human health professional, 213
- human intelligence, 48
- human therapist, 213
- human tourist guide, 402
- human visual perception, 716
- human-competitive awards (Humies), 964
- human-competitive machine intelligence, 987
- human-computer interaction (HCI), 185
- human-human interaction, 203, 204, 216
- humanoid robot, 3
- hybrid coloring algorithm (HCA), 896
- hybrid computational intelligence (CI) system, 43
- hybrid evolutionary algorithm (HEA), 881, 913
- hybrid grouping genetic algorithm (HGGA), 916
- hybrid system, 19
- hybridization, 1066, 1067, 1071, 1080, 1082

- hyper-heuristic, 969
- hyperbolic tangent function, 521
- hypercube fuzzy set, 47
- hypertext, 174
- hypertext markup language (HTML), 384
- hypothesis, adaptive market, 967
- hypothesis, efficient market (EMH), 548, 565, 566, 573, 967
- hypothesis, permanent income, 573
- hypothesis, rational expectations (REH), 549, 565, 566, 573

- identification, user, 1105
- idiomatic phrase, 310
- if...then production rule, 9, 22, 23, 543
- iJADE FreeWalker, 382, 389, 395, 397, 400, 402
- iJADE FreeWalker client, 397
- iJADE FreeWalker server, 399
- iJADE tourist guidance system, 401
- iJADE tourist information center, 399
- image processing, 784, 965, 966
- image watermark, 966
- IMBS, 1105
- immediate before/after, 286
- immune algorithm, 1101–1103
- immune cell, 159
- immune memory, 1097
- immune network, 1096
- immune system, 1091–1093, 1106, 1110, 1113
- immune system model, 1096
- immunity, adaptive, 1113
- immunity-based computing (IBC), 40, 155
- immunity-based problem solver, 1106
- immunity-based problem solving, 1111, 1113
- immunity-based system (IMBS), 159, 1091, 1092, 1095, 1102, 1113
- immunologic memory, 1102
- immunology, 1092
- impulse generator module, 824
- included before/after, 287
- independent agent, 156
- independent component analysis (ICA), 721

- index, performance, 612, 614, 619, 631, 633
- index, projection, 734
- index, separation, 617
- indirect load management, 501
- individual learning, 556, 564, 567
- individual-based modeling (IBM), 410, 411
- individual-based modeling (IBM), vector-based, 1178
- inductive logic programming, 319
- inference engine, 22, 23
- inference system, adaptive-network fuzzy (ANFIS), 45
- inference system, fuzzy (FIS), 25, 45
- inference, probabilistic, 23
- inference, semi-strict, 258
- inflation rate, 544, 553, 555, 556, 562
- InfoDaemon, 700
- informatics, bio-, 968, 1092, 1110
- information, 14
- information conjunction, 364
- information granule, 598, 624, 625
- information merging, 363
- information processing, 766
- information processing system, 715, 763, 766, 768
- information retrieval (IR), 381, 751
- information retrieval (IR), mobile, 398
- information storage, 771
- information system (IS), 163, 173, 441, 1093
- information theory, 969
- information, domain, 367
- information, geographical, 402
- information, tourist, 382, 402, 403
- inheritance, 362
- inheritance hierarchy, 355, 362
- inheritance rule, 372
- inheritance, deontic, 257
- inheritance, feature structure (FS), 355
- inhibitory connection, 722
- initialization
 - genetic programming, 931
- input fuzzification, 25
- input layer, 29
- input signal register (ISR), 810
- input space, 741, 742
- input/output training data, 17, 26

- input/output transfer function, 17, 521, 526, 770, 778
- insertion mutation, 890, 903, 918
- instance, 352, 359, 360
- instance-based learning (IBL), 533
- instrument, scientific, 443
- Intel C++ compiler (ICC), 430
- intelligence, 3, 7, 719
- intelligence, artificial (AI), 3, 4, 7, 9, 26, 351, 381, 567, 927, 1071
- intelligence, collective, 36
- intelligence, computational (CI), 4, 16, 17, 20, 42, 155, 158, 174, 517, 570, 571, 579, 1092
- intelligence, human, 48
- intelligence, machine, 718, 927
- intelligence, mindless, 49
- intelligence, swarm (SI), 36
- intelligent agent, 21, 42, 386, 446, 543
- intelligent Java agent-based development environment (iJADE), 382, 389
- intelligent network services, 381
- intelligent system, 4, 17, 155, 158, 351, 769
- intelligent tourist guidance system, 395
- intension, feature structure (FS), 366
- intentional emotions, 190
- interpretation, Herbrand, 242
- inter module signaling interface (IMSI), 809–811, 813, 826, 838
- inter-agent communication, 489, 490, 502
- inter-agent interaction, 485
- inter-connectivity, module, 832, 833
- inter-operability, 439, 445, 447, 451
- inter-operability, grid, 446
- inter-task communication, 440, 445, 446
- interaction, agent, 409, 453, 1100
- interaction, inter-agent, 485
- interaction, multi-agent, 486
- interclass distance, 96
- interconnections, neuron, 764
- interface agent, 201, 206
- interface, graphical user (GUI), 422
- interface, robot-brain, 813
- interface, voice, 402
- internalized grammar, 309
- internet, 155, 174, 396, 417, 439, 440, 442, 444, 689
- internet being, 156, 174, 175
- internet streaming, 691
- interoperability, ontology, 372
- interpretability, 646, 673
- interpretability-accuracy tradeoff, 674, 675
- interpretability-complexity tradeoff, 643
- interval discretization, 643, 654, 655
- interval rule, 644, 649, 653–655
- interval rule-based classifier, 644, 653
- interval type hierarchy, 371
- interval, confidence, 606
- interval-valued fuzzy set, 606
- intrusion detection, 1105
- intrusion detection, network, 157
- invention
 - patentable, 963
- inversion operator, 885, 903, 912, 914
- ionosphere classification, 100
- ionosphere data set, 100
- iris data set, 750
- Isomap, 732
- iterated greedy algorithm (IG), 900, 916, 919
- iterated prisoner's dilemma (IPD), 160, 859
- iterative rule learning approach, 667
- itinerary, 403
- Izhikevich model, 775, 785
- Java, 418, 420–422, 424, 425, 427, 430
- Java applet, 41
- Java framework, 399
- Java Virtual Machine (JVM), 41, 400, 424, 428, 431
- jellyfish agent, 426
- Jena framework, 399
- Jerne network, 1105
- jitter, 699
- job management service, 440
- job scheduling, 440, 708
- join operator, 357, 361, 366, 371
- join rule, 355
- join subsumption, 374
- join, external, 356
- joint before/after, 286
- just-in-time compilation, 420, 431

- k nearest neighbors (KNN), 532
- k-harmonic means, 146, 147
- k-means agent, 535
- k-means clustering, 530, 863
- k-means clustering algorithm, 112, 141
- k-means clustering algorithm, harmonic, 141
- Kalman filter, 532, 541
- keep-it-simple, stupid (KISS), 571
- kernel function, 529, 740, 742, 774
- kernel method, 529, 740
- kernel principal component analysis (PCA), 734
- kernel regression, 735
- kernel self-organizing map (SOM), 740
- kernel, Gaussian, 531, 741, 742, 747
- kernel-based topographic map, 732
- keyword, 381
- keyword search, 381
- kinematics, 964
- KNN agent, 535
- knowledge, 14, 382, 383
- knowledge acquisition, 801
- knowledge base (KB), 8, 9, 22, 351
- knowledge combination, 364
- knowledge conjunction, 351, 363–369, 372, 374
- knowledge conjunction model, 369
- knowledge conjunction reasoning tool, 367, 370
- knowledge discovery (KD), 548, 549
- knowledge discovery (KD) tree, 102
- knowledge engineer (KE), 4, 8
- knowledge engineering (KE), 351, 381
- knowledge representation, 9, 352, 374
- knowledge representation system, 364
- knowledge representation tool, 353
- knowledge representation, graphical, 352
- knowledge retrieval, 373
- knowledge reuse, 384
- knowledge sharing, 384, 397
- knowledge structure, 365, 366
- knowledge, defense, 370
- knowledge, domain, 351, 363, 388, 597, 598, 624
- knowledge, expert, 363, 370
- knowledge-based system (KBS), 21
- knowledge-level reasoning, 369
- Kohonen map, temporal (TKM), 747
- Kohonen's self-organizing map (SOM), 117, 715
- Kolmogorov complexity, 548
- Kolmogorov equation, 160
- Kolmogorov-Sinai entropy, 495
- Kullback-Leibler information distance, 738
- Kullback-Liebler divergence, 87, 95
- kurtosis, 566
- kurtotic noise, 135
- Lévy mutation, 863
- label substitution, 309
- label substitution operation, 313
- labeled graph, 352
- labeled tree, 319
- Lagrange multiplier, 615
- landscape, energy, 27
- landscape, error, 27
- language semantics, 373
- language comprehension, 311
- language derivation, 320
- language, agent communication, 42
- language, agent implementation, 419, 431
- language, C, 807
- language, camera, 813
- language, extensible markup (XML), 383, 384, 388
- language, formal, 42
- language, hardware description (HDL), 803, 807
- language, high-level (HLL), 3, 6, 8, 18, 418, 798
- language, hypertext markup (HTML), 384
- language, LISt Processing (LISP), 8
- language, LOGic PROgramming (Prolog), 3, 8, 9
- language, low-level, 18
- language, markup, 381
- language, natural, 18, 571
- language, ontology, 374, 383
- language, simple protocol and RDF query (SPARQL), 399
- language, standard generalized markup (SGML), 384

- language, web ontology (OWL), 385, 388, 392, 399, 400
- Laplacian, 135
- Laplacian noise, 135
- largest first heuristic, 900
- latent point, 121, 123, 125, 126, 129, 131, 133, 136, 139
- latent space, 121, 123, 125, 126, 133, 135
- lateral inhibition, 716, 717
- lattice, 356, 371
- lattice, bi-, 236, 290, 292
- lattice, complete, 235, 245, 271–274
- Laura, 202, 207, 210, 213
- layer, hidden, 29
- layer, input, 29
- layer, logic, 383
- layer, output, 29
- layer, proof, 383
- layer, trust, 383
- lazy learning, 518, 534
- leaky integrate-and-fire (LIF) model, 772–775, 779, 785
- learning, 14, 719, 802
- learning algorithm, 641
- learning algorithm, backpropagation (BP), 27, 28, 39
- learning algorithm, Oja, 721
- learning rate, 720, 723, 725, 854, 857, 868
- learning rule, 781, 782, 851
- learning rule, artificial neural network (ANN), 851
- learning rule, Bayesian, 575
- learning rule, delta, 27
- learning rule, min-max, 47
- learning rule, ToPoE, 128
- learning supervised, 692
- learning theory, 21
- learning vector quantization (LVQ), 29, 114, 726
- learning, adaptive, 767
- learning, anti-Hebbian, 777
- learning, co-evolutionary, 859
- learning, competitive, 530, 724
- learning, computational, 307
- learning, enhanced, 1072
- learning, error-correction, 719
- learning, explanation-based, 319
- learning, Hebbian, 113, 716, 720, 723, 724, 734, 777, 782
- learning, individual, 556, 564, 567
- learning, instance-based (IBL), 533
- learning, lazy, 518, 534
- learning, machine (ML), 9, 16, 856, 927
- learning, negative correlation (NCL), 860, 866
- learning, population, 519
- learning, reinforcement, 571, 580, 720, 851
- learning, relational, 307
- learning, social, 519, 556, 563, 567
- learning, supervised, 112, 518, 719, 740
- learning, unsupervised, 112, 334, 518, 693, 715, 719, 731
- least mean square (LMS) learning, 27, 693
- least squares error, 856
- least upper bound (LUB), 358, 367
- Leighton graph, 912
- Lena image, 743
- length of derivation, 313
- letter recognition data set, 868
- lexical grammar, stochastic, 332
- lexical-functional grammar (LFG), 314
- lexical-syntactic category, 309
- LFG annotation, 331
- LFG-annotated corpora, 314
- library, space, 424, 426, 429
- library, XML parsing, 450
- lie detector, 199
- life, artificial (Alife), 156, 174, 409, 411, 417
- ligation, 1066, 1067, 1080, 1082
- linear combination method, 853, 855
- linear combination method, rank-based, 853, 855, 858
- linear fuzzy membership function, 613
- linear genetic programming (GP), 955
- linear projection, 733
- linear regression model, 534, 570
- linear separability, 11, 27, 740
- linearization, 601
- linguistic interpretability, 641
- linguistic parsing, 327
- linguistic productivity, 310
- linguistic term, 598, 645, 649
- linguistic value, 642, 645

- linguistic variable, 24
- linguistically quantified variable, 597
- linguistics, psycho-, 311
- link, subtype, 355
- LISt Processing (LISP) language, 3, 8
- list programming (LISP), 546
- list, preference, 1109, 1110
- list, tabu, 1040
- literal, 247, 254, 255, 257, 262, 270, 285, 295
- literal, annotated, 238
- literal, bf-, 285, 293, 294
- literal, bf-EVALP, 290, 294
- literal, vector annotated, 243, 244
- literal, well extended vector annotated (WEVA), 246
- literal, well vector annotated (WVA), 244
- load balancing, 440, 443, 450, 463
- load balancing, network, 37
- load management, direct, 501
- load management, indirect, 501
- local area network (LAN), 164, 442, 447
- local best (lbest), 1030, 1033, 1034, 1036, 1037, 1051
- local heuristic, 1057, 1058
- local minima, 732, 743, 854
- local minimum, 19, 27
- local optima, 1036, 1044, 1049
- local optimum, 1029, 1044, 1057
- local scheduler, 440, 446, 447
- local search, 659, 663, 665, 870, 1053, 1059
- location, 388
- location awareness, 395, 402
- location-aware mobile tourist guidance system, 395
- location-aware tourist information, 402
- location-aware tourist information retrieval system, 403
- locking, process, 280, 283
- logic circuit, Boolean, 1071
- logic gate, 764, 772, 806, 837
- logic gate, universal, 764
- logic layer, semantic web, 383
- logic network
 - evolution, 957
- logic program, 237, 238, 242
- logic program, paraconsistent, 235
- logic program, paraconsistent
 - annotated, 238
- logic program, stochastic, 324
- logic programming, 233–235, 264, 307, 364
- LOGIC PROgramming language (Prolog), 3, 8, 9
- logic programming, annotated, 233
- logic programming, constraint, 364
- logic programming, inductive, 319
- logic, annotated, 233, 235
- logic, autoepistemic, 233
- logic, Boolean, 23, 41, 766
- logic, default, 233
- logic, defeasible, 233, 244, 249, 254
- logic, defeasible deontic, 244, 257
- logic, deontic, 244
- logic, description, 373, 386
- logic, first-order (FOL), 9, 22, 360, 373
- logic, formal, 9
- logic, fuzzy, 18, 23, 24, 45, 519, 1072
- logic, higher-order (HOL), 9
- logic, modal, 244
- logic, multi-valued, 25
- logic, Nute's defeasible deontic, 247, 256
- logic, paraconsistent, 233
- logic, paraconsistent annotated, 235
- logic, predicate, 1071
- logic, propositional, 9, 22
- logic, propositional paraconsistent
 - annotated, 235
- logic, two-valued, 25
- logical entity, 267
- logical state, 267
- logistic function, 698
- Logo, 427
- long-horizon agent, 575
- long-term depression (LTD), 777
- long-term memory (LTM), 764, 777
- long-term potentiation (LTP), 777
- lookup table (LUT), 35, 47, 649, 774, 809, 838
- Lotka-Volterra equation, 1097
- low pass filter, 1031
- low-level language, 18
- lower bound, 357, 360
- lower bound, greatest (GLB), 356, 358, 363
- Lucasian economic agent, 541–544

- machine code, 546
- machine code genetic programming (GP), 955
- machine intelligence, 718, 927
 - human-competitive, 987
- machine learning (ML), 4, 9, 14, 16, 641, 675, 690, 828, 856, 927
- machine translation, 308
- machine, Turing (TM), 1066
- machine, virtual (VM), 420, 424
- MACK, 207
- macroeconomic model, agent-based, 556
- macroeconomic modeling, 562
- macroeconomics, 541, 544, 549, 572, 575
- macroeconomics, agent-based, 573
- macroeconomics, neo-classical, 549, 572
- magnetic bead, 1069, 1080, 1083
- magnetic resonance imaging (MRI), 763
- Mahalanobis distance, 95
- mainframe computer, 441
- maintenance, self, 156, 159
- major histocompatibility complex (MHC), 1092
- majority class, 650
- majority voting, 44, 853, 855, 857, 863, 865, 871
- mammalian brain, 773, 836
- management agent, 450, 463
- management, computing, 464, 466, 474
- management, elevator group, 1065
- management, energy, 501
- management, grid, 464
- management, power load, 501
- management, resource, 443, 446, 462, 471
- manifold mapping, 732
- manipulation, symbol, 7
- map, cluster, 491, 493
- map, cortex, 724
- map, fuzzy cognitive (FCM), 23
- map, generative topographic (GTM), 111, 121, 129, 143, 147
- map, generic topographic (GTM), 732, 736
- map, harmonic topographic (HaToM), 140, 143, 147
- map, retinotopic, 114
- map, self-organizing (SOM), 142, 146, 725
- map, self-organizing feature (SOFM), 725
- map, somatosensory, 114
- map, temporal Kohonen (TKM), 747
- map, tonotopic, 114
- map, topographic, 111, 146, 721
- map, topological, 725
- map, topology preserving, 111, 114, 121, 146, 715, 726
- mapping function, 740
- mapping, fuzzy, 624, 634
- mapping, granular, 634
- mapping, manifold, 732
- mapping, relational, 629
- mapping, retina-cortex, 715
- mapping, Sammon, 119, 732, 750
- market diversity, 567
- market economy, 157
- market efficiency, 568
- market force, 544
- market forecasting rule, 413, 415
- market modeling, 411
- market price, 544
- market, financial, 577
- Markov chain model
 - program execution, 985
- Markov chain theory
 - evolutionary algorithms, 984
- Markov model, 159–161
 - genetic programming, 984
- Markov random field (MRF) model, 744
- markup, 384
- markup language, 381
- markup language, extensible (XML), 384, 388
- markup language, hypertext (HTML), 384
- markup language, standard generalized (SGML), 384
- marriage problem, stable (SMP), 1106, 1107, 1113
- Mason, 413, 416, 417, 421, 424–426, 428, 430, 431
- massively parallel processor (MPP), 442
- master-slave, 443
- master-slave genetic programming, 978
- matching, graph, 368
- matching, pattern, 113

- mathematical function approximation, 28
- mathematical model, 443
- mating, 32
- mating pool, 537, 538
- matrix, affinity, 1106, 1109
- matrix, covariance, 734
- matrix, mutual correlation, 1099
- matrix, partition, 614
- matrix, payoff, 162
- matrix, resource, 471
- maximum compatibility method, 652
- maximum likelihood estimator, 315
- maximum likelihood learning algorithm, 84
- McCulloch & Pitts neuron model, 26, 778
- mean field analysis, 166
- mean squared error (MSE), 16, 701, 733
- meaning, 382
- meaning-based search, 381
- means, k-harmonic, 146, 147
- means-ends analysis (MEA), 1110, 1113
- measure, bf-, 290
- measure, performance, 173, 905, 912
- measure, symmetric similarity, 730
- mechanism design, 158
- mechanism design, distributed
 - algorithmic, 158
- mechanism, scheduling, 471
- media content, 690
- media content format, 699
- media grid, 689–691, 695
- media grid portal, 695
- media resolution, 699
- media server, 702
- media streaming, 689, 690
- media streaming, QoS-aware, 690
- medical imaging, 966
- medication, 1091, 1092
- medicine, personalized, 1091, 1092
- meet operator, 357, 361
- melodic analysis, 308, 315
- membership degree, fuzzy set, 617
- membership estimation, fuzzy set, 620
- membership estimation, user-based
 - fuzzy set, 620
- membership function, 642, 643, 647, 648, 655, 668
- membership function estimation,
 - data-driven fuzzy, 598
- membership function estimation, fuzzy, 597
- membership function estimation,
 - horizontal fuzzy, 605
- membership function estimation,
 - pairwise fuzzy, 605, 607
- membership function estimation,
 - priority fuzzy, 607
- membership function estimation,
 - user-centric, 599
- membership function estimation,
 - vertical fuzzy, 605, 606
- membership function, asymmetric, 662
- membership function, fuzzy, 47, 601, 608
- membership function, Gaussian, 662
- membership function, linear fuzzy, 613
- membership function, trapezoidal, 662
- membership function, unimodal fuzzy, 610
- membership grade, 598, 605
- membership, fuzzy set, 25
- membrane-based computing (MBC), 40
- memetic algorithm, 919
- memetic EMO algorithm, 663
- memetic evolutionary algorithm, 881
- memetic Pareto artificial neural network (MPANN), 870
- memory, 518, 802
- memory optimized accelerator for spiking neural networks (MASPINN), 784
- memory, associative (AM), 715, 724, 726, 782
- memory, bidirectional associative (BAM), 693
- memory, collective, 1032
- memory, computer, 1065, 1066
- memory, correlation matrix (CMM), 782
- memory, hierarchical temporal (HTM), 20
- memory, immune, 1097
- memory, immunologic, 1102
- memory, long-term (LTM), 764, 777
- memory, non-holographic, 782
- memory, short-term (STM), 597, 764

- memory, social collective, 1030
- memory, sparse distributed (SDM), 782
- merging crossover (MOX), 888, 890, 903
- merging independent sets crossover (MIS), 904, 912, 914
- merging, ontology, 363
- message passing, 440, 442
- message passing interface (MPI), 442, 980
- message passing/routing framework, 462
- message routing, 440
- message, cluster-information, 493
- message, forward, 492
- message, recruit, 491
- meta-heuristics, 969
- metacomputing, 442
- metadata, 383, 388
- metaheuristic algorithm, 881
- metaphysics, 9
- method, combination, 855, 863
- method, data projection, 732
- method, filter, 83
- method, gradient, 733
- method, kernel, 740
- method, linear combination, 853, 855
- method, Newton optimization, 733
- method, non-linear combination, 853
- method, nonparametric, 735
- method, numerical, 734
- method, population-based, 1105
- method, subset, 853, 858
- method, wrapper, 83
- metric entropy, 495
- metric, performance, 431, 692
- metric, similarity, 747
- mexican hat function, 724
- MI-based feature selection method with uniform modification (MIFS-U), 93
- MI-raw based feature selection method (MI-raw-FS), 98
- Michigan approach, 667
- micro-architecture, cortical, 766
- micro-architecture, neural, 766, 767
- micro-chip, 767
- micro-macro relation, 572
- microarray gene expression data set, 102
- microkernel grid container, 463
- microprocessor, 786
- middleware, 444
- middleware, grid, 689, 695
- millions of instructions per second (MIPS), 785
- Mimic, 210
- mimicking, biological, 1092
- min-max learning rule, 47
- min-max neural network, fuzzy, 47
- mind, 14, 764
- mindless intelligence, 49
- minima, local, 732, 743, 854
- minimization, 884
- minimization function, 1054
- minimum description length, 969
- minimum energy broadcast problem, 487
- minimum operator, 648
- minimum, global, 19, 27
- minimum, local, 19, 27
- mining, data (DM), 1101
- mining, document, 726
- mining, text, 117, 726
- minor components analysis, 128
- MIPS per watt, 785
- mission rehearsal exercise (MRE), 198
- Mitra's multi-scale method, 87
- mix operation, 1070
- mix, controlled, 268, 272, 275
- mixture of experts, 121, 129, 131, 141, 146
- Mixture-of-Experts, 44
- ML-DOP, 315
- mobile (cell) phone, 382, 397
- mobile ad hoc network, 488
- mobile agent, 389, 395, 397, 398, 401, 402
- mobile information retrieval (IR), 398
- mobile service, context-aware, 388
- modal logic, 244
- model car racing, 970
- model of emotion, 197
- model, adaptive, 570
- model, agent-based (ABM), 411, 419, 428, 431, 517, 549, 550, 560, 561, 568–572, 574–579
- model, beliefs-desires-intentions (BDI), 42

- model, black-and-white, 1098
- model, Black-Scholes, 548
- model, channel noise, 728
- model, cobweb, 544, 549, 552, 571
- model, compartmental, 776
- model, computational, 233
- model, computational (emotion), 197
- model, density matching, 738
- model, digital spike communications, 782
- model, domain ontology, 403
- model, Domany-Kinzel (DK), 165
- model, embedded, 91
- model, EVALPSN stable, 265
- model, fuzzy, 47
- model, gray, 1098
- model, Hodgkin-Huxley, 766, 776
- model, immune system, 1096
- model, Izhikevich, 775, 785
- model, knowledge conjunction, 369
- model, leaky integrate-and-fire (LIF), 772–775, 779, 785
- model, linear regression, 570
- model, Markov, 159–161
- model, mathematical, 443
- model, noise, 135
- model, parametric, 570
- model, perfect, 264
- model, point-neuron, 773, 774, 776, 785
- model, shape-space, 1097
- model, skeptical, 1098
- model, spike response, 774
- model, spiking neuron, 782
- model, stable, 255, 256
- model, stupid, 426–429, 431
- model, task/service (TS), 451, 457, 462
- model, VALPSN stable, 258
- modeling environment, agent-based (ABM), 416, 431
- modeling framework, agent-based (ABM), 421, 423, 424, 431
- modeling platform, agent-based (ABM), 419, 428, 429, 431
- modeling system, agent-based (ABM), 409, 418, 420–422, 426
- modeling, brain, 769
- modeling, cognitive, 799
- modeling, individual-based (IBM), 410, 411
- modeling, macroeconomic, 562
- modeling, market, 411
- modeling, neural, 768, 774, 782, 786, 789
- modeling, population-based, 410
- modeling, time series, 28, 533
- modeling, traffic, 411
- modeling, wavelet, 692
- modular artificial neural network (ANN), 859
- modular software development, 598
- modularization, automatic, 859
- module inter-connectivity, 832, 833
- module, artificial neural network (ANN), 797, 798, 804, 805, 807–809, 813, 826, 827, 830, 834, 837, 838
- module, behavior control, 814
- module, boredom meter, 824
- module, decision, 814
- module, impulse generator, 824
- module, multigen, 823, 827
- module, pattern detector, 830
- module, pattern recognition, 814, 836
- module, randomizer, 825
- module, threshold detector, 825
- molecular biology, 1065
- molecular computing, 1065, 1066
- molecular dynamics (MD), 410
- molecule, DNA, 1065, 1069, 1070, 1080, 1082
- momentum, 29, 868
- momentum, swarm particle, 1031, 1032, 1036
- money supply, 562
- monitor processor, 786
- Mono, 424
- Monte Carlo simulation, 411, 423
- moods, 190
- Moore's Law, 48, 767, 799, 800, 803, 835, 837
- morphism, 357
- most probable derivation, 329
- Mousetrap, 417
- moving-average model, 561
- Mozart's G minor symphony, 316
- MTP algorithm, 916
- multi-agent, 382

- multi-agent algorithm, decentralized, 501
- multi-agent coalition, 502
- multi-agent data analysis system, 694, 696
- multi-agent dynamics, 489, 503
- multi-agent interaction, 486
- multi-agent network, 487, 489
- multi-agent network, self-organizing, 488
- multi-agent organism, 156
- multi-agent system (MAS), 23, 42, 386, 388, 485, 488
- multi-agent system (MAS) ontology, 42
- multi-cast routing, 786
- multi-dimensional scaling, 118
- multi-layer perceptron (MLP), 13, 27, 91, 520–524, 693, 697, 717, 719, 864
- multi-level performance metric, 702
- multi-modal distribution, 128
- multi-module evolution, 834
- multi-objective evolutionary algorithm (EA), 870
- multi-objective evolutionary optimization, 5
- multi-objective fitness, 942
 - genetic programming, 966
- multi-objective genetic programming (GP), 934, 968
- multi-objective optimization, 866, 870
- multi-point crossover, 885
- multi-scale method, 84
- multi-storey building, 1065, 1073
- multi-test evolution, 828–830
- multi-valued logic, 25
- multidimensional scaling (MDS), 732, 737
- multigen module, 823, 827
- multimedia file exchange, 444
- multimedia services, 689, 690
- multimedia, real-time, 689
- multiobjective fuzzy GBML algorithm, 667
- multiobjective fuzzy genetics-based machine learning (GBML), 644
- multiobjective fuzzy rule selection, 663, 665
- multiobjective optimization (MO), 643, 655, 658, 920
- multiple optima, 1037
- multiply constrained problem, 920
- multiprocessor, 786
- multivariate auto-regression (MAR), fuzzy, 48
- music
 - evolutionary, 970
- music analysis, computational, 307
- music derivation, 320
- musical parsing, 327
- mutation, 1101, 1104, 1105
 - genetic programming, point, 935
 - genetic programming, subtree, 935
- mutation operator, 31, 32, 36, 47, 537, 539, 550, 556, 656, 657, 806, 808, 853, 871, 885, 887, 891, 906, 909, 912, 914, 928, 946
- mutation probability, 657, 664, 668
- mutation, biased, 664, 665
- mutation, bit-flip, 657, 664
- mutation, Cauchy, 863
- mutation, Gaussian, 863
- mutation, insertion, 890, 903, 918
- mutation, Lévy, 863
- mutation, non-biased, 664
- mutation, non-Gaussian, 863
- mutation, order based, 890, 903
- mutation, position-based, 890
- mutation, super-, 909
- mutation, swap, 890
- mutual copying, 156, 159, 163
- mutual correlation matrix, 1099
- mutual information (MI), 92, 93
- mutual information-based feature selection method (MIFS), 93, 98
- mutual repair, 159, 160, 162, 1093
- mutual voting, 1098
- mutual, copying, 1093
- naïve Bayesian, 44
- nanotechnology, 1071
- Nash equilibrium, 155–157, 173–175
- natural language, 18, 571
- natural language processing (NLP), 3, 307, 334, 402
- natural selection, 31, 542, 544, 883

- Nature, 6, 17, 20, 27, 32, 40, 49, 536, 716, 1029
- Nature-inspired computing (NIC), 40
- nearest neighbor, 84, 535
- nearest neighbor (NN) model, 1082
- nearest neighbour rule, 99
- nearest trajectory algorithm, 532
- negation, 236, 262
- negation as failure, 264
- negation, epistemic, 236, 244, 246, 262, 272–274, 289
- negation, ontological, 233, 237
- negation, strong, 233, 237, 264
- negative correlation learning (NCL), 860, 866
- negative emotions, 205
- negotiation, agent, 159
- neighbor agent, 156, 158, 164, 170, 173
- neighbor unit, 1093
- neighbourhood, 20, 724
- neighbourhood function, 114, 725, 727, 728, 731, 737, 740, 742–744
- neighbourhood, swarm particle, 1033, 1036, 1045
- neo-classical economics, 576
- neo-classical macroeconomics, 549, 572
- neocortex, 49, 801
- netlist, connectivity, 786
- netlist, neural, 768
- NetLogo, 413, 421, 425–428
- network bandwidth, 397, 403, 691
- network bandwidth prediction, 691
- network capacity, 691
- network cleaning, 1093
- network convergence, 27, 29
- network intrusion detection, 157
- network latency, 387
- network load, 387
- network load balancing, 37
- network over-training, 28
- network performance, 692
- network pruning, 29
- network resources, 689
- network services, intelligent, 381
- network structure, 488
- network throughput, 691
- network topology, 488, 782
- network traffic, 387, 397
- network training time, 17, 28, 29
- network weather service (NWS), 692
- network, active sensor, 488
- network, agent, 158, 426
- network, artificial neural (ANN), 26, 45, 47, 733, 749, 883
- network, Bayesian, 5, 10, 21, 23, 580, 1100
- network, belief, 5, 10, 23
- network, brewery pipeline, 267
- network, computer, 442, 444
- network, counterpropagation, 693
- network, dynamic, 488, 1097
- network, feedback, 693
- network, feedforward, 693
- network, feedforward associative, 734
- network, gene, 1091
- network, Hopfield, 27, 693, 719
- network, immune, 1096
- network, Jerne, 1105
- network, local area (LAN), 164, 442, 447
- network, mobile ad hoc, 488
- network, multi-agent, 487, 489
- network, pipeline, 270
- network, principal component, 721
- network, randomly connected, 491
- network, recurrent, 26, 693, 719
- network, scale-free, 158, 489–491, 494
- network, self-organizing, 724
- network, self-organizing multi-sensor, 488
- network, self-repairing, 168, 169, 175, 1093
- network, semantic, 9, 22, 352, 1071
- network, sensor, 158, 487–489
- network, social, 517, 578
- network, subspace, 734
- network, supervised, 26
- network, unsupervised, 26
- network, wide area (WAN), 442
- network, wireless ad hoc, 157
- network-on-chip (NoC)
 - communications, 785, 786
- networked agent, 156
- networked recognition, 1096, 1100, 1105
- networked selfish agent, 158
- networking, high-speed, 689
- neural architecture, 719
- neural code, 764, 769

- neural computation, 769, 789
- neural cortical column, 800
- neural Darwinism, 802
- neural dynamics, 771
- neural gas, 731
- neural micro-architecture, 766, 767
- neural model, analogue, 782
- neural modeling, 768, 774, 782, 786, 789
- neural modeling system, 771
- neural netlist, 768
- neural network
 - evolution, 957, 965
- neural network modeling, spiking, 785
- neural network predictor, 692
- neural network, artificial (ANN), 4, 5, 9, 17, 20, 23, 41, 120, 403, 425, 518, 519, 532, 536, 579, 641, 715, 733, 749, 772, 778, 781, 782, 798, 803, 831, 837, 851, 860, 872, 1071
- neural network, auto-associative (AANN), 525
- neural network, biological, 769
- neural network, feedforward, 526, 719
- neural network, higher-order (HONN), 13
- neural network, spiking (SPINN), 784, 801
- neural processor, 782
- neural system, 767, 778, 782
- neural system adaptation, 769, 770
- neural system, analogue, 768
- neural system, biological, 769, 780, 784, 786
- neural systems engineering, 768, 777, 783
- neural winter, 27, 32
- neuro-fuzzy system, 45
- neurobiology, 763
- neurocomputer for spiking neural networks (NESPINN), 784
- neurological model of emotion, 189
- neuromorphic computing, 768, 782, 789
- NEURON, 784
- neuron, 763, 764, 767–770, 772–774, 777, 779
- neuron activation, 775, 804
- neuron dynamics, 775
- neuron interconnections, 764
- neuron model, McCulloch & Pitts, 26, 778
- neuron model, spiking, 785
- neuron, artificial, 801, 804, 811
- neuron, biological, 26, 766, 767, 773, 775
- neuron, fuzzy, 628
- neuroplasticity, 766
- neuroscience, 799–801, 836
- neutralization, noise, 1102, 1104
- neutralizing signal, 1105
- Newton optimization method, 733
- niche behavior, 1045
- nicheing particle swarm optimization (PSO), 1038
- NLP benchmark, 314
- node, abnormal, 1094, 1095
- node, computing, 446, 447
- node, normal, 1094
- noise cancelation, 1102
- noise filtering, 17, 21, 28
- noise model, 135
- noise neutralization, 1102, 1104
- noise tolerance, 535, 715, 743
- noise tolerant vector quantization (VQ), 731
- noise, Gaussian, 135
- noise, kurtotic, 135
- noise, Laplacian, 135
- noise, uniform, 135
- noisy data, 81
- non-algorithmic approach, 17
- non-biased mutation, 664
- non-deterministic search engine, 98
- non-dominated sorting, 870
- non-dominated sorting genetic algorithm (NSGA), 5, 659
- non-Gaussian mutation, 863
- non-hierarchical clustering algorithm, 530
- non-holographic memory, 782
- non-linear combination method, 853
- non-linear dynamics, 565
- non-linear forecasting model, 524
- non-monotonic formalism, 244
- non-monotonic reasoning, 233, 242
- non-monotonic system, 233
- nonhead-word, 308
- nonintentional emotions, 190

- nonlinear data projection, 732
- nonlinear principal component analysis (NLPKA), 734
- nonparametric method, 735
- nonself, 159, 1092, 1093, 1097, 1102, 1105, 1113
- normal agent, 164, 166, 168, 170, 172, 1098
- normal node, 1094
- normalization, 723
- normalization, weight, 720
- normative reasoning, 257
- notation, fuzzy set, 599
- notion, deontic, 247
- noun phrase (NP), 309, 310, 316
- NP-complete problem, 1065, 1071
- NP-hard, 487, 881
- NP-hard problem, 1065
- nucleotide, 1080
- numeric regression, 943
- numerical control, 967
- numerical method, 734
- Nute's defeasible deontic logic, 247, 256

- object-based programming, 41
- object-oriented programming (OOP), 22, 41
- object-oriented programming (OOP) language, 419
- Objective C, 420, 421, 423
- objective function, 32, 162, 527, 536, 614, 643, 731, 742, 884, 893, 904, 919, 1043
- objective space, 658, 660
- obligation, 244, 247, 262
- offspring, 537, 550, 656, 668, 853, 871, 882, 892, 896, 903
- offspring agent, 412
- Oja learning algorithm, 721
- Olga, 207
- oligonucleotide, 1067, 1080–1082
- one point crossover, 656, 885, 903
- one-point crossover theory, 984
- one-point crossover, genetic programming, 935
- oneself, 1106
- ontological negation, 233, 237

- ontology, 9, 19, 23, 351, 354, 358, 359, 362, 365, 372, 374, 381, 383, 384, 395, 401
- ontology interoperability, 372
- ontology language, 374, 383
- ontology merging, 363
- ontology model, domain, 403
- ontology tool, 373
- ontology, building design, 368
- ontology, cuisine, 393, 394
- ontology, domain, 382
- ontology, multi-agent system (MAS), 42
- ontology, travel, 389, 390, 393, 399, 402
- ontology, travel website, 382
- ontology, unification, 365
- ontology-based agent, 381
- ontology-based tourist guidance system, 389
- Open Directory, 390, 402
- open environment, 439, 440, 444, 446, 485
- open grid services architecture (OGSA), 444
- open source, 411, 417, 420, 428
- operation, amplify, 1071
- operation, combination, 321
- operation, detect, 1071
- operation, mix, 1070
- operation, separate, 1066, 1070
- operation, substitution, 309, 317, 320, 321, 324–326
- operator
 - architecture-altering, 928, 949
 - crossover, 928, 947
 - genetic, 928, 946
 - mutation, 928, 946
 - reproduction, 929, 946
- operator, crossover, 31, 32, 36, 47, 537, 550, 556, 656, 668, 806, 853, 871, 885, 887, 891, 896, 902, 906, 919
- operator, deontic, 256, 262
- operator, election, 550, 556
- operator, expectation, 553
- operator, fuzzy set, 25
- operator, genetic, 537, 550, 556, 570, 881, 891, 1102, 1105
- operator, inversion, 885, 903, 912, 914
- operator, join, 361, 366, 371
- operator, meet, 361

- operator, minimum, 648
- operator, mutation, 31, 32, 36, 47, 537, 539, 550, 556, 656, 657, 806, 808, 853, 871, 885, 887, 891, 906, 909, 912, 914
- operator, product, 648
- operator, projection, 361, 362
- operator, recombination, 537, 539, 853, 1102
- operator, reproduction, 31, 32, 550
- operator, selection, 556, 891
- operator, unify, 361
- opinion, expert, 598
- optical character recognition (OCR), 966
- optima, local, 1036, 1044, 1049
- optima, multiple, 1037
- optimal elevator scheduling, 1074
- optimal scheduling, 1065
- optimal search engine, 97
- optimal vector quantization (VQ), 727
- optimal, Pareto, 556, 572
- optimization, 27, 32, 403, 558, 881, 883, 904
- optimization algorithm, 1030, 1032, 1040, 1048, 1049
- optimization method, Newton, 733
- optimization problem, 601, 626, 634
- optimization, ant colony (ACO), 39
- optimization, combinatorial, 1071
- optimization, dynamic, 1043
- optimization, function, 549, 1044
- optimization, gradient-based, 631
- optimization, multi-objective, 866, 870, 920
- optimization, multi-objective evolutionary, 5
- optimization, particle swarm (PSO), 5, 38, 47, 1029, 1041, 1046, 1051
- optimizer, compiler, 420
- optimum, global, 1029, 1036, 1044, 1049, 1052, 1054
- optimum, local, 1029, 1044, 1057
- order based framework, 919
- order based genetic algorithm (GA), 898
- order based genetic simulated annealing (GSA), 913, 914, 917
- order based mutation, 890, 903
- order based representation, 887
- order crossover (OX), 888, 889, 903
- order, process, 284, 294, 295, 297
- ordering, 354
- ordering heuristic, 899
- ordering, topographical, 715
- organism multi-agent, 156
- organism, digital, 410
- organization, self-, 156, 163, 448, 453, 486, 719–721
- organization, virtual, 443
- Ortony Clore Collins (OCC) model, 197
- Othello, 970
- output de-fuzzification, 25
- output layer, 29
- output signal register (OSR), 810
- over-fitting, 28, 91, 92, 641, 672, 969, 972, 973
- over-training, network, 28
- overlapping generations (OLG) model, 552, 554, 555, 557, 558, 560–562, 573
- overlapping generations (OLG) model, two-period, 552, 556
- OWL DL, 386
- OWL Full, 386
- OWL Lite, 386
- Pac-Man, 970
- packet-switching communications system, 786
- pair, blocking, 1106, 1108
- pairwise comparison, 598, 608, 634
- pairwise fuzzy membership function estimation, 605, 607
- Palm Pilot, 402
- paraconsistent annotated logic, 235, 300
- paraconsistent annotated logic program, 238
- paraconsistent annotated logic, propositional, 235
- paraconsistent before/after, 288
- paraconsistent logic, 233
- paraconsistent logic program, 235
- paraconsistent vector annotation, 300
- paradigm, agent-based, 419
- paradigm, computing, 156
- parallel algorithm discovery and orchestration, 958

- parallel computer, 440, 1065
- parallel computing, 41, 442, 977
- parallel distributed genetic programming (GP), 957
- parallel evolutionary algorithm (EA), 977
- parallel genetic programming (GP), 977, 979
- parallel processing, 980, 1065, 1070
- parallel processing, super-, 1066
- parallel virtual machine (PVM), 442, 980
- parallel, super-, 1069
- parallelism, 767
- parameterized self-organizing map (PSOM), 731
- parameters, 598
- parametric model, 570
- parasitic computing, 155, 156, 174
- parent, 537, 550, 656, 885, 896, 903
- parent selection, 668
- Pareto differential evolution, 871
- Pareto front, 658
- Pareto optimal, 556, 572
- Pareto-inferior equilibrium, 554
- Pareto-optimal fuzzy rule-based system, 643
- Pareto-optimal rule set, 664
- Pareto-optimal solution, 658
- Pareto-superior equilibrium, 554, 555
- Parisian genetic programming, 966
- parse accuracy, 308
- parse tree, 308, 314
- parsed corpus, 308
- parser, resource description framework (RDF), 384
- parseval metric, 330
- parsing model, 307
- parsing system, 336
- parsing system, corpus-based, 336
- parsing system, probabilistic, 336
- parsing, corpus-based probabilistic, 307
- parsing, data-oriented (DOP), 307
- parsing, deterministic, 307
- parsing, linguistic, 327
- parsing, musical, 327
- parsing, probabilistic, 307
- parsing, rule-based deterministic, 307
- parsing, supervised, 335
- parsing, XML, 450, 463
- part-of-speech string, 334
- part-of-speech tagger, 334
- partially matched crossover (PMX), 888
- partialness, 365
- particle swarm, 883
- particle swarm optimization (PSO), 5, 38, 47, 969, 1029, 1041, 1046, 1051
- particle swarm optimization (PSO) algorithm, 1029, 1048, 1054, 1056, 1059
- particle swarm optimization (PSO), niching, 1038
- particle, explorer, 1044
- particle, quantum, 1046
- particle, swarm, 1030, 1034, 1037
- partition matrix, 614
- Parzen window estimator, 93
- Pascal language, 6
- patentable invention, 963
- pathogen, 1091, 1092
- pattern, 81
- pattern analysis, 708
- pattern class, 20
- pattern classification, 10, 16, 26, 28, 642, 644, 647, 648, 651, 662, 664
- pattern classifier, 43
- pattern detector module, 830
- pattern distance-based criteria, 96
- pattern matching, 113, 369
- pattern recognition, 20, 28, 81, 199, 530, 545, 715, 716, 719, 734, 801, 830, 835
- pattern recognition module, 814, 836
- pattern space, 650, 653
- pattern vector, 644
- pattern, spatio-temporal, 771, 780
- pattern, x-ray diffraction, 745
- payoff, 160, 167, 169, 170, 172, 173, 175
- payoff matrix, 162
- PC cluster, 801, 802
- PC, Pocket, 382, 398, 400, 402
- PCR, denaturation temperature gradient (DTG-PCR), 1068, 1080, 1081, 1083
- PCR, production-detection, 1071
- PCR, quantitative (Q-PCR), 1068
- peer-to-peer (P2P) architecture, 440, 447, 462

- peer-to-peer (P2P) computing, 440, 444
- penalized Cox model, 102
- penguin, genetically-altered, 250
- Penn Treebank, 309, 312, 330
- peptide, 1093, 1113
- perception, 42
- perception, fuzzy set, 599, 629, 634
- perception, human visual, 716
- perceptron, 11, 27, 28, 717
- perceptron, multi-layer (MLP), 27, 520, 717, 719, 864
- percolation theory, 159
- perfect model, 264
- performance density, 786
- performance index, 612, 614, 619, 631, 633
- performance measure, 173, 892, 905, 912
- performance metric, 431, 692, 701
- performance metric, multi-level, 702
- performance, network, 692
- performance, PSO algorithm, 1036
- Perlin's responsive face, 202
- permanent income hypothesis, 573
- permission, 244, 247, 265, 266, 272, 276, 280-284
- permutation order based crossover (POP), 903, 912, 914, 916, 918
- persistence, state, 446
- Persona, 207
- personal digital assistant (PDA), 382, 397, 402
- personal exploration, 1033
- personality, computer, 208
- personalized medicine, 1091, 1092
- Petri Net, colored (CPN), 42, 447, 453, 455
- phenotype, 1105
- pheromone, 37
- phone, cell, 382, 397
- phone, mobile, 382, 397
- phonetic typewriter, 117
- phrase structure tree, 309, 314
- phrase, idiomatic, 310
- phrase, noun (NP), 309, 310, 316
- phrase, prepositional (PP), 309
- phrase, verb (VP), 309, 310, 316
- physical entity, 267
- physical state, 267
- physics derivation, 320
- piecewise fuzzy membership function, 623
- piecewise linear transformation, 622
- pipe, 267, 274
- pipeline control safety verification, EVALPSN pipeline control, 266
- pipeline network, 270
- pipeline network, brewery, 267
- pipeline process, 270, 297, 299
- pipeline process control, 234
- pipeline process order control, 234
- pipeline process, brewery, 284
- pipeline safety property, 270
- pipeline valve control, 234
- pipeline valve control safety verification, 284, 299
- pipeline valve safety, 270
- pipeline, brewery, 265
- Pittsburgh approach, 667
- place, 455
- planned economy, 157
- platform, agent-based modeling (ABM), 410, 431
- plausibility, biological, 1044
- plurality voting, 44
- Pocket PC, 382, 398, 400, 402
- point mutation
 - genetic programming, 935
- point, cut, 885, 903, 912
- point, data, 131
- point, latent, 121, 123, 125, 126, 129, 131, 133, 136, 139
- point-neuron model, 773, 774, 776, 785
- poker, 970
- policy, chaining, 453
- policy, scheduling, 456, 462
- polychronization, 771
- polychronization, 780
- polymerase chain reaction (PCR), 1066, 1067, 1071, 1080
- polymerase, DNA, 1066
- population, 536, 538, 546, 555, 809
- population code, 766
- population diversity, 906
- population dynamics, 1097
- population learning, 519
- population size
 - genetic programming, 942

- population, agent, 156, 1101
- population, generation, 32
- population, genetic, 32
- population, sub-, 978
- population-based method, 1105
- population-based modeling, 410
- population-based search, 928
- portable digital assistant (PDA), 382, 397, 402
- portal, media grid, 695
- portal, service, 463
- portal, tourist information, 388
- portal, travel, 390
- portal, web, 696
- Poser, 210
- position update equation, 1034
- position, swarm particle, 1030
- position-based mutation, 890
- post-genome era, 1091
- power efficiency, 767, 768
- power grid, electricity, 441
- power grid, scale-free, 502
- power load management, 501
- pre-processing, data, 16, 19–21, 25, 26, 32, 33
- precision test, 401
- predicate calculus, 9
- predicate logic, 1071
- prediction, 852, 1099
- prediction error, 701
- prediction, bandwidth, 690, 691
- prediction, network bandwidth, 691
- prediction, time series, 532, 533
- prediction, traffic, 690
- prediction, video bandwidth, 691
- predictor, convergence, 503
- predictor, neural network, 692
- preference list, 1109, 1110
- prepositional phrase (PP), 309
- preprocessing, data, 81, 82, 700, 919
- price dynamics, 564
- price euphoria, 551
- price, commodity, 157
- price-volume relation, 573
- primer, 1068, 1080
- primitive set, 930
- principal component analysis (PCA), 16, 525, 527, 733
- principal component analysis (PCA), kernel, 734
- principal component analysis, nonlinear (NLPCA), 734
- principal component network, 721
- priority fuzzy membership function estimation, 607
- prisoner's dilemma (PD), 159, 161, 542
- prisoner's dilemma, iterated (IPD), 160, 859
- prisoner's dilemma, spatial (SPD), 160, 167, 173
- probabilistic cellular automata (PCA), 159, 164, 165, 168, 169
- probabilistic context-free grammar (PCFG), 308, 328, 329
- probabilistic inference, 23
- probabilistic parsing, 307
- probabilistic parsing system, 336
- probabilistic reasoning, 21
- probability density, 84
- probability density function (PDF), 84, 128, 130, 727
- probability distribution, tree, 312
- probability divergence-based criteria, 94
- probability, crossover, 656
- probability, fuzzy conditional, 651
- probability, mutation, 657, 664, 668
- probability, tree, 314
- probe, 421, 425, 426
- probe, agent, 426
- problem bin packing (BPP), 882
- problem constraint, 1035, 1049
- problem solution, exemplary, 322
- problem solver, immunity-based, 1106
- problem solving, 327
- problem solving, artificial, 1111
- problem solving, biological, 1111
- problem solving, derivational, 320
- problem solving, immunity-based, 1111, 1113
- problem space, 1030, 1034, 1035, 1049, 1050, 1056
- problem space search, 1035
- problem space, continuous, 1046
- problem space, quantized, 1047, 1049
- problem specification, 598
- problem, bin packing (BPP), 887, 892, 894, 899, 905, 914, 917

- problem, combinatorial, 1065, 1107, 1113
- problem, credit assignment, 1097
- problem, dynamic, 1040, 1046
- problem, elevator dispatch, 1080, 1082
- problem, equal piles, 895
- problem, frequency assignment, 896, 899
- problem, graph coloring (GCP), 882, 887, 892, 893, 897, 899, 905, 912, 917
- problem, grouping, 892
- problem, Hamiltonian path, 1066
- problem, multiply constrained, 920
- problem, NP-complete, 1065, 1071
- problem, NP-hard, 1065
- problem, optimization, 601, 626, 634
- problem, set partitioning (SPP), 881, 887, 892, 896, 898, 904, 906, 910, 913, 919, 920
- problem, stable marriage (SMP), 1106, 1107, 1113
- problem, static, 1040, 1046
- problem, timetabling, 882, 1049
- problem, travelling salesman (TSP), 731, 881
- problem-oriented formation, fuzzy set, 599
- procedural programming, 6
- process, 156, 267
- process control, 234, 300, 967
- process control, pipeline, 234
- process locking, 280, 283
- process order, 284, 294, 295, 297
- process order control, 234, 300
- process order control, pipeline, 234
- process order safety, 295
- process order safety verification, 298, 299
- process order safety verification system, 291
- process release, 282, 283
- process release control, 277
- process release safety control, 279
- process safety, 269
- process safety verification, 282–284
- process schedule, 269, 280, 282–284, 292, 297, 298
- process schedule chart, 269, 279
- process time chart, 286–288, 292
- process, affinity separation, 1068
- process, biochemical, 1066
- process, brewery pipeline, 284
- process, DNA, 1065
- process, pipeline, 270, 297, 299
- process, safety verification, 298
- process, sub-, 267
- processing
 - image, 965, 966
 - signal, 965, 966
- processing, parallel, 1065, 1070
- processing, real-time, 299
- processing, signal, 1100
- processing, sub-symbolic, 48
- processing, super-parallel, 1066
- processing, symbolic, 48
- processor, ARM, 786
- processor, embedded, 786
- processor, fascicle, 786
- processor, general-purpose, 789
- processor, micro-, 786
- processor, monitor, 786
- processor, multi-, 786
- processor, neural, 782
- product of experts, 126, 130, 131, 146
- product of experts, topographic (ToPoE), 121, 126, 140, 142, 143, 147
- product operator, 648
- production rule, if..then, 9
- production system, 1071
- production-detection PCR, 1071
- productivity, linguistic, 310
- profile of mood states (POMS), 200
- profile-aware eager scheduling (PAES), 457
- profiling agent, 450, 463, 471
- program, generally Horn (GHP), 239
- program, logic, 237, 238, 242
- program, paraconsistent annotated logic, 238
- program, stratified, 264
- programmable logic array (PLA), 34
- programmable logic device (PLD), 34
- programming language, conceptual graph (CG), 364
- programming language, object-oriented (OOP), 419

- programming, agent-oriented (AOP), 41
- programming, annotated logic, 233
- programming, evolutionary (EP), 32, 36, 853, 883
- programming, genetic (GP), 32, 36, 547, 548, 883
- programming, genetic network, 1072
- programming, logic, 233–235, 264, 307, 364
- programming, object-based, 41
- programming, object-oriented (OOP), 22, 41
- programming, procedural, 6
- projection, 131, 357, 358, 360
- projection function, 733
- projection index, 734
- projection operator, 361, 362
- projection, data, 732, 749
- projection, graph, 361
- projection, linear, 733
- proof layer, semantic web, 383
- proof tree, 319
- proof, DSD, 257
- proof, SD, 257
- property, pipeline safety, 270
- property, safety, 274, 295, 297
- proportional integrative and derivative (PID), controller evolution, 963
- proportionate selection, 552
- propositional logic, 9, 22
- propositional paraconsistent annotated logic, 235
- propositional symbol, 238
- Protégé, 392
- protein, 1091, 1092
- protein-based computing, 1105
- proteome, 1091
- protocol, 453
- protocol, agent interaction, 453
- protocol, communication, 440, 447
- protocol, streaming, 689, 691
- protocol, TCP/IP, 157
- prototyping, 806, 807
- provability relation, 249
- proximity, 732, 749
- proxy scheduler, 443
- pruning, 91, 535
- pruning, decision tree (DT), 17
- pruning, network, 29
- pseudo random number generator (RNG), 422, 431
- PSO algorithm performance, 1036
- PSO, niching, 1038
- psycholinguistics, 311
- pulsed artificial neural network (ANN), 27
- purification, affinity, 1081
- Python language, 422, 424, 425
- Q- α -FS, 96, 98
- QoS-aware media grid, 690
- QoS-aware media streaming, 690
- quadratic MI-based feature selection method (QMIFS), 98
- Quality-of-Service (QoS), 441, 689, 696
- quantitative finance, 518
- quantitative PCR (Q-PCR), 1068
- quantization, 112, 117, 132
- quantization error, 737
- quantization error, vector (VQE), 83
- quantization, learning vector (LVQ), 726
- quantization, vector (VQ), 715, 726, 732, 743
- quantized problem space, 1047, 1049
- quantum algorithm evolution, 963
- quantum computer
 - genetic programming, 964
- quantum computing (QC), 41
- quantum Fourier transform evolution, 964
- quantum particle, 1046
- quantum Turing Machine (QTM), 1084
- queue, wait, 1074
- Rényi entropy, 495, 502
- radial basis function (RBF), 29, 135, 524, 693, 742, 748
- radial basis function (RBF) neural network, 522, 528
- radial basis neural network (RBN), 520, 522, 528
- railway interlocking safety, 266
- random attack, 491
- random coil, 1080
- random number generator (RNG), 424, 425, 427, 429

- random number generator (RNG),
 - pseudo, 422, 431
- random sampling, 88
- random search, 1030
- random search engine, 98
- random walk, 565, 566
- randomizer module, 825
- randomly connected network, 491
- rank-based linear combination method,
 - 853, 855, 858
- rank-based selection scheme, 853
- rank-order code, 780, 782
- Rastrigin function, 1048, 1052, 1054
- rate code, 766
- rate, crossover, 891
- rate, damage, 168, 169, 173
- rate, failure, 168–170, 173
- rate, genetic operator, 942
- rate, recognition success, 1094
- rate, repair, 164, 168, 173, 174
- rate, repair success, 168, 1093
- rate, selfish repair, 169
- rational expectations equilibrium
 - (REE), 544–546, 550, 551, 562, 565
- rational expectations hypothesis (REH),
 - 549, 565, 566, 573
- reaction, antigen-antibody, 1096, 1106, 1107
- reaction, DNA, 1065
- reaction, enzyme, 1067
- real estate agent (REA), 203, 207
- real-time (RT), 387, 443, 767, 768, 785, 789, 811, 826, 835, 837, 838
- real-time multimedia, 689
- real-time processing, 299
- reasoning, 23, 363, 365, 388
- reasoning tool, knowledge conjunction,
 - 367, 370
- reasoning, approximate, 25
- reasoning, automated, 319, 374
- reasoning, case-based (CBR), 21
- reasoning, constrained, 21
- reasoning, default, 233
- reasoning, defeasible, 233, 243, 244, 247, 254
- reasoning, defeasible deontic, 275, 299
- reasoning, EVALPSN defeasible deontic,
 - 266, 274
- reasoning, fuzzy, 370, 644, 647, 648
- reasoning, knowledge-level, 369
- reasoning, non-monotonic, 233, 242
- reasoning, normative, 257
- reasoning, probabilistic, 21
- reasoning, temporal, 300
- receiver, global positioning system
 - (GPS), 401
- receptor, 159, 1101
- reciprocal matrix, 607
- recognition, 1101
- recognition accuracy, 82, 199
- recognition performance, 81
- recognition success rate, 1094
- recognition, adaptive, 1096, 1105
- recognition, networked, 1096, 1100, 1105
- recognition, pattern, 20, 28, 734
- recombination, 1105
- recombination operator, 537, 539, 853, 1102
- reconciliation, 629, 634
- reconfigurable silicon, 807
- recruit message, 491
- recurrent neural network, 26, 520, 523, 524, 693, 719, 851
- recurrent self-organizing map (RSOM),
 - 731, 747
- recurrent transition network
 - evolution, 957
- recursive agent simulation toolkit
 - (Repast), 569
- recursive least-square (RLS) algorithm,
 - 853
- recursive self-organizing map, 731
- reduced data set, 83
- reduction, dimensionality, 28
- redundancy, representational, 896, 905
- redundant data, 81
- references, 426
- referent, restrict, 356
- referential fuzzy set, 621
- reflection, 421, 423
- refractory period, 774, 775
- regression
 - numeric, 943
 - symbolic, 943, 944, 959
- regression, kernel, 735
- regression, symbolic, 547

- reinforcement learning, 571, 580, 720, 851
- relation, 352–354, 359, 364
- relation, (before-after) bf, 234, 284–288, 290, 293–295
- relation, conformity, 354
- relation, fuzzy, 629, 634
- relation, provability, 249
- relation, subsumption, 362
- relation, subtype, 354
- relation, subtype-supertype, 362
- relation, superiority, 253, 254, 256, 262, 275
- relational learning, 307
- relational learning, statistical, 319
- relational learning, stochastic, 324
- relational mapping, 629
- relationship, hierarchical, 382
- release, process, 282, 283
- reliability theory, 159, 160
- reliability threshold, 1043
- reliability, agent, 1098
- reliability, fitness, 1041, 1042, 1056
- remote procedure call (RPC), 395
- repair by copying, 159, 163, 1095
- repair rate, 164, 168, 173, 174
- repair rate, selfish, 169
- repair success rate, 168, 1093
- repair unit, 1093
- repair, mutual, 159, 160, 162, 1093
- repair, self-, 49, 156, 174
- repair, strategic, 171, 173
- repair, uniform, 171, 174
- repaired agent, 164, 168
- Repast, 413, 416, 417, 420–422, 424–426, 428–431
- replicator dynamics, 167
- representation
 - genetic programming, 929
 - prefix notation, 931
 - syntax tree, 929
 - tree-based in genetic programming, 931
- representation system, knowledge, 364
- representation, knowledge, 9, 374
- representational redundancy, 896, 905
- representative entropy (RE), 85
- representative entropy data reduction (REDR), 85, 88
- reproduction, 882, 884, 1101
- reproduction mechanism, 884
- reproduction operator, 31, 32, 550, 929, 946
- request-related scheduling, 462
- resource, 382
- resource allocation, 157, 173, 440, 443
- resource description framework (RDF), 373, 383, 384, 399
- resource description framework (RDF) graph, 399
- resource description framework (RDF) parser, 384
- resource description framework (RDF) triple, 383, 385
- resource description framework schema (RDFS), 383, 385, 399
- resource management, 440, 443, 445, 446, 471
- resource management framework, 462
- resource matrix, 471, 472
- resource sharing, 439, 442
- resource utilization, 471
- resource, computing, 441, 444
- resource, web, 383
- resource, web services (WSR), 452, 475
- resource, world wide web (WWW), 384
- resources, distributed, 691
- resources, network, 689
- responsibility, 117, 121, 122, 126, 128, 130, 131, 133, 146
- restrict referent, 356
- restrict type, 356
- retina-cortex mapping, 715
- retinotopic map, 114
- retinotopic model, 716
- retrieval system, location-aware tourist information, 403
- retrieval, knowledge, 373
- ribonucleic acid (RNA), 1068
- RLS algorithm, 856, 858
- RoboCup, 963
- robot, 802, 811, 812, 815, 823, 829
 - Elvis, 960
- robot control, 117
- robot control evolution, 963
- robot, autonomous, 797, 836
- robot, Darwin IV, 802
- robot, humanoid, 3

- robot-brain interface, 813
- robotic football, 970
- robustness, 1111
- root, 1078
- rough set, 21
- roulette wheel selection, 557, 884
- route planning, 403
- routing, 830, 837
- routing, multi-cast, 786
- routing, selfish, 156, 157
- rule, 359, 383
- rule base, fuzzy, 47
- rule evaluation, 383
- rule length, 662
- rule probability, 307
- rule set, 663
- rule weight, 647, 650
- rule, agent forecasting, 415
- rule, association, 22
- rule, canonical formation, 351, 353, 356, 358, 361, 366, 374
- rule, context-free, 307
- rule, copy, 355
- rule, decision, 541–543, 546
- rule, defeasible, 248, 249, 254–256, 262
- rule, fuzzy, 9, 22, 641–644, 646–655, 662–670, 672–674
- rule, generalized delta learning, 27
- rule, if...then production, 9, 22, 23
- rule, inheritance, 372
- rule, interval, 644, 649, 653–655
- rule, join, 355
- rule, learning, 851
- rule, market forecasting, 415
- rule, production, 22
- rule, simplify, 355
- rule, strict, 249, 275
- rule, syntax, 373
- rule, weight update, 29
- rule-based classifier, fuzzy, 641, 644, 647, 648, 652, 653, 655, 661, 663, 672, 674
- rule-based classifier, interval, 644, 653
- rule-based system, 23, 543, 859
- rule-based system, fuzzy, 627, 634, 641, 643, 648, 663, 673–675
- rule-based, deterministic parsing, 307
- rule-of-thumb, 1071
- runtime environment, 450, 463
- s-included before/after, 287
- safety control, process release, 279
- safety property, 274, 295, 297
- safety property, pipeline, 270
- safety verification, 234, 265
- safety verification cycle, 296
- safety verification framework, EVALPSN, 284
- safety verification process, 298
- safety verification system, bf-EVALPSN process order, 294, 295
- safety verification system, EVALPSN, 266, 267
- safety verification system, process order, 291
- safety verification, bf-EVALPSN, 297
- safety verification, EVALPSN, 266, 277, 279, 284
- safety verification, formal, 234, 266, 299, 300
- safety verification, pipeline valve control, 284, 299
- safety verification, process, 282–284
- safety verification, process order, 298, 299
- safety, pipeline valve, 270
- safety, process, 269
- safety, process order, 295
- safety, railway interlocking, 266
- Sammon mapping, 119, 732, 750
- Sammon stress, 733
- Santa Fe artificial stock market, 413
- Santa Fe Institute (SFI) economics, 544
- scalability, 82, 440, 448
- scalability, grid, 446
- scale-free graph, 490
- scale-free network, 158, 489–491, 494
- scale-free power grid, 502
- scale-free sensor grid, 491
- scale-free sensor network, 496, 502
- scale-free topology, 490
- scale-free tree graph, 489, 496, 498
- scaling, 749
- scaling, multidimensional (MDS), 118, 732
- schedule, process, 269, 279, 280, 282–284, 292, 297, 298
- scheduler, centralized, 440
- scheduler, discrete event, 422, 424, 425

- scheduler, local, 440, 446, 447
- scheduler, proxy, 443
- scheduler, super-local, 440
- scheduler, task, 443
- scheduling, 382, 386, 450, 453, 455, 459, 1071
- scheduling agent, 450, 459
- scheduling algorithm, 440
- scheduling mechanism, 471
- scheduling policy, 451, 456, 462
- scheduling policy, tagged, 456
- scheduling strategy, 445, 446
- scheduling strategy, super-local, 446
- scheduling, eager, 457
- scheduling, elevator, 1065
- scheduling, job, 440, 708
- scheduling, optimal, 1065
- scheduling, optimal elevator, 1074
- scheduling, profile-aware eager (PAES), 457
- scheduling, request-related, 462
- scheduling, task, 445
- scheduling, task-related, 455, 462
- scheduling, two-commit, 440
- scheduling, vehicle, 896
- schema, 543
- schema theory, 983
- schema, resource description framework (RDFS), 383, 385, 399
- Scholl and Klein (SK) data set, 914, 916, 919
- Schwefel function, 1049, 1054
- science, computational, 442, 444
- scientific computing, 420, 421, 444
- scientific instrument, 443
- scripting, 422, 426, 427
- SD-proof, 257
- search, 384, 1029, 1041, 1046
 - evolutionary
 - models, 983
 - population-based, 928
 - stochastic, 983
- search agent, 381
- search algorithm, tabu, 897
- search engine, 97, 381, 1101
- search engine, heuristic, 97
- search engine, non-deterministic, 98
- search engine, optimal, 97
- search engine, random, 98
- search engine, stochastic, 98
- search engine, weighting-based, 98
- search space, 32, 566, 983, 1045, 1110
 - genetic programming, 940
- search spaces
 - theory
 - genetic programming, 984
- search, associative, 782
- search, breadth-first, 10, 16
- search, compound, 98
- search, depth-first, 10, 16
- search, exhaustive, 10, 16, 881
- search, floating, 98
- search, frequency-based, 381
- search, fuzzy, 403
- search, heuristic, 10, 16
- search, keyword, 381
- search, local, 870, 1053, 1059
- search, meaning-based, 381
- search, problem space, 1035
- search, random, 1030
- search, semantic, 381
- search, sequential backward, 98
- search, sequential forward, 98
- search, tabu, 881, 913
- secondary emotions, 190
- seignorage, 553, 562
- selection
 - fitness-proportionate, 934
 - tournament, 934, 975
- selection operator, 556, 891
- selection probability, 884
- selection scheme, rank-based, 853
- selection theory, clonal, 1101
- selection, classifier, 44
- selection, proportionate, 552
- selection, roulette wheel, 557
- selection, tournament, 557
- self, 1092, 1093, 1097, 1102, 1105, 1106
- self maintenance, 156, 159
- self organization, 693
- self, non-, 159, 1092, 1093, 1097, 1102, 1105, 1113
- self-nonself discrimination, 1102
- self-organization, 156, 163, 448, 453, 486, 489, 719–721, 769
- self-organizing feature map (SOFM), 725

- self-organizing map (SOM), 84, 88, 111, 117, 131, 142, 146, 530, 532, 693, 715, 725
- self-organizing map (SOM), kernel, 740
- self-organizing map, adaptive subspace (ASSOM), 731
- self-organizing map, Bayesian, 740
- self-organizing map, circular, 748
- self-organizing map, parameterized (PSOM), 731
- self-organizing map, recurrent (RSOM), 731, 747
- self-organizing map, recursive, 731
- self-organizing map, stochastic (SSOM), 731
- self-organizing map, visualization induce (ViSOM), 731, 736, 749
- self-organizing mixture network (SOMN), 732, 738, 745
- self-organizing multi-agent network, 488
- self-organizing network, 724
- self-reactive agent, 1102, 1104
- self-repair, 49, 156, 174, 768
- self-repair network, 1093
- self-repairing network, 168, 169, 175
- selfish agent, 155, 156, 162, 163, 167, 168, 173, 175
- selfish agent, cooperative, 175
- selfish agent, networked, 158
- selfish repair rate, 169
- selfish routing, 156, 157
- selfish task allocation, 156
- selfishware, 156, 158, 174, 175
- semantic annotation, 311
- semantic gap, 18
- semantic network, 9, 22, 352, 1071
- semantic relationship, 352
- semantic search, 381
- semantic web, 372, 381, 383, 388, 399
- semantic web logic layer, 383
- semantic web proof layer, 383
- semantic web trust layer, 383
- semantics, 9, 233, 352, 381, 385, 597
- semantics, domain, 359
- semantics, fixpoint, 240
- semantics, fuzzy set, 598, 599, 621
- semantics, knowledge, 359
- semantics, language, 373
- semantics, stable model, 235, 242, 264
- semantics, weak, 373
- semantics, XML, 384
- semi-strict inference, 258
- sensibility constraint, 353, 359
- sensing, 81
- sensor, 443, 811, 817
 - soft, 959
- sensor array, 488
- sensor diagnosis, automobile engine, 1099
- sensor grid, 488, 489
- sensor grid, scale-free, 491
- sensor network, 158, 487–489
- sensor network, active, 488
- sensor network, scale-free, 496, 502
- sensor, air-flow, 1099
- sentence, 316
- sentence structure, 307
- sentence structure fragment, 307
- sentence, annotated, 309
- separate operation, 1066, 1070
- separate, controlled, 268, 272, 275, 276
- separation index, 617
- separation, affinity, 1083
- sequence data, DNA, 1091
- sequence, DNA, 1066, 1068, 1069, 1078, 1080
- sequence, DNA base, 1077
- sequential backward search, 98
- sequential forward search, 98
- serial exploration, 1037, 1038
- serum, blood, 1108
- server, 440
- server, Darwin streaming, 702
- server, iJADE Freewalker, 399
- server, media, 702
- server, streaming, 694
- server-based architecture, 447
- server-based grid, 440
- service container, 440
- Service Oriented Architecture Protocol (SOAP), 463
- service portal, 463
- service, grid, 695
- service, job management, 440
- service, web, 463
- service-oriented architecture (SOA), 446, 451
- service-oriented grid, 443

- service-oriented grid computing, 440
- services, web (WS), 439, 440, 446, 475
- set membership, 896
- set membership, fuzzy, 25
- set operator, fuzzy, 25
- set partitioning problem (SPP), 881, 887, 892, 896, 898, 904, 906, 910, 913, 919, 920
- set, fuzzy, 25, 598, 599, 605, 606, 616, 624, 625, 630
- set, rough, 21
- SFI artificial stock market, 545, 563, 564, 575
- shape-space model, 1097
- shell, expert system (ES), 24
- shoal of fish, 1029
- short-range attraction, 1039
- short-term memory (STM), 597, 764
- shortest derivation, 323, 329
- sigmoid, 778
- sigmoid function, 521, 698, 723, 804
- signal processing, 965, 966, 1100
- signal register, input (ISR), 810
- signal register, output (OSR), 810
- signal strength detector (SSD), 817, 829
- signal, neutralizing, 1105
- signaling speed, 803
- silicon, 768
- silicon compiler, 805
- silicon, reconfigurable, 807
- silicon-based computer, 1065, 1071
- similarity, 83, 92
- similarity index, 96
- similarity measure, symmetric, 730
- similarity metric, 747
- simple object access protocol (SOAP), 450
- simple protocol and RDF query language (SPARQL), 399
- simplicity, 311
- simplify rule, 355
- simulated annealing (SA), 854, 871, 881, 909, 936, 969
- simulated annealing, genetic (GSA), 906, 910, 916, 918, 919
- simulated emotion, 207, 208
- simulated synapse, 460, 462, 468
- simulation, Monte Carlo, 411, 423
- single nucleotide polymorphism (SNP), 1091
- single-strand DNA (ssDNA), 1068, 1081, 1083
- single-strand helix, DNA, 1066
- size, dynamic swarm, 1041
- size-fair crossover theory, 984
- size-fair crossover, genetic programming, 935
- skeptical model, 1098
- skyscraper, 1072
- small world phenomenon, 491
- Smalltalk language, 428
- smartGRID, 447
- smartGRID container, 450, 463
- smartGRID2, 462
- smartGRID2 container, 463
- smoothing spline, 735
- social actor, 212
- social cognitive theory, 213
- social collective memory, 1030
- social exploitation, 1033
- social interaction, 519
- social learning, 519, 556, 563, 567
- social network, 517, 578
- social sciences, agent-based, 517
- society, agent, 412
- soft computing, 4, 17, 1071
- soft sensor, 959
- software agent, 41, 386, 398, 409, 411, 578
- software bloat, 48
- software development, modular, 598
- software engineering (SE), 7, 18
- solution infeasibility, 896
- solution space, 27, 32
- solution, candidate, 883
- SOM Toolbox, 116, 121
- soma, 770, 784
- somatosensory map, 114
- sonar classification, 100
- sonar data set, 100
- sorting, non-dominated, 870
- soybean data set, 868
- space library, 424, 426, 429
- space, data, 121, 135
- space, feature, 112, 740–742
- space, input, 741, 742

- space, latent, 121, 123, 125, 126, 133, 135
- space, problem, 1030, 1034, 1035, 1049, 1050, 1056
- space, search, 32, 1045, 1110
- space, solution, 27, 32
- space, state, 27
- space, weight, 27
- spam email, 158, 174
- sparse distributed memories (SDM), 782
- spatial prisoner's dilemma (SPD), 160, 167, 173
- spatial prisoner's dilemma (SPD), deterministic, 167
- spatial prisoner's dilemma (SPD), stochastic, 167
- spatial strategy, totalistic, 167
- spatio-temporal, 1092
- spatio-temporal pattern, 771, 780
- specialization hierarchy, 358
- specialization, common, 357
- speciated evolutionary algorithm (EA), 859
- speciation, 859
- specification, concept, 382
- specification, problem, 598
- speculator, 568, 571
- speech interface, 202
- speech understanding, 308
- speech, emotive, 202
- speech, synthetic, 202
- speedup factor, 797, 798, 806, 807, 809, 815, 834, 837
- speedup factor, GPU, 981
- spike event, 766, 771, 772, 774, 775, 777-779, 781, 786, 788
- spike response model, 774
- spike-time-dependent plasticity (STDP), 777, 782
- spiking neural network (SPINN), 784, 801
- spiking neural network modeling, 785
- spiking neuron model, 782, 785
- spiking pattern, 777, 780
- SPINN emulation engine (SEE), 784, 785
- SpINNaker, 785, 786, 788, 789
- spline, smoothing, 735
- spyware, 174
- squared error, 492
- squashing function, 804
- stable marriage problem (SMP), 1106, 1107, 1113
- stable model, 255, 256
- stable model semantics, 235, 242, 264
- stable model, EVALPSN, 265
- stable model, VALPSN, 258
- standard generalized markup language (SGML), 384
- standard, web services (WS), 445, 447, 450, 451, 462, 463, 475
- StarLogo, 427
- state persistence, 440, 445, 446, 451
- state space, 27
- state, agent, 158
- state, logical, 267
- state, physical, 267
- state, valve control, 276
- static problem, 1040, 1046
- stationarity, 561, 563
- statistical analysis, 1100
- statistical learning theory, 528, 641
- statistical relational learning, 319
- statistics, 690
- steady state genetic algorithm (GA), 891
- Steve, 207
- stigmergy, 37
- stochastic approximation, 725
- stochastic context-free grammar, 331
- stochastic gradient descent, 728
- stochastic lexicalized grammar, 332
- stochastic logic program, 324
- stochastic relational learning, 324
- stochastic search, 983
- stochastic search engine, 98
- stochastic self-organizing map (SSOM), 731
- stochastic spatial prisoner's dilemma (SPD), 167
- stochastic tree substitution grammar (STSG), 328
- stock dividend, 413
- stock market, 966
- stock market, agent-based, 564
- stock market, artificial, 413, 545, 562, 571

- stopping condition, 892
- storage-reduction algorithm, 535
- strand, DNA, 1065, 1069, 1080, 1082–1084
- strategic repair, 171, 173
- strategy update cycle, 167
- strategy, evolutionarily stable (ESS), 174
- strategy, evolutionary, 1072
- strategy, scheduling, 446
- strategy, trading, 413
- stratification, 265
- stratified program, 264
- streaming protocol, 689, 691
- streaming server, 694
- streaming, audio, 690
- streaming, internet, 691
- streaming, media, 689, 690
- streaming, video, 690
- stress, Sammon, 733
- strict rule, 249, 275
- strike aircraft, 370
- string, character, 1066, 1070
- string, genetic, 32
- string, genetic algorithm (GA), 884
- strong negation, 233, 237, 264
- strongly-typed genetic programming theory, 984
- structural information, 382
- structure simplicity, 311
- structure, feature (FS), 366
- structure, knowledge, 366
- structure, network, 488
- stupid model, 426–429, 431
- sub-population, 978
- sub-process, 267
- sub-swarm, 1030, 1031, 1034, 1037, 1038, 1040, 1045, 1046
- sub-symbolic processing, 48
- subset method, 853, 858
- subset, disjunctive, 354
- subspace network, 734
- substitution operation, 309, 317, 320, 321, 324–326
- substitution, label, 309
- subsumption, 356, 358, 359, 363, 365
- subsumption architecture, 49
- subsumption relation, 362
- subsumption type hierarchy, 356
- subsumption, join, 374
- subsumption, type, 357, 361, 364, 369, 374
- subtree, 307
- subtree crossover, genetic programming, 934
- subtree mutation, genetic programming, 935
- subtree, depth-1, 307, 312
- subtree, treebank, 314
- subtype, 355, 360, 361
- subtype link, 355
- subtype relation, 354
- subtype-supertype relation, 362
- success rate, recognition, 1094
- success rate, repair, 1093
- sufficiency, genetic programming, 939
- Sugarscape, 412
- Sugeno model, 45
- sum of experts, 130
- sunspot effect, 573
- sunspot equilibrium, 557
- super-local grid architecture, 440, 444
- super-local scheduler, 440
- super-local scheduling strategy, 446
- super-mutation, 909
- super-parallel, 1069
- super-parallel processing, 1066
- supercomputer, 20, 439, 767, 768, 789, 800, 801, 979
 - genetic programming, 977
- superiority relation, 253, 254, 256, 262, 275
- supertype, 355, 359, 361
- supervised DOP, 334
- supervised learning, 112, 518, 692, 719, 740
- supervised network, 26
- supervised parsing, 335
- support, 611
- support vector machine (SVM), 5, 21, 30, 91, 99, 528, 740
- support vector machine based recursive feature elimination (SVM-RFE), 98
- survival-of-the-fittest, 31, 36, 519, 552, 556, 572, 886
- swap mutation, 890
- Swarm, 410, 413, 416, 417, 420–426, 428

- swarm, 36, 39, 883, 966
- swarm food source, 1031
- swarm intelligence (SI), 36
- swarm particle, 1030, 1034, 1037
- swarm particle acceleration, 1033
- swarm particle communication, 1030
- swarm particle momentum, 1031, 1032, 1036
- swarm particle neighbourhood, 1033, 1036, 1045
- swarm particle position, 1030
- swarm particle velocity, 1031, 1033, 1034, 1038, 1039, 1044
- swarm size, dynamic, 1041
- swarm update equation, 1046
- swarm, autonomous nanotechnology (ANTS), 37
- swarm, sub-, 1030, 1031, 1034, 1037, 1038, 1040, 1045, 1046
- symbol manipulation, 7
- symbol, propositional, 238
- symbolic artificial intelligence (AI), 718
- symbolic processing, 48
- symbolic regression, 547, 943, 944, 959
- symmetric similarity measure, 730
- synapse, 113, 770, 777, 801
- synapse, simulated, 460, 462, 468
- synaptic plasticity in spiking neural networks (SP²INN), 785
- synaptic weight, 720, 772, 774, 777, 781, 782, 784
- syntactic disambiguation, 311
- syntax rule, 373
- syntax tree, 929
- synthetic approach, 1091
- synthetic emotion, 207, 208
- synthetic speech, 202
- synthetic therapist, 213, 215
- system identification, 943
- system integration, 387
- system, adaptive, 851, 1105
- system, agent-based, 488
- system, artificial, 18, 49, 767, 768, 1092, 1111
- system, artificial immune (AIS), 40
- system, artificial neural, 769, 781, 782
- system, biological, 18, 157, 769, 781, 788, 1091, 1092, 1111
- system, biological system, 767, 768, 779
- system, chaotic, 1071
- system, cognition, 716
- system, complex, 17, 155, 157, 174, 175
- system, complex adaptive, 485
- system, complex dynamical, 769
- system, content management, 751
- system, context-aware, 388
- system, context-aware tourist guidance, 388
- system, corpus-based parsing, 336
- system, database management (DBMS), 22
- system, deduction, 360
- system, distributed, 387
- system, dynamical, 1097
- system, ecological, 1097
- system, economic, 157
- system, expert (ES), 21
- system, frame, 1071
- system, fuzzy, 4, 1071
- system, fuzzy inference (FIS), 25
- system, fuzzy-neuro, 46
- system, global positioning (GPS), 382, 388, 397, 401–403
- system, hybrid, 19
- system, iJADE tourist guidance, 401
- system, immune, 1091–1093, 1106, 1110, 1113
- system, immunity-based (IMBS), 159, 1091, 1092, 1095, 1102, 1113
- system, information (IS), 163, 173, 1093
- system, information processing, 715
- system, intelligent, 4, 17, 155, 158, 351, 769
- system, intelligent tourist guidance, 395
- system, knowledge representation, 364
- system, knowledge-based (KBS), 21
- system, location-aware mobile tourist guidance, 395
- system, location-aware tourist information retrieval, 403
- system, multi-agent (MAS), 23, 42, 386, 388, 488
- system, neural, 778, 782
- system, neural modeling, 771
- system, non-monotonic, 233
- system, ontology-based tourist guidance, 389
- system, parsing, 336

- system, probabilistic parsing, 336
- system, production, 1071
- system, rule-based, 23, 859
- system, tourist guidance, 388, 397, 402
- system, virtual exhibition (VES), 388
- system, vision, 782
- system-on-a-chip (SoC), 807
- systems approach, constructive, 1091

- t-test, 857, 858, 865
- T7 programming language, 985
- table, lookup (LUT), 35, 47
- tabu list, 1040
- tabu search, 881, 913
- tabu search algorithm, 897
- tag, 384
- tagged scheduling policy, 456
- Taguchi method, 47
- targeted attack, 491
- task decomposition, 440, 445–447, 451
- task scheduler, 443
- task scheduling, 445
- task-related scheduling, 455, 462
- task/service (TS) model, 451, 457, 462
- task/service description (TSD), 451
- Tauber-Wiener function, 521
- taxonomic hierarchy, 362, 363
- taxonomy, 359
- Tchebyshev distance, 616
- TCL, 422, 423, 427, 430
- TCP/IP protocol, 157
- TD-gammon, 49
- teaching agent, 204
- technology, grid, 689
- temperature gradient gel electrophoresis (TGGE), 1081
- template, DNA, 1068
- temporal gene expression clustering, 747
- temporal Kohonen map (TKM), 731, 747
- temporal reasoning, 300
- term, 382
- term, linguistic, 598
- termination condition, 657
- termination criteria, 536, 537
- termination criterion, 943, 944
- tertiary emotions, 190
- test chromosome, 808
- test tube, 1066, 1067, 1070
- test, China, 813
- test, precision, 401
- test, Turing, 8
- test, usability, 401
- testing data set, 857, 865
- text mining, 117, 726
- text, emotive, 200
- text-based agent, 210
- TGGE, 1084
- theme, 359, 364
- theorem proving, 319, 364
- theorem proving, automated, 3, 8
- theorem, central limit, 727
- theory
 - bloat
 - genetic programming, 982, 986
 - schema, 983
 - search spaces
 - genetic programming, 984
 - Turing complete programs, 984
 - theory of reasoned action/planned behavior, 213
- theory, clonal selection, 1101
- theory, conceptual graph (CGT), 360
- theory, defeasible, 248, 254–257, 262
- theory, defeasible deontic, 258, 262
- theory, features structure (FS), 365
- theory, game, 155–157, 159, 161, 174, 175
- theory, learning, 21
- theory, percolation, 159
- theory, reliability, 159, 160
- therapist, 212
- therapist, human, 213
- therapist, synthetic, 213, 215
- therapist-client relationship, 212, 215
- therapy, computer-mediated, 212
- threat aircraft, 371
- threat level, 370
- threshold, 773–775, 778, 784, 1050, 1053
- threshold detector module, 825
- threshold, reliability, 1043
- throughput, network, 691
- thymine, 1066
- tick, clock, 805, 808, 824, 827–829, 831
- Tierra, 410
- time chart, process, 292
- time series, 490

- time series analysis, 1099
- time series forecasting, 28, 535
- time series modeling, 28, 533
- time series prediction, 532, 533
- time series, chaotic, 532
- time series, financial, 519, 531, 532, 548, 552, 563
- time, wait, 1074
- time-to-live (TTL), 453, 459, 469
- timetabling problem, 882, 897, 899, 902, 917, 1049
- timetabling, examination, 894, 917
- TK Solver, 329
- token, 453
- tolerance, 1097
- tolerance, fault, 715
- tolerance, noise, 715
- tonotopic map, 114
- tool, knowledge representation, 353
- tool, ontology, 373
- top-down approach, 833
- ToPoE learning rule, 128
- ToPoE, twinned, 135
- topographic map, 111, 146
- topographic map, generative (GTM), 111, 121, 129, 143, 147
- topographic map, harmonic (HaToM), 121, 140, 142
- topographic product of experts (ToPoE), 121, 126, 140, 142, 143, 147
- topographic projection, 131
- topographical ordering, 715
- topographic, map, 721
- topological map, 725
- topology, 488, 1093
- topology preservation, 749
- topology preserving map, 111, 114, 121, 146, 715, 726
- topology product, 729
- topology, network, 488, 782
- topology, scale-free, 490
- totalistic spatial strategy, 167
- tourist domain, 397
- tourist guidance system, 388, 397, 402
- tourist guidance system, agent-based, 382
- tourist guidance system, context-aware, 382, 388
- tourist guidance system, iJADE, 401
- tourist guidance system, location-aware mobile, 395
- tourist guidance system, ontology-based, 389
- tourist guide, human, 402
- tourist information, 382, 402, 403
- tourist information center, 398
- tourist information center, iJADE, 399
- tourist information portal, 388
- tourist information retrieval system, location-aware, 403
- tourist information, customized, 403
- tourist information, location-aware, 402
- tournament selection, 557, 656, 661, 934, 975
- tracker, 447, 455, 459
- tradeoff, accuracy-complexity, 641, 643, 670, 672–674
- tradeoff, interpretability-accuracy, 674, 675
- tradeoff, interpretability-complexity, 643
- trading strategy, 413
- traffic modeling, 411
- traffic prediction, 690
- traffic volume, 490
- training data set, 534, 834, 855, 857, 864
- training data, input/output, 17, 26
- training exemplar, 26, 27, 29
- training time, network, 17, 28, 29
- transfer function input/output, 17, 521, 526, 770, 778
- transform, Fourier, 745
- transform, wavelet, 692
- transformation, Gelfond-Lifschitz, 242
- transfusion, blood, 1108
- transistor, 767, 833
- transition, 453, 457
- transitivity, 608, 609
- Transputer, 979
- transtheoretical model (TTM), 213–215
- trapezoidal membership function, 662
- travel guide website, 390
- travel ontology, 389, 390, 393, 399, 402
- travel portal, 390
- travel website, 382, 402
- travel website ontology, 382

- travel, Hong Kong, 390, 402
- travelling salesman problem (TSP), 37, 487, 731, 881, 969
- tree graph, scale-free, 489, 496, 498
- tree probability, 314
- tree probability distribution, 312
- tree structure, 307
- tree substitution grammar (TSG), 327
- tree substitution grammar, stochastic (STSG), 328
- tree, analysis, 314
- tree, dendritic, 776
- tree, derivation, 319, 323
- tree, knowledge discovery (KD), 102
- tree, labeled, 319
- tree, parse, 308, 314
- tree, phrase structure, 309, 314
- tree, proof, 319
- tree-based DOP model, 314
- treebank subtree, 314
- triangular fuzzy set, 612, 621, 623
- triple, resource description framework (RDF), 383, 385
- triplet, 916
- Tron, 970
- trust layer, semantic web, 383
- Turing complete genetic programming, 956
- Turing complete program theory, 984
- Turing Machine (TM), 1066
- Turing Machine, deterministic (DTM), 1084
- Turing Machine, quantum (QTM), 1084
- Turing test, 8, 962, 988
- twinned ToPoE, 135
- two point crossover, 885
- two-commit scheduling, 440
- two-period overlapping generations (OLG) model, 552, 556
- two-valued logic, 25
- type, 352, 359, 364
- type hierarchy, 353, 355, 359, 361, 363, 371, 374
- type hierarchy, subsumption, 356
- type subsumption, 357, 361, 364, 369, 374
- type, absurd, 359
- type, concept, 364
- type, feature structure (FS) , 355
- type, restrict, 356
- type, sub-, 355, 360, 361
- type, super-, 355, 359, 361
- type, universal, 359
- type-2 fuzzy set, 620
- typewriter, phonetic, 117
- typicality, 601
- U-matrix, 731, 736, 749
- ubiquitous computing, 1105
- UCI knowledge discovery in databases repository (UCI-KDD), 87, 99
- UCI machine learning (UCI-ML) repository, 136, 644, 669, 856, 868
- UML-DOP, 335
- unexploded ordnance (UXO), 812, 813, 835, 968
- unification, 356, 357, 362–366, 369, 374
- unification, conceptual graph (CG), 362, 364
- unification, graph, 367
- unification, ontology, 365
- uniform crossover, 656, 664
 - genetic programming, 935
- uniform noise, 135
- uniform repair, 171, 174
- uniform, order based crossover (UOBX), 903
- unify operator, 361
- unimodal fuzzy membership function, 610
- unit, connected, 1093
- unit, neighbor, 1093
- unit, repair, 1093
- universal approximator, 31, 521, 641
- universal logic gate, 764
- universal markup language (UML), 373
- universal resource locator (URL), 385
- universal type, 359
- universality, 766
- universe of discourse, 598, 603, 605, 610, 625
- universe, Herbrand, 239
- unrestricted DOP (U-DOP), 308, 333–336
- unsupervised learning, 112, 334, 518, 693, 715, 719, 731
- unsupervised network, 26

- unsupervised neural network, 530
- UNURAN, 427
- update cycle, strategy, 167
- upper bound, least (LUB), 358, 367
- usability test, 401
- user identification, 1105
- user preferences, 388, 403
- user profile, 388
- user-based fuzzy set membership estimation, 620
- user-centric membership function estimation, 599

- validation data set, 855, 857
- validity, conceptual graph (CG), 361
- VALPSN stable model, 258
- valve, 267, 271, 274, 280
- valve control state, 276
- valve, brewery pipeline, 265
- variable bit rate video (VBR), 691
- variable similarity-based criteria, 92
- variable, linguistic, 24
- vector annotated literal, 243, 244
- vector annotated literal, well (WVA), 244
- vector annotated literal, well extended (WEVA), 246
- vector annotated logic program with strong negation (VALPSN), 234, 243, 247, 255, 264
- vector annotation, 243, 271–274, 284, 293–295, 299
- vector annotation, paraconsistent, 300
- vector quantization (VQ), 715, 726, 732, 743
- vector quantization (VQ), Bayesian, 726
- vector quantization (VQ), error tolerant, 726
- vector quantization (VQ), noise tolerant, 731
- vector quantization (VQ), optimal, 727
- vector quantization error (VQE), 83
- vector, weight, 725
- vector-based individual-based modeling (IBM), 410
- vehicle scheduling, 896
- velocity, swarm particle, 1031, 1033, 1034, 1038, 1039, 1044
- verb phrase (VP), 309, 310, 316

- verification, 807
- verification, formal safety, 266
- verification, safety, 234, 265
- vertex degree distribution, 490
- vertical fuzzy membership function estimation, 605, 606
- very large-scale integrated circuit (VLSI), 803
- video bandwidth prediction, 691
- video compression, 744
- video streaming, 690
- video, variable bit rate (VBR), 691
- Virtex, 797, 805–807, 837
- virtual exhibition system (VES), 388
- virtual machine (VM), 420, 424
- Virtual Machine, Java (JVM), 424, 428, 431
- virtual organization (VO), 443
- virus, 1113
- virus, computer, 158, 174
- vision system, 782
- visualization, 112, 117, 131, 146
- visualization induce self-organizing map (ViSOM), 731, 736, 749
- visualization, data, 443, 715, 732, 736, 737, 749
- Viterbi optimization, 328
- vocabulary, 383, 384
- voice interface, 402
- voice recognition, emotional, 197
- von Neumann computer, 1071, 1084
- Voronoi cell, 727
- voting, majority, 44
- voting, mutual, 1098
- voting, plurality, 44
- voting-based scheme, 649

- wait queue, 1074
- wait time, 1074
- Wall Street Journal (WSJ) corpus, 328, 330–332, 334, 335
- watermark, image, 966
- Watson-Crick complement, 1069
- Watson-Crick complementarity, 1066, 1071
- wavelet, 532, 1105
- wavelet modeling, 692
- wavelet transform, 692
- waves, 1040, 1057, 1058

- waves of warm particles (WoSP), 1038, 1040, 1046, 1048, 1049, 1054, 1056, 1057
- weak semantics, 373
- web authoring, 373
- web ontology language (OWL), 373, 385, 388, 392, 399, 400
- web page, 381, 384
- web portal, 696
- web resource, 383
- web service, 463
- web service description language (WSDL), 452, 463, 465
- web services, 424
- web services (WS), 439, 440, 446
- web services (WS) resource, 475
- web services (WS) standards, 445, 447, 450, 451, 462, 463, 475
- web services resource (WSR), 452
- web services resource framework (WSRF), 446
- web services standard, 463
- web, semantic, 372, 381, 383, 388, 399
- website, 382
- website, travel, 382, 402
- website, travel guide related, 390
- weight normalization, 720
- weight space, 27
- weight update rule, 29
- weight vector, 725
- weight, artificial neural network (ANN), 23, 27, 28, 521, 851, 853
- weight, synaptic, 772, 774, 777, 781, 782, 784
- weighted average cluster diameter, 493
- weighting-based search engine, 98
- well extended vector annotated (WEVA) literal, 246
- well vector annotated (WVA) literal, 244
- well-formed graph, 354
- wet computing, 1065
- white box model, 641
- wide area network (LAN), 442
- wine data set, 136
- winner-take-all, 648
- winner-take-all scheme, 649
- winner-takes-all (WTA), 857, 863, 865, 871
- winner-takes-all (WTA) circuit, 820, 822, 823
- winter, evolutionary computation, 32
- winter, neural, 27, 32
- wireless ad hoc network, 157
- workstation, 440
- world wide web (WWW), 373, 381, 384
- world wide web (WWW) resource, 384
- world wide web consortium (W3C), 383
- worm, 158, 174
- wrapper method, 83
- x-ray diffraction pattern, 745
- Xilinx, 797
- XML parsing library, 450, 463
- XML semantics, 384
- zero intelligent (ZI) agent, 568
- ZI Plus (ZIP) agent, 568