LNCS 7244

Alberto Moraglio Sara Silva
Krzysztof Krawiec Penousal Machado
Carlos Cotta (Eds.)

Genetic Programming

15th European Conference, EuroGP 2012
Málaga, Spain, April 2012
Proceedings

Lecture Notes in Computer Science 7244

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK
Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Alfred Kobsa
University of California, Irvine, CA, USA
Friedemann Mattern
ETH Zurich, Switzerland
John C. Mitchell
Stanford University, CA, USA
Moni Naor
Weizmann Institute of Science, Rehovot, Israel
Oscar Nierstrasz
University of Bern, Switzerland
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
TU Dortmund University, Germany
Madhu Sudan
Microsoft Research, Cambridge, MA, USA
Demetri Terzopoulos
University of California, Los Angeles, CA, USA
Doug Tygar
University of California, Berkeley, CA, USA
Gerhard Weikum
Max Planck Institute for Informatics, Saarbruecken, Germany

Alberto Moraglio Sara Silva
Krzysztof Krawiec Penousal Machado
Carlos Cotta (Eds.)

Genetic Programming

15th European Conference, EuroGP 2012
Málaga, Spain, April 11-13, 2012
Proceedings

Springer

Volume Editors

Alberto Moraglio
University of Birmingham, School of Computer Science
Edgbaston, Birmingham B15 2TT, UK
E-mail: a.moraglio@cs.bham.ac.uk

Sara Silva
INESC-ID Lisboa, Rua Alves Redol 9, 1000-029 Lisboa, Portugal
E-mail: sara@kdbio.inesc-id.pt

Krzysztof Krawiec
Poznań University of Technology, Institute of Computing Science
Piotrowo 2, 60-965, Poznań, Poland
E-mail: krawiec@cs.put.poznan.pl

Penousal Machado
University of Coimbra, Department of Informatics Engineering
Pólo II - Pinhal de Marrocos, 3030-290 Coimbra, Portugal
E-mail: machado@dei.uc.pt

Carlos Cotta
Universidad de Málaga, ETSI Informática
Campus de Teatinos, 29071 Málaga, Spain
E-mail: ccottap@lcc.uma.es

Cover illustration:
"Chair No. 17" by The Painting Fool (www.thepaintingfool.com)

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-29138-8 e-ISBN 978-3-642-29139-5
DOI 10.1007/978-3-642-29139-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012934045

CR Subject Classification (1998): D.1, F.1, F.2, I.2, I.5, J.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

© Springer-Verlag Berlin Heidelberg 2012
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, re-use of illustrations, recitation, broadcasting, reproduction on microfilms 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. 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.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 15th European Conference on Genetic Programming (EuroGP) took place during April 11–13, 2012 in the historical city of Málaga, in southern Spain, a delightful place for the conference. Being the only conference exclusively devoted to genetic programming and the evolutionary generation of computer programs, EuroGP attracts scholars from all over the world.

The unique character of genetic programming has been recognized from its very beginning. Presently, with over 7000 articles in the online GP bibliography maintained by Bill Langdon, it is clearly a mature form of evolutionary computation. EuroGP has had an essential impact on the success of the field, by serving as an important forum for expressing new ideas, meeting, and starting up collaborations. Many are the success stories witnessed by the now 15 editions of EuroGP. To date, genetic programming is essentially the only approach that has demonstrated the ability to automatically generate and repair computer code in a wide variety of problem areas. It is also one of the leading methodologies that can be used to 'automate' science, helping researchers to find hidden complex models behind observed phenomena. Furthermore, genetic programming has been applied to many problems of practical significance, and has produced human-competitive solutions.

EuroGP 2012 received 46 submissions from 21 different countries across four continents. The papers underwent a rigorous double-blind peer-review process, each being reviewed by at least three members of the international Program Committee from 23 countries. The selection process resulted in this volume, with 18 papers accepted for oral presentation (39% acceptance rate) and five for poster presentation (50% global acceptance rate for talks and posters combined). The wide range of topics in this volume reflects the current state of research in the field, including different genres of genetic programming (tree-based, grammar-based, Cartesian), theory, novel operators, and applications.

Together with four other co-located evolutionary computation conferences, EvoCOP 2012, EvoBIO 2012, EvoMusArt 2012, and EvoApplications 2012, EuroGP 2012 was part of the Evo* 2012 event. This meeting could not have taken place without the help of many people.

First to be thanked is the great community of researchers and practitioners who contributed to the conference by both submitting their work and reviewing others' as part of the Program Committee. Their hard work, in evolutionary terms, provided both variation and selection, without which progress in the field would not be possible!

The papers were submitted, reviewed, and selected using the MyReview conference management software. We are sincerely grateful to Marc Schoenauer of INRIA, France, for his great assistance in providing, hosting, and managing the software.

We would like also to thank the local organizer Carlos Cotta and the CAESIUM group of the University of Málaga, and the following local sponsors of Evo*: University of Málaga, in particular the School of Computer Science and the School of Telecommunications, and respective Directors Professor José M. Troya and Professor Antonio Puerta, as well as the Málaga Convention Bureau.

We thank Penousal Machado of the University of Coimbra, assisted by Pedro Miguel Cruz, for creating and maintaining the official Evo* 2012 website, and Miguel Nicolau of University College Dublin, for refurbishing the software tools used to produce this volume.

We especially want to express our genuine gratitude to Jennifer Willies of the Institute for Informatics and Digital Innovation at Edinburgh Napier University, UK. Her dedicated and continued involvement in Evo* since 1998 has been and remains essential for building the image, status, and unique atmosphere of this series of events.

April 2012

Alberto Moraglio
Sara Silva
Krzysztof Krawiec
Penousal Machado
Carlos Cotta

Organization

Administrative details were handled by Jennifer Willies, Edinburgh Napier University, Institute for Informatics and Digital Innovation, Scotland, UK.

Organizing Committee

Program Co-chairs

Alberto Moraglio	University of Birmingham, UK
Sara Silva	INESC-ID Lisboa, Portugal

Publication Chair

Krzysztof Krawiec	Poznan University of Technology, Poland

Publicity Chair

Penousal Machado	University of Coimbra, Portugal

Local Chair

Carlos Cotta	University of Málaga, Spain

Program Committee

Alex Agapitos	University College Dublin, Ireland
Lee Altenberg	University of Hawaii at Manoa, USA
Lourdes Araujo	UNED, Spain
R. Muhammad Atif Azad	University of Limerick, Ireland
Wolfgang Banzhaf	Memorial University of Newfoundland, Canada
Mohamed Bahy Bader	University of Portsmouth, UK
Helio Barbosa	LNCC / UFJF, Brazil
Xavier Blasco	Universidad Politecnica de Valencia, Spain
Anthony Brabazon	University College Dublin, Ireland
Nicolas Bredeche	Université Paris-Sud XI / INRIA / CNRS, France
Stefano Cagnoni	University of Parma, Italy
Pierre Collet	LSIIT-FDBT, France
Ernesto Costa	University of Coimbra, Portugal
Luis Da Costa	Université Paris-Sud XI, France
Michael Defoin Platel	Rothamsted Research, UK

Antonio Della Cioppa	University of Salerno, Italy
Ian Dempsey	University College Dublin / Virtu Financial, Ireland
Stephen Dignum	University of Essex, UK
Federico Divina	Pablo de Olavide University, Spain
Marc Ebner	Ernst-Moritz-Arndt Universität Greifswald, Germany
Aniko Ekart	Aston University, UK
Anna Esparcia-Alcazar	S2 Grupo, Spain
Daryl Essam	University of New South Wales @ ADFA, Australia
Francisco Fernandez de Vega	Universidad de Extremadura, Spain
Gianluigi Folino	ICAR-CNR, Italy
James A. Foster	University of Idaho, USA
Steven Gustafson	GE Global Research, USA
Jin-Kao Hao	LERIA, University of Angers, France
Simon Harding	Memorial University of Newfoundland, Canada
Inman Harvey	University of Sussex, UK
Malcolm Heywood	Dalhousie University, Canada
David Jackson	University of Liverpool, UK
Colin Johnson	University of Kent, UK
Tatiana Kalganova	Brunel University, UK
Ahmed Kattan	Um Alqura University, Saudi Arabia
Graham Kendall	University of Nottingham, UK
Michael Korns	Korns Associates, USA
Jan Koutnik	IDSIA, Switzerland
Krzysztof Krawiec	Poznan University of Technology, Poland
Jiri Kubalik	Czech Technical University in Prague, Czech Republic
William B. Langdon	University College London, UK
Kwong Sak Leung	The Chinese University of Hong Kong
John Levine	University of Strathclyde, UK
Evelyne Lutton	INRIA, France
James McDermott	MIT, USA
Penousal Machado	University of Coimbra, Portugal
Bob McKay	Seoul National University, Korea
Nic McPhee	University of Minnesota Morris, USA
Jorn Mehnen	Cranfield University, UK
Julian Miller	University of York, UK
Alberto Moraglio	University of Birmingham, UK
Xuan Hoai Nguyen	Hanoi University, Vietnam
Miguel Nicolau	INRIA, France
Julio Cesar Nievola	Pontificia Universidade Catolica do Parana, Brazil

Michael O'Neill	University College Dublin, Ireland
Una-May O'Reilly	MIT, USA
Fernando Otero	University of Kent, UK
Ender Ozcan	University of Nottingham, UK
Andrew J. Parkes	University of Nottingham, UK
Clara Pizzuti	Institute for High Performance Computing and Networking, Italy
Gisele Pappa	Federal University of Minas Gerais, Brazil
Riccardo Poli	University of Essex, UK
Thomas Ray	University of Oklahoma, USA
Denis Robilliard	Université Lille Nord, France
Marc Schoenauer	INRIA, France
Lukas Sekanina	Brno University of Technology, Czech Republic
Yin Shan	Medicare Australia
Sara Silva	INESC-ID Lisboa, Portugal
Moshe Sipper	Ben-Gurion University, Israel
Alexei N. Skurikhin	Los Alamos National Laboratory, USA
Guido Smits	The Dow Chemical Company, USA
Terence Soule	University of Idaho, USA
Lee Spector	Hampshire College, USA
Ivan Tanev	Doshisha University, Japan
Ernesto Tarantino	ICAR-CNR, Italy
Jorge Tavares	University of Coimbra, Portugal
Theo Theodoridis	University of Essex, UK
Leonardo Trujillo	Instituto Tecnológico de Tijuana, Mexico
Leonardo Vanneschi	Universidade Nova de Lisboa, Portugal, and University of Milano-Bicocca, Italy
Sebastien Verel	University of Nice-Sophia Antipolis/CNRS, France
Katya Vladislavleva	University of Antwerp, Belgium
Man Leung Wong	Lingnan University, Hong Kong
Lidia Yamamoto	University of Strasbourg, France
Mengjie Zhang	Victoria University of Wellington, New Zealand

Table of Contents

Oral Presentations

Evolving High-Level Imperative Program Trees with Strongly Formed Genetic Programming 1
Tom Castle and Colin G. Johnson

Android Genetic Programming Framework 13
Alban Cotillon, Philip Valencia, and Raja Jurdak

Genetic Programming for Generalised Helicopter Hovering Control 25
Dimitris C. Dracopoulos and Dimitrios Effraimidis

Cartesian Genetic Programming for Memristive Logic Circuits 37
Gerard David Howard, Larry Bull, and Andrew Adamatzky

A New, Node-Focused Model for Genetic Programming 49
David Jackson

Medial Crossovers for Genetic Programming 61
Krzysztof Krawiec

Improving Face Detection 73
Penousal Machado, João Correia, and Juan Romero

Grammar Bias and Initialisation in Grammar Based Genetic Programming 85
Eoin Murphy, Erik Hemberg, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon

Improving Relevance Measures Using Genetic Programming 97
Kourosh Neshatian and Mengjie Zhang

An Investigation of Fitness Sharing with Semantic and Syntactic Distance Metrics 109
Quang Uy Nguyen, Xuan Hoai Nguyen, Michael O'Neill, and Alexandros Agapitos

Evolving Reusable Operation-Based Due-Date Assignment Models for Job Shop Scheduling with Genetic Programming 121
Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan

Evolving Interpolating Models of Net Ecosystem CO_2 Exchange Using Grammatical Evolution 134
Miguel Nicolau, Matthew Saunders, Michael O'Neill, Bruce Osborne, and Anthony Brabazon

Multi-Objective Ant Programming for Mining Classification Rules 146
Juan Luis Olmo, José Raúl Romero, and Sebastián Ventura

Matrix Analysis of Genetic Programming Mutation 158
Andrew J. Parkes, Ender Özcan, and Matthew R. Hyde

An Ecological Approach to Measuring Locality in Linear Genotype to Phenotype Maps 170
Tom Seaton, Julian F. Miller, and Tim Clarke

Coevolution in Cartesian Genetic Programming 182
Michaela Šikulová and Lukáš Sekanina

Evolutionary Design of Message Efficient Secrecy Amplification Protocols 194
Tobiáš Smolka, Petr Švenda, Lukáš Sekanina, and Vashek Matyáš

Automatic Design of Ant Algorithms with Grammatical Evolution 206
Jorge Tavares and Francisco B. Pereira

Posters

Random Sampling Technique for Overfitting Control in Genetic Programming 218
Ivo Gonçalves, Sara Silva, Joana B. Melo, and João M.B. Carreiras

Evolutionary Operator Self-adaptation with Diverse Operators 230
MinHyeok Kim, Robert Ian (Bob) McKay, Dong-Kyun Kim, and Xuan Hoai Nguyen

The Effect of Bloat on the Efficiency of Incremental Evolution of Simulated Snake-Like Robot 242
Ivan Tanev, Tüze Kuyucu, and Katsunori Shimohara

Bayesian Network Structure Learning from Limited Datasets through Graph Evolution 254
Alberto Paolo Tonda, Evelyne Lutton, Romain Reuillon, Giovanni Squillero, and Pierre-Henri Wuillemin

Efficient Phenotype Evaluation in Cartesian Genetic Programming 266
Zdeněk Vašíček and Karel Slaný

Author Index 279

Evolving High-Level Imperative Program Trees with Strongly Formed Genetic Programming

Tom Castle and Colin G. Johnson

School of Computing, University of Kent,
Canterbury, CT2 7NF, UK
{tc33,C.G.Johnson}@kent.ac.uk
http://www.kent.ac.uk

Abstract. We present a set of extensions to Montana's popular Strongly Typed Genetic Programming system that introduce constraints on the structure of program trees. It is demonstrated that these constraints can be used to evolve programs with a naturally imperative structure, using common high-level imperative language constructs such as loops. A set of three problems including factorial and the general even-n-parity problem are used to test the system. Experimental results are presented which show success rates and required computational effort that compare favourably against other systems on these problems, while providing support for this imperative structure.

Keywords: Genetic programming, Imperative programming, Loops.

1 Introduction

Evolving high-level imperative programs with genetic programming (GP) [1] is challenging. Programs are required to abide by complex structural rules just to be legal, and introducing commonly used constructs such as iteration bring further complexities to ensure that programs terminate and can be evaluated effectively. Because of these challenges, the GP research community has traditionally focused on the evolution of functional programs with a simple nested structure. However, the success of recent work using GP to perform tasks such as bug fixing commercial imperative programs [2], demonstrates the value of being able to evolve high-level imperative code. Modern imperative languages such as C/C++, Java and Python are widely used in the software development industry, so the evolution of code in this form is highly relevant for any application of GP in this area.

In this paper, we introduce Strongly Formed Genetic Programming (SFGP), a novel approach to constraining the structure of the program trees evolved with GP. SFGP extends previous work by Montana on Strongly Typed Genetic Programming [3] and combines it with constraints similar to those used by Koza in his work on *constrained syntactic structures* [1]. We demonstrate how the additional structural constraints of SFGP can be used to evolve high-level imperative code within a tree representation.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 1–12, 2012.
© Springer-Verlag Berlin Heidelberg 2012

The rest of this paper is organised as follows. Section 2 discusses some of the previous work on evolving imperative programs. Section 3 then gives a detailed description of how SFGP works. Section 4 describes the experiments that were conducted with the results presented and discussed in section 5. Finally, we conclude and summarise some of the research that this work leads on to in section 6.

2 Background

Some of the earliest attempts at evolving imperative programs were with a linear GP [4,5] approach. In linear GP, programs are comprised of a sequence of either machine code or interpreted higher- level instructions. The instructions read from and write to registers. The main incentive for using linear GP is faster execution speed [6], since the instructions can often be executed directly on hardware with little or no interpretation.

Grammar-guided GP is another area that has prompted work into imperative GP. Grammars provide a mechanism for constraining source code that is generated within a valid predefined syntax. O'Neill and Ryan [7] evolved multi-line C programs in their Grammatical Evolution (GE) system to solve the Santa Fe ant trail problem. While shown to be successful at solving this problem, the use of the standard GE algorithm in evolving more complex imperative programs is difficult since it uses context-free grammars which lack the expressiveness to describe semantic constraints. Other authors [8,9,10] have described extensions to GE that use context-sensitive grammars, but none go as far as using the extensions to evolve imperative programs. More recently, Langdon [11] has demonstrated how a grammar-guided GP system can be used to evolve compilable C++ CUDA kernels. Other grammar-based approaches have made use of context-sensitive grammars such as DCTG- GP [12] and LOGENPRO [13] which uses logic grammars to induce programs in a range of languages, including imperative C programs.

Montana proposed Strongly Typed Genetic Programming (STGP) [3] to allow nodes to define data-type constraints on their inputs that will be satisfied by the algorithm. This removes the need for the nodes to satisfy the closure property [1]. STGP does not provide any explicit mechanism for restricting the structure of program trees as grammars do, but the data-typing constraints do go some way to providing the restrictions necessary for supporting an imperative structure. Imperative programs are inherently composed of sequences of statements which are to be executed in order. Koza [1] used `ProgN` functions to achieve something similar. However, as McGaughran and Zhang [14] observe, this approach may provide a sequential ordering but it does so without a control structure that corresponds to any standard imperative constructs. They go on to present their own system based on a method of chaining statements.

3 Strongly Formed Genetic Programming

In this section we describe our method of evolving programs with a naturally high-level imperative structure. We refer to the method by the name Strongly

Formed Genetic Programming (SFGP), since it extends Montana's STGP, with additional constraints to the structure or form of the program trees.

Let us first clarify the limitation of STGP with an example. STGP requires all non-terminals to define the required data-type of each of their inputs. But, no limitation can be imposed on which terminal or non-terminal will provide that input. The child node may be any terminal or non-terminal of the correct data-type. Consider a type of node that performs the variable assignment operation. Any non-trivial imperative program is likely to require such a node. This `Assignment` node will require two inputs: a variable, and a value of the same data-type to assign to that variable. STGP can easily constrain these two inputs to be of the same data-type, but requires additional constraints to limit the first child to be a `Variable` node, rather than any other node of that data-type. The same problem exists with constraining code-blocks to contain only statements, and loop constructs that require a variable to update with an index.

SFGP resolves this problem by introducing an additional requirement of non-terminal nodes; that they define both a *data-type* and a *node- type* for each argument. The node-type property of an argument is defined as being the required terminals or non-terminals that can be a child node at this point, which when evaluated will return a value of the specified data- type. This therefore provides an explicit constraint on the shape and structure of program trees. In the case of an integer `Assignment` node, this can be used to state that the first child should not only be of an integer data-type, but should also specifically be a `Variable` node. These constraints must then be satisfied throughout the evolutionary process with minor modifications to the initialisation, mutation and crossover operators, as described in the following sections.

3.1 Initialisation

SFGP uses a grow initialisation procedure to construct random program trees, where each node is selected at random from those with a compatible data-type and node-type required by its parent (or the problem itself for the root node). Montana's grow initialisation operator made use of lookup tables to check whether a data-type is valid at some depth, but the addition of a second constraint excessively complicates these tables. The alternative is to allow the algorithm to backtrack when no valid nodes are possible for the required constraints. At each step, if no valid nodes are possible within the available depth, then the function returns an error, and if the construction of a subtree fails with an error then an alternative node is chosen and a new subtree generated at that point. The algorithm ensures that all program trees that are generated satisfy all data-type and node-type limitations and that each tree is within the *maximum-depth* parameter.

Pseudo-code for the grow initialisation algorithm is listed in Algorithm 1. The `GenerateTree` function is initially called with a `data_type` parameter that is the required return type for the problem and a `node_type` parameter which defines the node-type required for the root of the program tree. On all problems in this paper, a root node of `SubRoutine` is used, which is intended to model a

Algorithm 1. Initialisation procedure, where *dt*, *nt* and *depth* are the required data-type, node-type and maximum depth. The $filterNodes(S, dt, nt, depth)$ function is defined to return a set comprised of only those nodes in S with the given data-type and node-type, and with non-terminals removed if $depth = 0$.

```
 1: function GenerateTree(dt, nt, depth)
 2:     V ← filterNodes(S, dt, nt, depth)
 3:     while V not empty do
 4:         r ← removeRandom(V)
 5:         for i ← 0 to arity(r) do
 6:             dti ← required data-type for ith child
 7:             nti ← required node-type for ith child
 8:             subtree ← generateTree(dti, nti, depth − 1)
 9:             if subtree ≠ err then
10:                 attach subtree as ith child
11:             else
12:                 break and continue while
13:             end if
14:         end for
15:         return r                                  ▷ Valid subtree complete
16:     end while
17:     return err                                    ▷ No valid subtrees exist
18: end function
```

module of code such as a function or method. This means that all programs that are generated have the same basic imperative structure, shown in figure 1. The `SubRoutine` node requires two children: a `CodeBlock` with a void data-type and a `Variable` with the same data-type as the subroutine. Nodes with a void data-type do not return a value. When evaluated, a subroutine's code-block, which is a list of some predefined number of statements, is first executed and then the value of the variable is returned as the result of the subroutine.

3.2 Mutation

Our mutation operator employs the initialisation algorithm to grow new subtrees of the same data-type and node-type as an existing randomly selected node in a program tree. This node is then replaced with the newly generated subtree. The initialisation procedure is able to generate trees within a given maximum depth, so replacement subtrees are generated to be no deeper than the maximum depth parameter, minus the depth of the mutation point. Assuming the set of available nodes is unchanged, then it should always be possible to generate a legal replacement subtree for any existing node, but it is possible that the replacement is syntactically or semantically identical to the existing subtree. It is possible that this could lead to a high degree of neutral mutation if the syntax contains little variety.

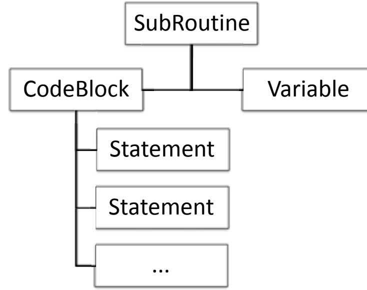


Fig. 1. The imperative structure of all program trees. In this paper, `CodeBlocks` with exactly 3 statement arguments were used. This was an arbitrary choice and an alternative size may produce different results. Evaluation executes each of the code-block's statements, before returning the value of the sub-routine's variable.

3.3 Crossover

The subtree crossover operator has been modified to maintain the node-type constraint while exchanging genetic material between two program trees. A node is selected at random in one of the programs. A second node is chosen at random from those nodes in the other program that are of the same data- type and node-type as the first node. The subtrees rooted at these two selected nodes are then exchanged. Those resultant child programs that have depths that exceed the maximum depth parameter are discarded.

3.4 Polymorphism

When implemented with an object-oriented approach, SFGP is able to support a simple form of polymorphism for both the data-type and node-type constraints. In figure 1 the `CodeBlock` node is shown to have a sequence of children with a `Statement` node-type. In an object-oriented system, this can be interpreted as any object that is an instance of the `Statement` class, or any sub-class. Nodes such as `Assignment`, `IfStatement` and `ForLoop` may then be implemented as sub-classes of `Statement` and may all appear in this position. In fact, it makes little sense to create a node of the type `Statement` itself, it is merely used to maintain the hierarchy of node-types. We refer to such node-types as *abstract* node-types. `Expression` is another abstract node-type that is used frequently. Data-type constraints can make use of the same polymorphic properties. If `Integer` and `Float` are both sub-classes of a class called `Number`, then either may appear where a required data-type of `Number` is specified. Note that the object-oriented approach we refer to here is a property of the implementation, rather than of the evolved programs, which are not themselves object-oriented.

3.5 Syntax

In this section we itemise the list of nodes that are used in the rest of this paper, along with the required data-type and node-types for their arguments.

- `SubRoutine` - requires a void `CodeBlock` and a `Variable` of the same data-type as the sub-routine. On evaluation, the code-block is evaluated and then the value of the variable is returned as the result. In all our experiments, `SubRoutine` is defined as the required root node-type.
- `CodeBlock` - is defined to require a fixed number of child nodes (3 used in all cases within this paper) of a `Statement` node-type and a void data-type. On evalutation, each child will be evaluated in sequence. `CodeBlock` has a void data-type and so does not return a value.

The following node-types are all subtypes of the abstract `Statement` node-type. All statements have a void data-type and so do not return any value.

- `Loop` - requires two children: an integer `Expression` and a `CodeBlock`. The expression is evaluated to provide a number of iterations to perform, and the code-block is evaluated the specified number of times. To maintain reasonable evaluation times, the number of iterations is capped at 100. No variables are manipulated by this loop construct.
- `ForLoop` - requires an integer `Variable`, which is updated with the index on each iteration from 1 to an upper bound; an integer `Expression` which is evaluated once to supply the upper bound (capped at 100 iterations) and a `CodeBlock` which is evaluated once per iteration.
- `ForEachLoop` - requires three children: a `Variable` of the element data-type, an `Expression` of some pre-defined array data-type and a `CodeBlock`. The code-block is evaluated once for each element of the array obtained by evaluating the expression argument. For each iteration, the value of the element is assigned to the variable.
- `IfStatement` - requires one boolean `Expression` and one `CodeBlock`. The code-block is conditionally evaluated only if the expression evaluates to true.
- `Assignment` - requires one `Variable` and one `Expression` input. Both inputs are required to have the same data- type specified on construction. On evaluation the expression is evaluated, and the result is assigned as the value of the variable.

The following node-types are all subtypes of the abstract `Expression` node-type. All expressions have non-void data-types.

- `Add`, `Subtract`, `Multiply` - require two integer `Expression` children each, with the integer result of the arithmetic operation returned.
- `And`, `Or`, `Not` - require two, two and one boolean `Expression` children respectively, with the boolean result returned.
- `Literal` - holds a fixed literal value of a given data-type.
- `Variable` - holds a value of a given data-type which may be modified (by assignment) throughout evaluation. The data-type of a variable is fixed at construction.

4 Experiments

A series of experiments were carried out to demonstrate SFGP's ability to generate programs composed of standard programming constructs, within the imperative structure enfored by our constraints. All experiments were carried out using the EpochX evolutionary framework [15], with our own extension in which the representation and operators were implemented as described in section 3. 500 runs were performed on each of three problems: factorial, Fibonacci and even-n-parity. These problems were chosen because they require the use of essential programming constructs such as loops and arrays and have also been used numerous times previously in the literature [16,17,18]. A maximum of 50 generations and a population of 500 were used on all problems. The subtree crossover and subtree mutation operators were chosen from with probabilities 0.9 and 0.1 respectively and tournament selection was used with a tournament size of 7. All other control parameters used are outlined in tables 1, 2 and 3.

4.1 Factorial

The program to be evolved here is an implementation of the factorial function. One input is provided, which is the integer variable `i`, where the ith element of the sequence is the expected result. The first 20 elements of the sequence were used to evaluate the quality of solutions, with a normalised sum of the error used as an individual's fitness score. The fitness function is defined in (1), where n is the size of the training set, i is the ith training case, $f(i)$ is the correct result for training case i, and $g(i)$ is the estimated result for training case i returned by the program under evaluation. Each individual which successfully handles all training inputs is tested for generalisation using a test set consisting of elements 21 to 50 of the sequence.

$$Fitness = \sum_{i=0}^{n} \frac{|f(i) - g(i)|}{|f(i)| + |g(i)|} \tag{1}$$

Table 1. Parameter tableau for the factorial problem

Root data-type:	`Integer`
Root node-type:	`SubRoutine`
Max. depth:	6
Non-terminals:	`SubRoutine`, `CodeBlock`, `ForLoop`, `Assignment`, `Add`, `Subtract`, `Multiply`
Terminals:	i, $loopVar$[1], 1 (integer `Literal`)

[1] The additional input *loopVar* is an integer variable, provided specifically for use by the `ForLoop` construct to contain the iteration index. Its initial value is 0.

4.2 Fibonacci

The Fibonacci problem was posed in a similar form as factorial, with an integer variable input `i` and an expected output which is the ith element of the Fibonacci sequence. Two further inputs were also provided in the form of variables containing the value of the first two elements of the sequence; 0 and 1. The same function (1) was also used to determine an individual's fitness, with the training inputs comprised of the first 20 elements of the Fibonacci sequence. A test set made up of elements 21 to 50 of the sequence were used to test the generalisation of successful programs.

Table 2. Parameter tableau for the Fibonacci problem

Root data-type:	`Integer`
Root node-type:	`SubRoutine`
Max. depth:	6
Non-terminals:	`SubRoutine`, `CodeBlock`, `Loop`, `Assignment`, `Add`, `Subtract`
Terminals:	i, $i0$, $i1$

4.3 Even-n-Parity

The boolean parity problems are widely used as a benchmark task in the GP literature [1,5,19]. However, they have only occasionally been tackled in the general form; for all values of `n` [18,20]. A program which successfully solves the even-n-parity problem, must receive as input an array of booleans, arr, of any length and must return a boolean `true` value if an even number of the elements are `true`, otherwise it must return `false`. All possible inputs to the 3-bit even-parity problem were used as the training data, as used by Wong and Leung [18]. The fitness of an individual was then a simple count of how many of the 8 inputs were incorrectly classified. A test set consisting of all possible input arrays of lengths 4 to 10 was used to test the generalisation of solutions that successfully solved the training cases.

Table 3. Parameter tableau for the even-n parity problem

Root data-type:	`Boolean`
Root node-type:	`SubRoutine`
Max. depth:	8
Non-terminals:	`SubRoutine`, `CodeBlock`, `ForEachLoop`, `IfStatement`, `Assignment`, `And`, `Or`, `Not`
Terminals:	arr, $boolVar1$, $boolVar2$[2]

[2] *boolVar1* and *boolVar2* are boolean variables for use by the `ForEachLoop` and root `SubRoutine` constructs. Their initial values were arbitrarily set as `false`.

5 Results

Table 4 lists a summary of the results, with the success rates and generalisability of the solutions that were discovered in the experiments. The table also describes the computational effort that was required to solve each problem, with the related performance curves displayed in figures 2a, 2b and 2c. The required computational effort was calculated in the manner of Koza [1] to be the number of individuals that must be processed to guarentee a solution with 99% confidence.

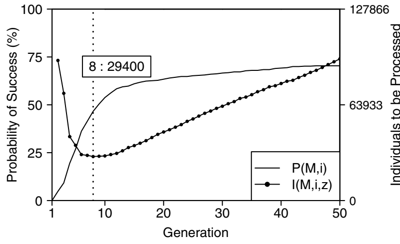
Previous attempts at evolving recursive structures that can generate the Fibonacci sequence include Harding et al [16], who used Self-Modifying Cartesian GP to generate both the first 12 and first 50 elements of the sequence with success rates up to 90.8% and up to 94.5% of those able to generalise to 74 elements of the sequence. In [21], a Linear GP system was used to achieve success rates up to 92%, and which generalised in 78% of cases. Other approaches have been less successful. Agapitos and Lucas used Object Oriented GP[17] to get success rates up to 25% on the first 10 elements of the sequence, but required a minimum computational effort of 2 million. Their success rates were slightly improved on the factorial problem, on which they report a 74% success rate and a minimum effort of 600,000. However, they note that their approach, which relies on Java's Reflection mechanism, is computationally expensive.

The even parity problems are considered to be difficult problems for GP to solve [20]. Koza's experiments required 1,276,000 individuals to be processed to yield a solution to just the 4-bit version of the problem and was unable to solve the problem with any higher number of bits without the use of automatic functions. In contrast, SFGP produced potential solutions with just 30,500 individuals processed, 81.4% of which were able to solve the general form of the problem including the 4-bit version. Other research has tackled the general even-n-parity problem. Agapitos and Lucas[17] required 680,000 individuals to be processed, where they used all the even-2-parity and even-3-parity problems as training data. In [18], Wong and Leung used just the even-3-parity problems as training inputs and reported their minimum computational effort as 220,000.

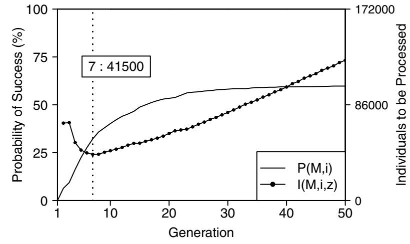
The primary reason for SFGP's greater performance on even-n-parity is likely to be due to the use of the `ForEachLoop` node, which encapsulates the necessary

Table 4. Results summary, where *Train%* is the proportion of runs that produced at least one solution for all training cases and *Test%* is the proportion that produced a solution that generalised to solve all test cases. The confidence interval for the computational effort is calculated using the Wilson 'score' method [22]. The *Evals* column shows the number of program evaluations required to find a solution, which is a product of the effort and number of training cases.

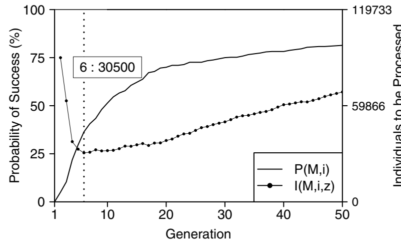
	Train %	Test %	Effort	95% CI	Evals
Factorial	70.8	70.6	29,400	25,800 - 33,500	588,000
Fibonacci	61.6	59.8	41,500	35,600 - 48,500	830,000
Even-n-Parity	91.6	81.4	30,500	26,400 - 35,400	244,000



(a) Factorial



(b) Fibonacci



(c) Even-n-parity

Fig. 2. Performance curves, where $P(M, i)$ is the success rate and $I(M, i, z)$ is the number of individuals to process to find a solution with 99% confidence

behaviour of performing an operation on each element of the array. The other studies mentioned relied on complex recursive structures developing through evolution.

5.1 Example Solution

As an example, one typical solution to the factorial problem that was found by SFGP is displayed below:

```
public long getFactorial(long i) {
    loopVar = loopVar;
    long x = i;
    loopVar = 1L;
    for (long y = 1L; y <= x; x++, loopVar = x) {
        loopVar = (loopVar * i);
        i = loopVar;
        loopVar = loopVar;
    }
    i = i;
    return i;
}
```

This example program is expressed using Java syntax, but the abstract syntax trees generated by SFGP can be interpreted as programs in the syntax of any

programming language that supports the imperative constructs used. This program has not undergone any post-processing, and contains statements such as $i = i$ which will have no impact on the result. These could easily be identified and removed by static analysis tools. Note that the loop structure contains the necessary infrastructure to ensure the index variable is updated but the bounds remain immutable, as defined by the `ForLoop` node that it represents.

6 Conclusions

In the course of this paper, we have introduced Strongly Formed Genetic Programming (SFGP) and demonstrated how it can be used to constrain the structure of program trees. In particular, we have shown that it is able to constrain evolved program trees to a more natural high-level imperative structure and make use of some standard imperative language constructs such as loops. The program trees that are evolved using this system may be easily expressed in the syntax of modern imperative programming languages. The results of using this imperative structure, that have been presented, compare very favourably with existing systems on these problems.

One current limitation with SFGP, that we would like to address in future work, is the lack of support for generic functions. Other possible future work includes investigating the impact of the arbitrary settings that have been used in this paper, such as the iteration bound on loop constructs and the fixed number of statements in a code-block. Support for imperative concepts such as variable declarations and recursion could also be of some value.

References

1. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
2. Weimer, W., Nguyen, T., Le Goues, C., Forrest, S.: Automatically finding patches using genetic programming. In: Fickas, S., et al. (eds.) ICSE 2009, Vancouver, pp. 364–374 (2009)
3. Montana, D.J.: Strongly typed genetic programming. Evolutionary Computation 3, 199–230 (1995)
4. Nordin, P.: A compiling genetic programming system that directly manipulates the machine code. In: Kinnear Jr., K.E. (ed.) Advances in Genetic Programming, pp. 311–331. MIT Press (1994)
5. Brameier, M., Banzhaf, W.: Linear Genetic Programming. Number XVI in Genetic and Evolutionary Computation. Springer, Heidelberg (2007)
6. Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programing. Lulu.com (2008)
7. O'Neill, M., Ryan, C.: Evolving Multi-line Compilable C Programs. In: Langdon, W.B., Fogarty, T.C., Nordin, P., Poli, R. (eds.) EuroGP 1999. LNCS, vol. 1598, pp. 83–92. Springer, Heidelberg (1999)
8. Cleary, R., O'Neill, M.: An Attribute Grammar Decoder for the 01 MultiConstrained Knapsack Problem. In: Raidl, G.R., Gottlieb, J. (eds.) EvoCOP 2005. LNCS, vol. 3448, pp. 34–45. Springer, Heidelberg (2005)

9. de la Cruz Echeandía, M., de la Puente, A.O., Alfonseca, M.: Attribute Grammar Evolution. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2005, Part II. LNCS, vol. 3562, pp. 182–191. Springer, Heidelberg (2005)
10. Ortega, A., de la Cruz, M., Alfonseca, M.: Christiansen grammar evolution: Grammatical evolution with semantics. IEEE Transactions on Evolutionary Computation 11, 77–90 (2007)
11. Langdon, W.B., Harman, M.: Evolving a CUDA kernel from an nVidia template. In: Sobrevilla, P., et al. (eds.) 2010 IEEE World Congress on Computational Intelligence, Barcelona, pp. 2376–2383. IEEE Press (2010)
12. Ross, B.J.: Logic-based genetic programming with definite clause translation grammars. In: Banzhaf, W., Daida, J.M., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M.J., Smith, R.E. (eds.) GECCO 1999, Orlando, vol. 2, p. 1236. Morgan Kaufmann (1999)
13. Wong, M.L., Leung, K.S.: Combining genetic programming and inductive logic programming using logic grammars. In: 1995 IEEE Conference on Evolutionary Computation, vol. 2, Perth, pp. 733–736. IEEE Press (1995)
14. McGaughran, D., Zhang, M.: Evolving more representative programs with genetic programming. International Journal of Software Engineering and Knowledge Engineering 19, 1–22 (2009)
15. Castle, T., Beadle, L.: Epochx: genetic programming software for research (2007), http://www.epochx.org
16. Harding, S., Miller, J.F., Banzhaf, W.: Self Modifying Cartesian Genetic Programming: Fibonacci, Squares, Regression and Summing. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 133–144. Springer, Heidelberg (2009)
17. Agapitos, A., Lucas, S.: Learning Recursive Functions with Object Oriented Genetic Programming. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 166–177. Springer, Heidelberg (2006)
18. Wong, M.L., Leung, K.S.: Evolving recursive functions for the even-parity problem using genetic programming. In: Angeline, P.J., Kinnear Jr., K.E. (eds.) Advances in Genetic Programming, vol. 2, pp. 221–240. MIT Press, Cambridge (1996)
19. Castle, T., Johnson, C.G.: Positional Effect of Crossover and Mutation in Grammatical Evolution. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 26–37. Springer, Heidelberg (2010)
20. Harding, S., Miller, J.F., Banzhaf, W.: Self modifying cartesian genetic programming: Parity. In: Tyrrell, A., et al. (eds.) 2009 IEEE Congress on Evolutionary Computation, Trondheim, pp. 285–292. IEEE Press (2009)
21. Wilson, G., Heywood, M.: Learning recursive programs with cooperative coevolution of genetic code mapping and genotype. In: Thierens, D., et al. (eds.) GECCO 2007, London, vol. 1, pp. 1053–1061. ACM Press (2007)
22. Walker, M., Edwards, H., Messom, C.: Confidence Intervals for Computational Effort Comparisons. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 23–32. Springer, Heidelberg (2007)

Android Genetic Programming Framework

Alban Cotillon, Philip Valencia, and Raja Jurdak*

Autonomous Systems Laboratory
CSIRO ICT Centre, Brisbane, Australia
alban.cotillon@insalien.org,
{philip.valencia,raja.jurdak}@csiro.au

Abstract. Personalisation in smart phones requires adaptability to dynamic context based on application usage and sensor inputs. Current personalisation approaches do not provide sufficient adaptability to dynamic and unexpected context. This paper introduces the Android Genetic Programming Framework (AGP) as a personalisation method for smart phones. AGP considers the specific design challenges of smart phones, such as resource limitation and constrained programming environments. We demonstrate AGP's utility through empirical experiments on two applications: a news reader application and an energy efficient localisation application. Results show that AGP successfully adapts application behaviour to user context.

Keywords: Genetic programming, Embedded, Smartphone.

1 Introduction

Smartphones have experienced exponential growth in recent years. These phones embed a growing diversity of sensors, such as gyroscope, accelerometer, Global Positioning System (GPS), and cameras, with broad applicability in areas such as urban sensing or environmental monitoring. Coupled with diverse user profiles [1], this provides significant user personalization opportunities, such as location-based and usage-based services, but it also involves significant challenges in adaptation to new or unexpected context.

Most smartphone algorithms, aiming at either data-centric [4] or user-centric personalization [2], are based on static or rule-based approaches. However, personalization increasingly depends on contextual information and user inputs [3]. Both are subject to dynamic changes, which motivates the use of methods that can not only adapt to expected changes or behaviors, but can also learn how to deal with unexpected changes in context.

Online learning is well-suited for smart phone personalization. In particular, online genetic programming supplies common basic constructs for a smart phone application that can evolve over time according to individual user preferences. This paper introduces Android Genetic Programming Framework (AGP)

* Cotillon is with the Autonomous Systems Lab, CSIRO ICT Centre and also with INSA Lyon. Valencia and Jurdak are with the Autonomous Systems Lab, CSIRO ICT Centre, and also with the School of ITEE, University of Queensland.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 13–24, 2012.
© Springer-Verlag Berlin Heidelberg 2012

for the mobile Operating System (OS), Android. We have chosen the Android platform as it is mainly open-source. As far as we know, this is the first genetic programming solution available on smartphones. The AGP framework can deal with dynamic fitness functions, providing a context-specific solution. Our goal is to demonstrate the flexibility of our new platform and how it can solve multi-objective problems in a dynamic environment.

2 Related Work

Several previous works in Genetic Programming have focused on architectural issues. Whereas some solutions provide generic frameworks for Evolutionary Computation problems [8,9,6], others propose application-specific solutions. Ismail et al. [10] describe a GP framework for extracting a mathematic formula needed for Fingerprint Matching, whereas, in Pattern Recognition [11], the authors focus on a genetic programming framework for content-based image retrieval. In Learning to Advertise [14], Lacerda et al. introduce a framework for associating ads with web pages based on GP. Valencia et al. [12] study genetic programming for Wireless Sensor Networks and propose the In Situ Distributed Genetic Programming (IDGP) framework. DGPF [15] brings utilities for Master/Slave, Peer-to-Peer, and P2P/MS hybrid distributed search execution. P-Cage [22] introduces and evaluates a complete framework for the execution of genetic programs in a P2P environment. It shows the relevance of using P2P networks scalability to counteract computation limitations. GA implementation has already been done on portable devices such as the Nokia N73 [7]. Its reliance on Python requires the user to install Python Runtime and various libraries whereas AGP can be used off-the-shelf without any additional module.

Design patterns describe the interaction between groups of classes or objects. They concentrate on specific concerns for implementing source code to support program organization. When they are well integrated into a framework, they ensure the goals of extensibility and reuse. Lenaerts and Manderick [13] discuss the construction of an object-oriented Genetic Programming framework using on design patterns to increase flexibility and reusability. McPhee et al. [8] extend the latter to Evolutionary Computation (EC). As the problem to solve becomes wider, it leads to a more abstract set of classes. Based on those works, Ventura et al. introduced JCLEC [6], a Java Framework for evolutionary computation. They present a layered architecture and provide a GUI for EC. This paper similarly uses Java for a genetic programming framework, albeit for a more resource constrained smart phone platform.

3 AGP: Android Genetic Programming Framework

3.1 Motivation

Mobile phone users have always tried to customise their devices, for instance through personalised ring tones. The emergence of smartphones takes the personalization possibilities to a new level. First of all, smartphones have access to

a huge diversity of Internet data which can be augmented with sensor-based context information. Secondly, the higher computing performance of smart phones enables developers to create novel applications. The combination of processing power, content accessibility and context awareness opens new opportunities for personalization. However, personalisation mechanisms have been slow to respond to these opportunities, relying hugely on static or rule-based algorithms. These approaches suffer in adaptability to unexpected changes, which requires a shift in personalisation methods.

Online genetic programming can evolve over time and is able to gather data from different sources. Because it continuously assesses new application configurations, genetic programming can improve the user experience by incorporating data from multiple sensors to tailor application performance to user preferences.

3.2 Design Challenges

Android is an operating system for mobile devices developed and maintained by Google and the open-source community. It provides an API which allows programmers to develop applications using the Java language. We used this particular API to implement the AGP framework. Although several other GP frameworks are available for the Java platform, none is suitable for Android because Android replaces several subsets of Java class libraries from the JavaSE with its own new classes. Moreover, developing a GP framework for a mobile OS brings some challenges that are discussed in this section.

User Interaction. GP frameworks designed for desktop machines focus on solving of complex problems offline. On the other hand, Android smartphones provide an opportunity for online learning of user-specific context to improve user experience. These devices provide increasingly sophisticated applications that can be customised to user preferences. To allow the capture of this diversity richness, the AGP framework directly accesses the Android API functions to learn from the user context without requiring a dedicated scripting language. As for security, the developer has to fill the application manifest file with the permissions required by the application, with Android being a privilege-separated operating system.

Limited Resources. The small form factor of smartphones brings computation and energy constraints, which restrict GP usage and require careful design of the GP infrastructure. Developers must carefully use AGP as it shouldn't contribute to undesirable user experiences, such as quick battery depletion. For instance, AGP can perform costly computation when energy resources are ample, and minimise learning when the user has high demand for the phone's resources or when the battery is running flat.

Battery level is easy to obtain through the Android API, which uses a coarse-grained scale from 0 to 100. However, real battery depletion can't be assessed from this battery level as the depletion rule differs from one device to another. We

Fig. 1. Interpreter package organization

recommend to track a system file used by Android to record live consumption. This file is available on most of the devices.

To validate our framework, we demonstrate AGP with applications on two different generations of Android devices: the early HTC Magic running Android 1.6; and the more capable Nexus S, embedding a dual-core processor with the recent Android Gingerbread (version 2.3).

Services and Intent. GP needs long-running background operations, which we implement using services under Android. An Android service is a component that can run in the background even when the user is not interacting with the device [16].

AGP implements one service for the Interpreter Shell which is used to process the programs generated. For each program launched, a new Interpreter Shell is created. By default, a service runs in the main thread of the application that hosts it. In our implementation, we forced the Interpreter Shell to be launched into a separate process in order to reduce Application Not Responding (ANR) errors when bugged programs are pushed.

Developers never directly deal with the Interpreter Shell. Rather, they launch it through a class called Interpreter. The latter attaches an Interpreter Context to the Interpreter Shell, which enables saving of variables and active links to other components during the program execution (cf. Figure 1).

The Interpreter Context is specifically needed for a GP framework running on a mobile OS such as AGP. Unlike GP frameworks running on desktop computers, we do not have direct access to some sensors. For example, an application may notify the Android layer that a sensor is no longer required, however the OS must consider others applications concurrently accessing the sensor before deciding whether to turn it off. Since a GP developer does not have full access to the system, they need to keep trace of the previous actions performed to provide the state of the sensors in the program context. The Interpreter Context provides this functionality.

Figure 2 provides a high level view of the AGP framework. The core AGP framework provides the infrastructure for writing new GP application for Android smartphones. An application using AGP will require the developer to implement at least two services which can easily communicate with a Communicator class.

3.3 Common Data Structures

Functions and Terminals. Functions and terminals are the primitives of any GP system. Our framework allows reusability of functions and terminals for

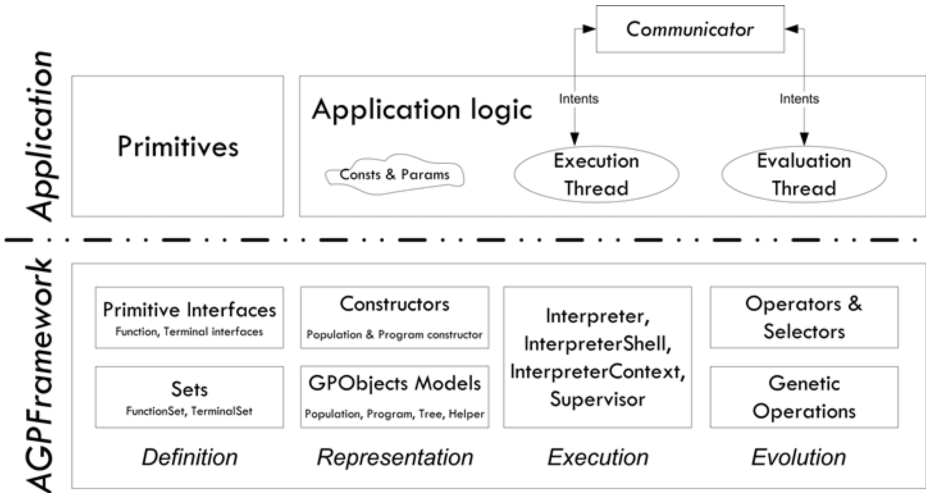


Fig. 2. Global architecture of AGP

all applications. With AGP, the developer has to implement our specific Java interface for functions or terminals, respectively FunctionInterface and TerminalInterface. It specifies the required methods for the GP framework such as the primitive arity, the string representation used to serialize, the execution and the estimated time cost.

To provide flexibility for the developer to vary the selection of primitives for their application, we integrate the Strategy Pattern [21]. FunctionSet and TerminalSet classes respectively store the available functions and terminals for an application, those which respectively implement FunctionInterface and TerminalInterface. The developer can easily add, remove or look for a primitive through dedicated methods implemented in FunctionSet and TerminalSet.

Population and Programs. Programs in AGP are represented as trees, and we define a population as a set of programs. We include the Builder Pattern [21] as a means for flexible population and programs inclusion into AGP. AGP already includes basic ways to get programs and populations, such as full random or empty set. Thanks to the Builder Pattern, the application developer is able to design their own program and population builders. For instance, to provide their program builder requires the implementation of the ProgramBuilderInterface (see Figure 3). It includes the getProgram() method which gets useful tools for creating programs such as the function and terminal sets, and a helper. We present the latter in Section 3.4.

Selectors and Genetic Operations. When the evaluation thread is running alongside the execution thread, programs are evaluated and receive a fitness value. In GP, the programs that perform well are chosen to breed the next generation. Selectors are organized following the Strategy Pattern in order to provide flexibil-

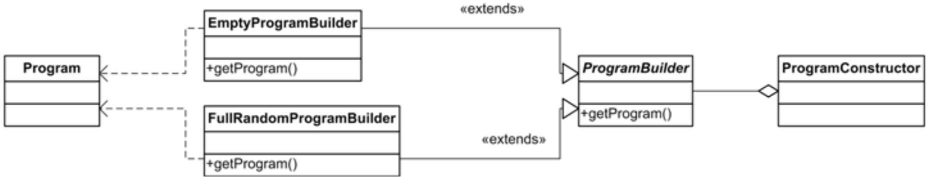


Fig. 3. Use of Builder Pattern in AGP illustrated with Program Constructor and two basics Program Builders. Abstract classes are shown in italic.

ity. This way, the developer can easily switch between standard selectors provided by AGP such as the Wheel selector, or create his own selector solution.

AGP then executes genetic operators on selected programs. In this regard, AGP currently supports two primary genetic operators widely used in GP, crossover and mutation, although it is extendible with more operators.

Save Current States. As smartphones are subject to reboot by the user, or undesired termination caused by battery depletion, AGP provides a safe mechanism to save the current state. We included serializable classes for the programs and the populations, which allow the developer to save these objects over reboot.

The developer is able to get program and population objects from their serialized form thanks to specific program and population builders (cf. Section 3.3). Indeed, UnserializePopulationBuilder and UnserializeProgramBuilder are builders available in AGP, able to construct, respectively, populations and programs from serialized form saved into a file.

3.4 Improvements

As GP is a stochastic process, convergence towards desired performance can take several generations. Contrary to typical computers, which usually run GP algorithms, smartphones have limited energy and computational capabilities. In order to ensure that convergence time remains within reasonable limits, AGP includes two components, namely the helper and the supervisor, which enable the use of expert knowledge to apply constraints to AGP-generated programs.

Helper. The helper is called during the program generation process done by any program builder. The developer can create one or several helpers for one application by implementing the AGP HelperInterface interface. Whenever a program is generated, AGP will refer to the helper evaluate() method to specifiy any correctness conditions that the program has to meet. For instance, in our geolocalization application (cf. Section 5), if a program doesn't call any location provider, we know that this program will be unable to locate the smartphone, so we can reasonably discard it without losing evaluation time.

Supervisor. The supervisor runs during the program interpretation. It can check constraints on-the-fly, and kill the Interpreter Shell if the program goes

out of bounds. For instance, the supervisor can control execution time. If the program execution is too long, the supervisor will automatically kill the program.

4 Google Reader Application

4.1 Application Purpose

Google Reader is a web-based aggregator released and maintained by Google. It works as a RSS feed reader, allowing users to get latest news from selected feeds. Many applications exist for consulting feeds from smartphones. However, news you like to read on a smartphone may depend on the kind of content, and its readability on a mobile device. For instance, it is easy to read short text news whereas it is uncomfortable to look for long articles, comics, infographics or flash animations. Moreover, such content might quickly deplete the user monthly capped data plan. Our Google Reader GP (GRGP) application takes advantage of the AGP framework to learn which feeds the user likes to read on their smartphone. Obviously, the feeds will be taken from the user Google Reader account.

The application is kept simple for demonstration purposes: whenever the user wants to get news, she asks for a news report which executes a GP program and returns the latest and unread news from feeds selected by the program.

4.2 Fitness Definition

The fitness definition is based on two sub-fitness functions:

$$Fitness = SubFitness_{News\ count} \times SubFitness_{News\ clicked}$$

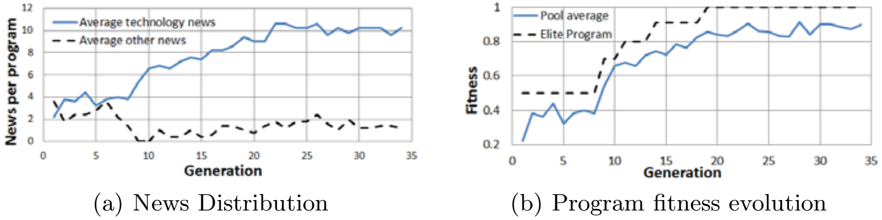
$$SubFitness_{News\ count} = \begin{cases} \frac{Displayed\ news}{Desired\ quantity} & Disp.\ news \leq Desired\ qty. \\ 1 & otherwise \end{cases}$$

$$SubFitness_{News\ clicked} = \left(\frac{Clicked\ news}{Displayed\ news}\right)$$

The news count refers to the minimum desired quantity of displayed news whenever the user requests a news report. This is a setting fixed by the user in the configuration menu. The clicked news corresponds to the quantity of news read over the amount of news given in the report. In other terms, it evaluates the interest of the user in the displayed news.

4.3 Results

We conducted our experiment with 7 sources from a real Google Reader subscription: 4 technology news websites (TechCrunch, TechLand, Engadget and Digital Trends), VisualLoop which gives fresh infographics, Break Videos for funny videos, and Business Green for latest green products. Even though the



(a) News Distribution (b) Program fitness evolution

Fig. 4. News distribution and program fitness evolution

concerned user was interested in all those sources, he is used to read the technology news on his smartphone rather than the other sources providing longer articles or heavy media files not optimised for reading on smartphone.

Figure 4(a) reports the average of displayed news per program over generations. In order to provide a clean graph, we grouped the news feed in two sub-groups: the 4 technology news, and the others (VisualLoop, BreakVideos, and Business Green). After 6 generations, our Google Reader using AGP learned the user preference for the technology news. However, it doesn't eliminate completely the diversity and keeps proposing some news from the other feeds yet with a lower likelihood.

As expected, GRGP learns to provide the desired minimum quantity of news per report, which we set to 10 for this experiment. Figure 4(a) confirms this convergence by looking at the sum of technology and other news.

The program fitness evolution indicates that our evolution strategy does its job, and leads to an increase of the elite program fitness. In Figure 4(b), the pool average represents the mean pool fitness (i.e. mean fitness of the 5 programs). Whenever the elite program gains in fitness, it subsequently leads to an increase of the pool average fitness.

5 Context-Aware Localization

With their application programming interface, modern smartphones OS enable programmers to develop their own solutions using available sensors on the device to get user context [17,18]. However, frequent usage of sensors remains a problem as it quickly depletes batteries. Thus, a key challenge is ensuring sensor sampling provides sufficient context without affecting battery lifetime. While user motion, sound activity or ambient light can be retrieved from one sensor, getting the position is a more complex problem as it can be obtained from several sources: ranging from the energy-hungry yet accurate GPS to the energy-efficient yet inaccurate cell-based method that relies on cellular phone towers. The various localization options on smartphones requires consideration of their availability and their energy/accuracy trade-off [20].

As this problem depends on too many contextual constraints as position, signal quality, device energy profile, it is unlikely to foretell which algorithm is

going to fit better than others for a user in a specific environment. The dynamic changes in constraints motivate the use of methods that can not only adapt to expected changes but can also learn how to deal unexpected changes in context. We introduce an application using AGP aimed to address this problem. We focus here on AGP's ability to achieve results for a problem depending on many contextual constraints.

5.1 Fitness Function Definition

The fitness function used in our localization application reflects the common trade-off between energy and accuracy in the localization field [19]. We introduce two fitness metrics: the accuracy fitness and the energy fitness, which respectively quantify the accuracy and energy efficiency of the provided solution. As positions are dynamic, we evaluate fitness every second during the evaluation period (n seconds). By the end of the evaluation time, we use the average of these subfitnesses to give a fitness to the program. The overall fitness is obtained by multiplying these subfitnesses. We choose this option as it discards any solution which doesn't provide any accuracy or could deplete the battery, while maintaining simplicity for the demonstration purpose of this paper.

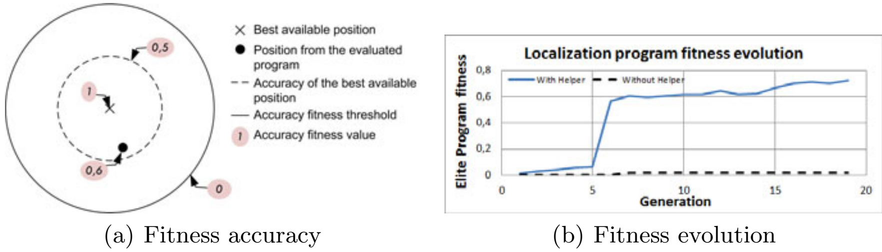
$$Fitness = \frac{\sum_{i=1}^{n} SubFitness_{Accuracy}(i) \times SubFitness_{Energy}(i)}{n} \quad (1)$$

Accuracy Fitness. In Android, localization can be achieved through several location providers used alone or combined. Usual location providers are GPS, Cellular Network and Wi-Fi. The developer could also brings other providers such as a contact-logging beacon method [20], or an accelerometer assisted algorithm. When the application is learning, the evaluation process keeps all the location provider on. It automatically picks the provider giving the most favorable accuracy. We call the output from this provider the best available position.

We use this best available position as a reference to attribute an accuracy fitness to the position provided by the evaluated program (cf. Figure 5(a)):
- if program position is within the accuracy of the best available position, we attribute an accuracy fitness following a linear rule from 1 to 0.5
- if program position is outside the former, but within a circle of twice the accuracy of the best available position (accuracy fitness threshold), we attribute an accuracy fitness following a linear rule from 0.5 to 0.

Energy Fitness. We define energy fitness according to a basic rule: we want to provide localization without depleting the battery by the end of the day as we assume users can charge their phones at the end of each day. We consider an average 1400 mAh battery capacity for the paper. To achieve our goal, the average power consumption should not exceed 63 mA (= 1400 / 22 hours). We define a day as 22 hours because we consider the phone as plugged for 2 hours per day.

We use the Android PowerProfile class to estimate the power consumption per chip. We access this class through the Java reflection mechanism. This enables



(a) Fitness accuracy (b) Fitness evolution

Fig. 5. The localisation application with AGP

AGP to assess the power cost of the evaluated program depending on the CPU usage and the chip used to locate the smartphone. For simplicity, we limit the energy fitness to a linear function, ranging from 1 for a idealistic case where the program doesn't cost any power to 0 for a program which requires more power than the one day energy budget.

5.2 Results

We conduct the experiment with populations of 12 programs. The program evaluation is limited to one minute. Our function set contains general operators such as addition, multiplication, and other application-specific operators: functions to switch location providers such as GPS, Wi-Fi or Cellular Network. These functions need access to some Android components, which is possible with AGP's design. Figure 5(b) shows the framework ability to get a program localizing the smartphone. After a rough first solution, it converges to smarter programs able to provide more efficient and accurate solutions.

We also conduct an experiment to evaluate the benefits from the use of Helper. Populations generated with the Helper provide a working solution in the first generation, and quickly have satisfying programs. On the other hand, populations generated without the Helper are stuck with non-working programs (zero fitness) for several generations. Then, they only get a slow evolution. It is mainly due to many programs which make no sense: they don't switch on a location provider or don't call any latitude nor longitude update.

6 Discussion and Conclusion

The innate inter-communication capability between mobile devices lends itself to the Island Model implementation where each mobile device hosts a population and evolved programs can be serialised and shared (migrated) [22]. While intuitively one expects the parallel resources will expedite convergence, the Island model is also known to generate better quality solutions [23]. As the framework currently does not implement such cooperative evolution mechanisms, our evolution is constrained by the modest computational resources of the mobile device. As such, we have attempted only simple problems where a single small population can converge within a reasonably short period. This configuration however would likely require very long convergence times for more complex problems.

The symbolic nature of GP means that logic can be readily seeded and is an intuitive choice where control of device's resources is typically performed programatically. It should be noted, however, that adaptive behaviour may not be desirable for all interactions. For example, the user interface should have a consistent feel across applications and adhere to the device or OS recommended UI design recommendations. However, within this constraint, adaptive behaviour may provide a method to overcome consistently undesirable application behaviour. The current configuration employs a fixed population structure which produces a number of likely low performing programs to be evaluated every generation even after the system has converged. This means that there will always be some 'annoying' behaviour. Ideally the population generation operations over time would change to converge the population to only desirable behaviour, however this would also reduce the ability of the system to respond quickly to changes in user preferences.

This paper has presented Android GP Framework (AGP) as the first genetic programming framework for the mobile Android OS. The framework considers the resource constraints and programming restrictions on Android smart phones for evolving logic, and introduces special components for quicker convergence by limiting execution to meaningful programs. We demonstrated AGP's capabilities with two applications and showed that it can converge to desirable performance quickly towards objective functions with multiple constraints. We believe AGP represents a first step towards versatile online personalisation in the growing smart phone market.

Acknowledgements. The authors would like to thank Brano Kusy for his valuable inputs in realising this work. This project was supported by the Sensors and Sensor Network Transformational Capability Platform at CSIRO.

References

1. Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., Estrin, D.: Diversity in Smartphone Usage. In: MobiSys 2010 (2010)
2. Dockhorn Costa, P., Ferreira Pires, L., Sinderen, M.: Designing a configurable services platform for mobile context-aware applications. International Journal of Pervasive Computing and Communications 1(1), 13–25 (2008)
3. Bae, J.S., Lee, J.Y., Kim, B.C., Rye, S.: Next Generation Mobile Service Environment and Evolution of Context Aware Services. In: Sha, E., Han, S.-K., Xu, C.-Z., Kim, M.-H., Yang, L.T., Xiao, B. (eds.) EUC 2006. LNCS, vol. 4096, pp. 591–600. Springer, Heidelberg (2006)
4. Miele, A., Quintarelli, E., Tanca, L.: A methodology for preference-based personalization of contextual data. In: EDBT 2009 (2009)
5. Koza, J.R.: Genetic Programming: on the programming of computers by means of natural selection. In: Complex Adaptive Systems. MIT Press, Cambridge (1992)
6. Ventura, S., Romero, C., Zafra, A., Delgado, J.A., Hervas, C.: JCLEC: a Java framework for evolutionary computation. Soft Comput. 12, 381–392 (2008)
7. Pyevolve, http://pyevolve.sourceforge.net/wordpress/?p=350

8. McPhee, N.F., Hopper, N.J., Reierson, M.L.: Sutherland: An extensible object-oriented software framework for evolutionary computation. In: Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22-25. University of Wisconsin, Morgan Kaufmann, Wisconsin, San Francisco (1998)
9. Gagne, C., Parizeau, M.: Open BEAGLE: A New Versatile C++ Framework for Evolutionary Computation. In: GECCO Late Breaking Papers, pp. 161–168 (2002)
10. Ismail, I.A., Ramly, N.A.E., Abd-ElWahid, M.A., ElKafrawy, P.M., Nasef, M.M.: Genetic Programming Framework for Fingerprint Matching. In: CoRR (2009)
11. Torres, R.S., Falcao, A.X., Goncalves, M.A., Papa, J.P., Zhang, B., Fan, W., Fox, E.A.: A genetic programming framework for content-based image retrieval. Pattern Recognition 42, 283–292 (2009)
12. Valencia, P., Lindsay, P., Jurdak, R.: Distributed Genetic Evolution in WSN. In: IPSN 2010, Stockholm, Sweden, April 12-16 (2010)
13. Lenaerts, T., Manderick, B.: Building a Genetic Programming Framework: The Added-Value of Design Patterns. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 196–208. Springer, Heidelberg (1998)
14. Lacerda, A., Cristo, M., Goncalves, M.A., Fan, W., Ziviani, N., Ribeiro-Neto, B.A.: Learning to advertise. In: SIGIR 2006, pp. 549–556 (2006)
15. Weise, T., Geihs, K.: DGPF: An Adaptable Framework for Distributed Multi-Objective Search Algorithms Applied to the Genetic Programming of Sensor Networks. In: BIOMA 2006, Ljubljana, Slovenia, October 9-10, pp. 157–166 (2006)
16. Android Reference, http://developer.android.com/reference/packages.html
17. Lu, H., Pan, W., Lane, N.D., Choudhury, T., Campbell, A.T.: SoundSense: scalable sound sensing for people-centric applications on mobile phones. In: MobiSys, pp. 165–178 (2009)
18. Thiagarajan, A., Ravindranath, L., LaCurts, K., Madden, S., Balakrishnan, H., Toledo, S., Eriksson, J.: VTrack: accurate, energy-aware road traffic delay estimation using mobile phones. In: SenSys, pp. 85–98 (2009)
19. Lin, K., Kansal, A., Lymberopoulos, D., Zhao, F.: Energy-accuracy trade-off for continuous mobile device location. In: MobiSys, pp. 285–298 (2010)
20. Jurdak, R., Corke, P., Dharman, D., Salagnac, G.: Adaptive GPS duty cycling and radio ranging for energy-efficient localization. In: SenSys, pp. 57–70 (2010)
21. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-oriented modelling and design (1991)
22. Folino, G., Spezzano, G.: P-CAGE: An Environment for Evolutionary Computation in Peer-to-Peer Systems. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 341–350. Springer, Heidelberg (2006)
23. Martin, W.N., Lienig, J., Cohoon, J.P.: Island (Migration) Models: Evolutionary Algorithms Based on Punctuated Equilibria. In: Handbook of Evolutionary Computation, pp. C6.3:1–C6.3:16. Oxford University Press (1997)

Genetic Programming for Generalised Helicopter Hovering Control

Dimitris C. Dracopoulos and Dimitrios Effraimidis

School of Electronics and Computer Science
University of Westminster
London, United Kingdom
{d.dracopoulos,d.effraimidis}@wmin.ac.uk

Abstract. We show how genetic programming can be applied to helicopter hovering control, a nonlinear high dimensional control problem which previously has been included in the literature in the set of benchmarks for the derivation of new intelligent controllers . The evolved controllers are compared with a neuroevolutionary approach which won the first position in the 2008 helicopter hovering reinforcement learning competition. GP performs similarly (and in some cases better) with the winner of the competition, even in the case where unknown wind is added to the dynamic system and control is based on structures evolved previously, i.e. the evolved controllers have good generalisation capability.

Keywords: Helicopter hovering, Nonlinear control, Neuroevolutionary control, Reinforcement learning.

1 Introduction

Genetic programming (GP) can be considered as an ideal candidate for automatic controller designs due to the direct mapping of its individual structures and control laws. So far however, it has been applied to a relatively small number of control problems. From these, only a few have been included in the literature as challenging for modern controllers [1]. Other computational intelligence control techniques (referred also as intelligent control approaches) such as reinforcement learning with the use of approximate dynamic programming, are considered as the current state of the art for the control of complex dynamic systems [2,3].

Helicopter hovering is considered as a challenging control problem for which intelligent control techniques have been utilised, in order to overcome the limitations of classical control theory. The problem refers to a helicopter attempting to hover as close as possible to a fixed position. The dynamics of the helicopter is nonlinear, high dimensional, complex, asymmetric and noisy [4,5]. Additionally, there is coupling between the different state variables of the dynamic system and an attempt to control one of them results in the destabilisation of another.

Helicopter hovering belongs to the reinforcement learning (RL) set of problems as there is no "teacher" signal available to the controller indicating sample correct actions. The competence of a controller is determined by the difference of the

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 25–36, 2012.
© Springer-Verlag Berlin Heidelberg 2012

desire state and the actual one, which is used to construct a reward or penalty (the reinforcement signal) to the controller at every point in time.

The problem of helicopter hovering (together with a few other control problems) was included in recent years in reinforcement learning competitions as a challenging benchmark used to construct new intelligent controllers [6].

Here, a genetic programming approach for generalised helicopter hovering is presented and compared with the neuroevolutionary approach which won the first position in the 2008 RL competition for the specific control problem [4,6]. Unlike the standard hovering problem, generalised hovering includes unknown wind. In each generalised version (domain) of the problem, wind is added according to some unknown probability distribution. The wind in each case in not known to the agent (controller). Thus any approach which attempts to do well by overfitting to a single domain, will do badly in the other domains which include different amounts of wind.

The next section provides the background for the control problem and provides the details of the dynamic system of a helicopter. Section 3 describes the neuroevolutionary approach which won the first position in the 2008 RL competition. Section 4 presents the GP approach to the problem and section 5 includes the results obtained from the GP and the neuroevolutionary implementations and a comparison between them. Finally, section 6 presents the conclusions for this work, and outlines future research.

2 The Dynamic System

The state variables of the helicopter dynamic system and the inputs are:

$$q = \left[P \; v^p \; \Theta \; \omega^b\right]^T = \left[x \; y \; z \; v_x^p \; v_y^p \; v_z^p \; \phi \; \theta \; \psi \; \omega_1^b \; \omega_2^b \; \omega_3^b\right]$$

$$u = \left[T_m \; T_t \; a_1 \; a_2\right]^T$$

where P is the helicopter position in inertial coordinates and $\Theta = \left[\phi \; \theta \; \psi\right]^T$ are the helicopter Euler angles. ϕ, θ, ψ are the roll, pitch and yaw angles respectively. v^p is the velocity with respect to the three axes: v_x^p is the forward, v_y^p is the lateral and v_z^p is the vertical velocity. $\omega^b \in \mathbb{R}^3$ is the vector which includes the body angular velocities.

The forces f^b and torques τ^b generated by the main rotor are controlled by the main rotor thrust T_m and the longitudinal a_1 and lateral a_2 tilts of the tip path plane of the main rotor with respect to the shaft. The tail rotor anti-torque is controlled by the tail rotor thrust T_t [7].

The equations of the rigid body subject to the body force $f^b \in \mathbb{R}^3$ and torque $\tau^b \in \mathbb{R}^3$ applied at the centre of mass and specified with respect to the body coordinate frame are given by the *Newton-Euler* equations:

$$\begin{bmatrix} mI & 0 \\ 0 & \mathcal{I} \end{bmatrix} \begin{bmatrix} \dot{v}^b \\ \dot{\omega}^b \end{bmatrix} + \begin{bmatrix} \omega^b \times m v^b \\ \omega^b \times \mathcal{I} \omega^b \end{bmatrix} = \begin{bmatrix} f^b \\ \tau^b \end{bmatrix}$$

where $m \in \mathbb{R}$ specifies the mass, $I \in \mathbb{R}^{3\times3}$ is an identity matrix, and $\mathcal{I} \in \mathbb{R}^{3\times3}$ is the inertial matrix [8]. If $R(\Theta)$ is the rotation matrix of the body axes relative to the inertial axes (superscript p), then by using the fact that $v^p = R(\Theta)v^b$ and $\Theta = \Psi(\Theta)\omega^b$, the equations of motion of the rigid body can be written as:

$$\begin{bmatrix} \dot{P} \\ \dot{v}^b \\ \dot{\Theta} \\ \dot{\omega}^b \end{bmatrix} = \begin{bmatrix} v^p \\ \frac{1}{m}R(\Theta)f^b \\ \Psi(\Theta)\omega^b \\ \mathcal{I}^{-1}(\tau^b - \omega^b \times \mathcal{I}\omega^b) \end{bmatrix}$$

The above system is a coupled non-linear, multivariable and under-actuated system with fewer independent control actuators than degrees of freedom to be controlled [7].

All the experiments described here, have been conducted with the use of an XCell Tempest helicopter simulator [9]. The specific simulator has been created for the RL-Competition [6]. The objective is to hover the helicopter around the origin. The time period is comprised of 6000 steps (0.1 seconds each) and in every step a penalty (reward) is returned to the controller from the simulator, describing the deviation of the helicopter from the origin. The penalty (or reward) constitutes the reinforcement signal and it is the only information available to the agent for learning. The reward is calculated as the negative sum over all state features, of the squared difference between that state feature and the fixed target position in which the helicopter wishes to hover:

$$R = -\sum_i (s_i - t_i)^2 \tag{1}$$

where s_i and t_i are the current value and the target value, respectively, of the ith state feature.

If the helicopter crashes during learning, a very large penalty is given. Its magnitude is the sum of the largest penalty for every remaining step. The range of the four controls of the helicopter has been normalised to the region $[-1, 1]$.

3 The Neuroevolutionary Approach

The neuroevolutionary reinforcement learning approach which won the first place in the 2008 RL Competition for helicopter hovering evolves neural network controllers using the accumulated reward. The neural networks are different policies mapping observations to actions. Starting from an initial population of 50 randomly generated networks the process of optimisation is based on a steady state evolutionary method which does not use generations. At every step, the worst performing network is selected and its weights are modified with crossover and mutation [4].

There are four different types of networks used in that work, which are distinguished based on their architecture and the initial configuration of their weights. SLP (single layer perceptrons) and MLP (multi-layer perceptrons) are the two

architectures used. The topology of the MLP was manually designed by a human specialist specifically for the helicopter control problem. Some initial knowledge can be applied to the neural networks by setting the weights to an equivalent base-line controller given by the RL competition which never causes the helicopter to crash. The controller is naive and can not solve the problem satisfactorily, as it performs poorly and does not approach the hovering point. According to this methodology, the initial population is formed by repeatedly performing weight mutations to the base-line controller with a predefined probability. The process of mutation involves the random generation of weights from a Gaussian distribution with zero mean and standard deviation of 0.8, which either replace or add itself to the initial weight.

The full details for this neuroevolutionary approach can be found in [4].

4 Application of Genetic Programming

The action space for the helicopter hovering control is a four dimensional continuous space. The four different actions are:

α_1 : longitudinal cyclic pitch (aileron)
α_2 : latitudinal cyclic pitch (elevator)
α_3 : main rotor collective pitch (rudder)
α_4 : tail rotor collective pitch (coll)

To apply the GP paradigm, four independent trees were allocated for each individual, one for every action. The population consisted of 500 quad trees. The initial structure of individuals were generated with the 'ramped half-and-half' method [10], with the 'full' method creating complete trees of length 2 to 6 and the 'grow' method setting the maximum length equally. Duplication was disallowed for the initial structures and one of the 500 trees was the equivalent of the default baseline controller supplied from the RL Competition described in the previous section. The controller is capable to avoid to crash the helicopter but it is unable to do well in the hovering problem. It has to be noted that the same baseline controller was also included in the initial population of the neuroevolutionary approach which won the RL Competition. Figure 1 shows the default controller in tree structure:

GP utilises the reinforcement signal given in equation (1), by accumulating it during every episode. The sum is used as the raw fitness. The reinforcement signal is negative and the adjusted fitness is calculated as follows:

$$a(i) = \frac{1}{1 + s(i)}$$

where $s(i)$ is the standard fitness and defined as the negative of the accumulated RL signal. i is the index of the individual in the population. Finally, the normalised fitness was calculated:

$$n(i) = \frac{a(i)}{\sum_{k=1}^{M} a(k)}$$

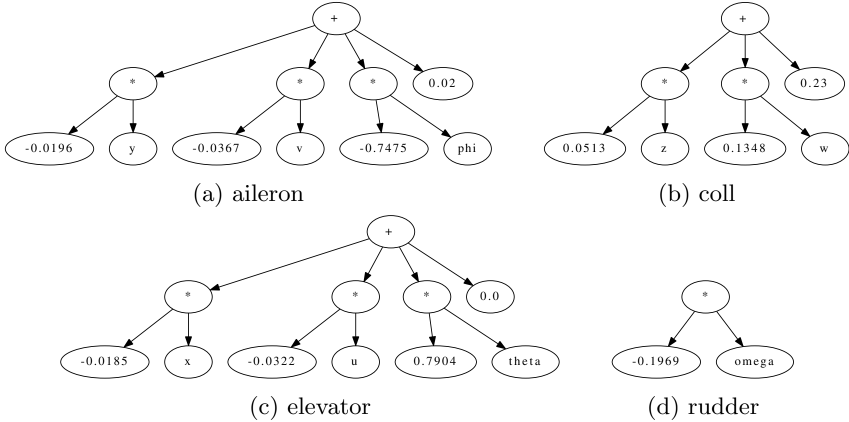


Fig. 1. The default baseline controller in the initial GP population

Table 1. Function set for the application of GP in helicopter hovering

Function	Symbol	Function	Symbol
Addition	ADD	Square	SQ
Subtraction	DIV	Cube	CUB
Multiplication	MUL	Greater Than	GT
Division	DIV	Sign	SIG
Absolute Value	ABS	Sin	SIN
Square Root	SQRT	Cosine	COS
Exponential	EXP		

The normalised fitness was used for the entire GP evolution.

The function set selected is shown in Table 1. From the 12 initial states of the dynamic system only 9 are independent, as the angular velocities can be calculated from the rest. Therefore, the terminal set used included the coordinates x, y, z, the velocities u, v, w, the angles θ, ϕ, ω and the real number coefficients that were essential to construct the default baseline controller.

The generalised version for the helicopter hovering problem was considered. This version includes noise in the flight dynamics. The actual noise added, its magnitude and range are unknown to the controller agent. Such generalised version of the problem (which was the case for the RL competition and its software simulator used in this work) forces the developed controllers to adapt their behaviour in dynamic changes in environmental conditions and prevent the overfitting to a specific control problem. The noise represents wind in the x and y dimensions. The wind velocities vary randomly from $-5m/sec$ to $5m/sec$ (this is not known to the evolving GP controllers). The RL simulation software has 10 different wind patterns providing 10 different modes of the simulation, i.e. 10 different sequential decision problems (SDPs). GP has been applied to all 10 of them. For every different mode, 10 different runs were done. The aim is to derive a GP controller which generalises well across different SDPs, including unseen cases.

The evolution was carried on for 119 generations (excluding the first initial population). The reproduction probability was 10% and the crossover probability was 90%. The maximum depth permitted after the crossover operation was 17 and in the case that one of the offsprings had depth which was longer than the allowed, the selected parent tree was copied without modification. The individual selection was fitness proportionate. Permutation and mutation were not allowed. The mutation process was tested but did not produce successful results.

During evolution, the default controller was copied to the next population independently from the reproduction process. Although, especially in early generations, the fitness of the default controller is high compared with the other individuals, thus it is highly probable to reproduce, the experiments showed that it might be extinguished after some generations resulting to less good controllers.

The output of a GP individual is calculated based on the following equation:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

where x is the evaluation of the GP tree. This was done so that the control inputs to the plant are in the range $[-1, 1]$, something which is required by the simulator.

5 Results

This work addresses the generalised version of the helicopter hovering problem. From the RL Competition simulator only 10 different SDPs were available for both training and testing. To optimise the process of finding the best controller which can then be applied to other SDPs as well (besides the 10 available), the 10 SDPs need to be used for training, validation and testing.

The same optimisation procedure was followed for both the GP and the neuroevolutionary approach. For every mode of the wind pattern (10 modes), 10 different populations were formed. The derivation of the controllers took 120 generations for the GP and 120000 episodes for the neuroevolutionary method.

The best controller in every generation was subsequently validated to the modes 0 to 4, and in the case the where the controller was actually evolved on any of the modes 0 to 4, this mode was excluded from the validation. According to the performance in the validation set, the best individual for every mode was selected. Thus, the validation runs determined which individual performs better (generalises) in unseen SDPs (i.e. SDPs not used for its evolution) .

Finally, the 10 best controllers (one from each mode), for both approaches (GP and neuroevolutionary) were tested in modes 5 to 9 to compare the results of the two approaches. Similarly with the validation runs, if the controller was evolved (trained) in one of the modes 5 to 9, this mode was excluded from the test runs. The best controller found by Genetic Programming is shown in in Appendix A.

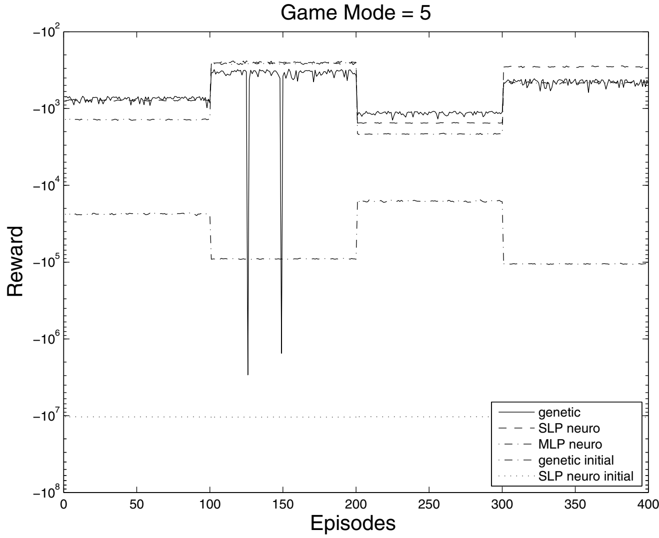


Fig. 2. Best Agents of mode 5

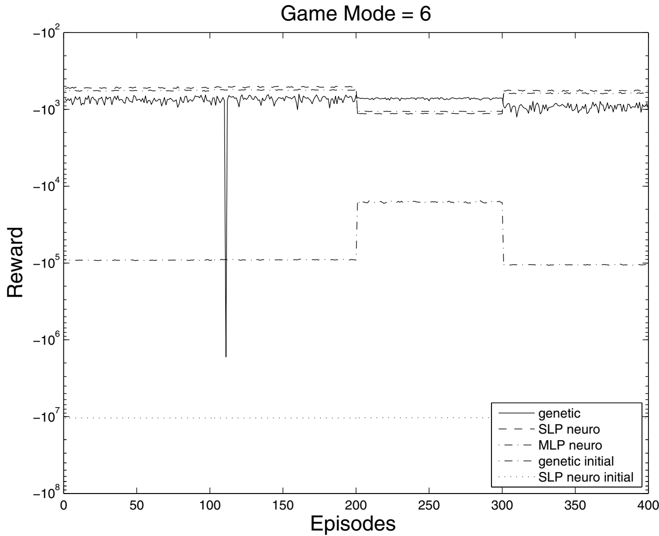


Fig. 3. Best Agents of mode 6

Figures 2, 3 show the performance of the two approaches for 2 out of the 5 test modes (SDPs). From Figures 2 and 3 it can be seen that the performance of the GP is equivalent to that of the SLP and MLP controllers evolved using the neuroevolutionary approach. For some modes, the GP is better.

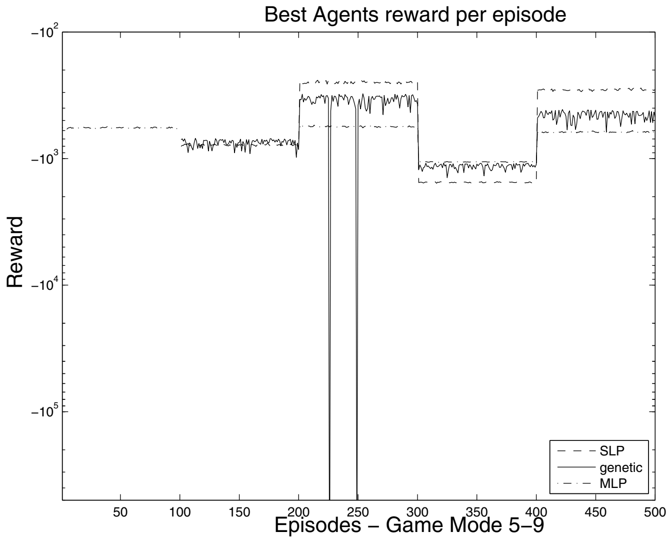


Fig. 4. The performance of the best agents for the various approaches

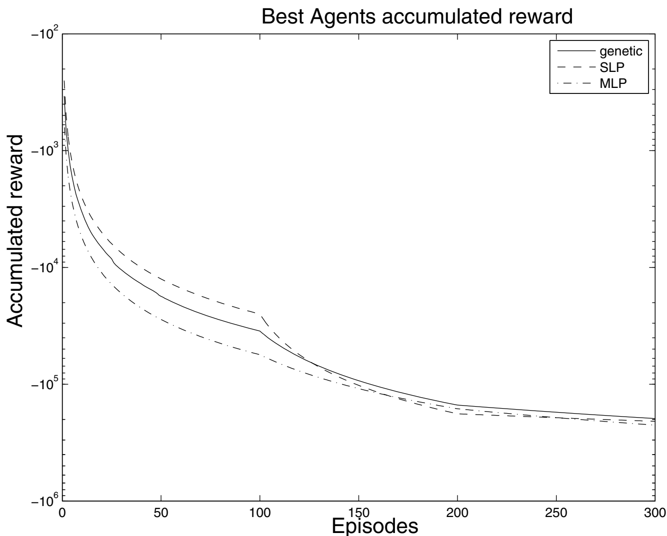


Fig. 5. The performance of the best agents for the various approaches

The diagrams also illustrate the difference in performance between the initial baseline controller in the population and the final one after the optimisation. The GP manages to improve the initial controller by two orders of magnitude.

The comparison in performance between the best GP and the best neuroevolutionary controller among all modes is shown in Figures 4, 5. Figure 4 illustrates

Table 2. Performance comparison for each mode between GP and SLP

Agent Mode		Mode 5			Mode 6			Mode 7			Mode 8			Mode 9		
		Crh	Bet	Acc	Crh	Bet	Acc	Crh	Bet	Acc	Crh	Bet	Acc	Crh	Bet	Acc
0	GP	1	0	-87792	13	0	-83736	0	0	-83681	28	72	-88301	4	0	-123590
	SLP	0	100	-56242	0	100	-40383	0	100	-56332	0	28	-99329	0	100	-54965
1	GP	3	34	-50547	0	3	-83438	4	2	-49350	0	0	-133935	2	0	-66411
	SLP	0	66	-47000	0	97	-74001	0	98	-41285	0	100	-60516	0	100	-46266
2	GP	7	35	-47512	51	0	-73633	5	11	-45030	63	29	-92901	23	0	-61243
	SLP	0	65	-45238	0	100	-36679	0	89	-40606	0	71	-98758	0	100	-48464
3	GP	0	100	-51732	0	100	-65718	3	97	-48658	0	1	-143389	0	100	-67374
	SLP	0	0	-96287	1	0	-109319	0	3	-90617	0	99	-130921	30	0	-154985
4	GP	1	0	-87257	2	0	-176645	1	2	-82308	0	0	-226003	3	0	-134277
	SLP	0	100	-74344	0	100	-130160	0	98	-69355	0	100	-171885	0	100	-86932
5	GP	-	-	-	0	79	-75581	2	0	-34468	0	100	-115604	0	0	-45969
	SLP	-	-	-	0	21	-78253	0	100	-25051	0	0	-154021	0	100	-28622
6	GP	0	0	-75617	-	-	-	1	0	-73645	0	100	-72229	0	0	-93995
	SLP	0	100	-52513	-	-	-	0	100	-51048	0	0	-113669	0	100	-57072
7	GP	2	97	-39566	0	98	-88398	-	-	-	0	99	-128574	2	97	-48604
	SLP	1	3	-50939	0	2	-127104	-	-	-	0	1	-155043	11	3	-63450
8	GP	31	3	-178694	0	0	-75126	45	4	-174114	-	-	-	42	25	-203970
	SLP	3	97	-80322	0	100	-45421	4	96	-78954	-	-	-	27	75	-100123
9	GP	3	0	-49156	0	0	-147568	1	0	-47917	0	0	-200945	-	-	-
	SLP	0	100	-38384	0	100	-77955	0	100	-35751	0	100	-159757	-	-	-

Table 3. Performance comparison between GP and MLP

Agent Mode		Mode 5			Mode 6			Mode 7			Mode 8			Mode 9		
		Crh	Bet	Acc	Crh	Bet	Acc	Crh	Bet	Acc	Crh	Bet	Acc	Crh	Bet	Acc
0	GP	1	99	-87792	13	0	-83736	0	100	-83681	28	72	-88301	4	96	-123590
	MLP	0	1	-173088	0	100	-63677	1	0	-172145	0	28	-108277	2	4	-195310
1	GP	3	0	-50547	0	100	-83438	4	0	-49350	0	100	-133935	2	0	-66411
	MLP	0	100	-33185	0	0	-259734	0	100	-33027	0	0	-397802	0	100	-46010
2	GP	7	93	-47512	51	31	-73633	5	95	-45030	63	37	-92901	23	77	-61243
	MLP	0	7	-104444	0	69	-74733	0	5	-106599	0	63	-161663	0	23	-114475
3	GP	0	100	-51732	0	100	-65718	3	97	-48658	0	100	-143389	0	100	-67374
	MLP	0	0	-110556	0	0	-106775	0	3	-106793	0	0	-187406	0	0	-150084
4	GP	1	97	-87257	2	98	-176645	1	92	-82308	0	100	-226003	3	97	-134277
	MLP	0	3	-113862	0	2	-429170	0	8	-98078	0	0	-537801	0	3	-252467
5	GP	-	-	-	0	100	-75581	2	0	-34468	0	100	-115604	0	60	-45969
	MLP	-	-	-	0	0	-139729	0	100	-25958	0	0	-214368	0	40	-46168
6	GP	0	0	-75617	-	-	-	1	0	-73645	0	100	-72229	0	0	-93995
	MLP	0	100	-57107	-	-	0	0	100	-55847	0	0	-106240	0	100	-61786
7	GP	2	0	-39566	0	100	-88398	-	-	-	0	100	-128574	2	98	-48604
	MLP	0	100	-26953	0	0	-266833	-	-	-	0	0	-382199	0	2	-74924
8	GP	31	52	-178694	0	10	-75126	45	45	-174114	-	-	-	42	48	-203970
	MLP	1	48	-188102	0	90	-68571	2	55	-188486	-	-	-	6	52	-221276
9	GP	3	54	-49156	0	100	-147568	1	82	-47917	100	100	-200945	-	-	-
	MLP	0	46	-47782	0	0	-382501	0	18	-54595	0	0	-556661	-	-	-

that the GP stands between the two different versions of the neuroevolutionary approaches, again with the few exceptions when it crashes the helicopter. The interrupted lines in the diagram are due to the fact that different agents did not participate in all test modes (the training mode was excluded from the test cases, i.e. if a controller was trained for mode 5, then this mode was excluded from its set of test cases to test performance)

Figure 5 demonstrates the accumulated reward of the different controllers for modes 7, 8, 9 (as some controllers used modes 5 and 6 for training). Accumulated

reward was the performance measure used in the RL competition to determine the best approach. This diagram excludes the two crashes of GP and it can be seen that the GP performance is very similar with the neuroevolutionary approaches (SLP and MLP).

Tables 2 and 3 include the comparison between the GP, SLP and GP, MLP respectively, for each of the 10 controllers evolved in each approach after validation. `Crh` indicates the number of crashes, `Bet` indicates in how many episodes the approach performed better than the other and `Acc` is the accumulated reward over the test episode cases. For example, the first row of Table 2 compares the best controllers of GP and SLP which were derived based on training in the SDP of mode 0 (the choice of the best controller was determined after running the best of each generation for each validation mode). These controllers are compared based on the test modes to check their generalisation ability. In the cases of a crash, that episode was not included in the calculation of the accumulated reward.

6 Conclusions and Future Work

GP is tested in a challenging control problem, the generalised helicopter hovering problem which was included in the set of benchmarks in the RL competitions for deriving new controllers. The GP approach is compared with a neuroevolutionary approach which won the first position for helicopter hovering in the 2008 RL competition.

It is shown that GP can successfully generate controllers for the generalised version of the problem, in which unknown wind is added to the dynamic system in the form of noise. The performance of the evolved controllers is tested in versions of the dynamic system that they have not encountered during evolution (training).

GP performs similarly with the winner of the RL competition, although in some cases its evolved controllers leads to a crash, something which also occurs with the winner of the RL competition. In certain modes (SDPs), the GP performance is better than the neuroevolutionary approach.

Future research should include how the avoidance of crashes could be incorporated in the evolution of GP controllers and dual operation of GP controllers, i.e. a different evolved controller becomes active depending on the specific region of the current state of the dynamic system.

A Appendix

The best individual derived through GP for which all results are presented in the paper is shown (unedited) below:

α_1 - `aileron`:

(+ 0.02 (* -0.0367 v)(+ (+ (* (* -0.0196 y)(+ (* -0.0196 y)(+ (* -0.7475 -0.0196)(* -0.0367 v)(+ (* -0.0196 v)(+ 0.02 (* -0.0196 v)(+ (* -0.0196 y)(* -0.0367 v)(+ (* -0.0196 (* phi v))(+ -0.0196 (* -0.0196 y)(* -0.0367 v)0.02)(* -0.7475 phi)0.02)0.02)0.02)(* -0.7475 phi)0.02)0.02

)(+ -0.0367 (+ (* 0.02 -0.0196)(* -0.7475 v)(* (* -0.0367 v)(* -0.0196 -0.0367))0.02)(* -0.7475 phi)0.02)0.02))(* -0.0367 (* -0.0196 y))(* -0.7475 -0.0196)0.02)(+ -0.0196 (* -0.0196 v)(+ (* -0.7475 -0.0196)(+ (* (+ (* -0.0196 y)(* -0.0367 v)(* -0.7475 0.02)0.02)y)(* -0.0367 v)(* -0.7475 (* -0.0196 -0.0367))0.02)-0.0196 0.02)(+ (* -0.0196 y)(* -0.0367 v)(* -0.7475 (+ (* -0.0196 (* -0.7475 phi))(* (* -0.0367 v)v)-0.0196 0.02))0.02))(+ (* (* -0.0367 -0.0196)(+ (* (* -0.0367 0.02)y)(* -0.0367 v)(* -0.7475 (* -0.0196 -0.0367))0.02))(+ (* 0.02 y)(* -0.0367 v)(* -0.7475 (* phi v))0.02)(* -0.7475 phi)(* -0.0196 y))(+ 0.02 (* -0.0367 -0.0196)(+ (+ (* -0.0196 -0.0196)(* 0.02 v)(* -0.7475 -0.0367)(* -0.0196 y))(* 0.02 -0.0196)(+ (* (+ (* -0.0196 y)(* (* 0.02 -0.0196)v)(* -0.7475 -0.0196)0.02)y)(+ (* -0.7475 y)(* -0.0367 v)(* -0.7475 (* -0.0196 -0.0367))-0.0367)(* (* (+ (* -0.0196 y)(+ 0.02 (* -0.0367 v)(+ (* -0.0196 v)v (* -0.7475 phi)(* 0.02 y))0.02)(+ (* (+ (* 0.02 -0.7475)(* -0.0367 v)(+ (* -0.0196 y)(+ (* -0.0196 y)(* -0.0196 0.02)(* phi (* (* -0.0367 v)v))0.02)(* (* -0.7475 (* (* -0.0367 v)v))phi)0.02)0.02)y)(+ (* -0.0196 y)(* (* -0.0196 -0.0367)v)(* (* -0.0367 v)(* -0.0196 -0.0367))0.02)(* -0.7475 phi)0.02)0.02)(* -0.0196 -0.0367))phi)0.02)0.02)(+ (* -0.0196 v)(+ 0.02 (* -0.0367 v)(+ (* -0.0196 v)(+ 0.02 (* -0.0196 v)(+ (* -0.0196 y)(* -0.0367 v)(+ (* -0.0196 (* (* -0.0367 v)(* -0.0367 v)))(+ (* -0.0196 y)(* -0.0367 v)(* -0.0367 v)0.02)(* -0.7475 phi)0.02)0.02)0.02)(* -0.7475 phi)0.02)0.02)(+ (* 0.02 y)(+ (* -0.0196 y)(* 0.02 v)(* (* -0.0367 (* -0.0196 y))(* -0.0196 -0.0367))0.02)(* -0.7475 phi)(* -0.0367 y))0.02)))(+ 0.02 (* -0.0367 y)(+ (* (* -0.7475 v)0.02)(+ 0.02 (* -0.0367 v)(* -0.7475 (* (* -0.0367 v)-0.0367))0.02)(* -0.7475 phi)0.02)(* -0.0196 -0.0367)))

α_2 - `elevator`:

(+ (* -0.0185 x)(* -0.0322 u)(+ (* -0.0185 (* -0.0185 (+ (* -0.0185 x)(* -0.0322 u)(* 0.7904 theta)0.0)))(+ (* -0.0322 x)(* -0.0185 x)(+ (* 0.7904 (* -0.0322 theta))(* -0.0322 -0.0185)(+ (* -0.0185 x)(* -0.0322 u)(+ (* -0.0185 x)(+ (+ (* -0.0185 x)(* -0.0185 x)(+ (* x x)(+ (* -0.0322 x)(* (* 0.7904 (* -0.0322 u))(* u x))(+ (* (* -0.0322 u)(* x theta))(* -0.0322 u)(+ (* -0.0185 x)(* -0.0322 u)theta (* -0.0185 -0.0185))0.0)(+ (* -0.0185 (+ (* 0.7904 x)(* (* -0.0185 u)u)(* 0.7904 theta)(+ (* -0.0185 x)-0.0185 (* -0.0322 u)-0.0322)))(* -0.0322 u)(* 0.7904 theta)(* -0.0185 u)))(+ (* -0.0185 (+ (* u x)(* -0.0185 u)(* 0.7904 theta)0.0))(* -0.0322 u)(* x theta)(* -0.0185 x))(* -0.0185 x))(* x -0.0322))(* (* 0.7904 -0.0322)u)(* -0.0322 u)(* (* 0.7904 (+ (* -0.0185 x)(* -0.0185 x)(* (+ (* (* -0.0322 (* -0.0185 u))(* -0.0322 theta))(* u u)(+ (* -0.0185 x)(* -0.0322 u)theta (* -0.0185 x))0.0)(* (* -0.0322 u)u))(+ theta (* -0.0322 u)(* 0.7904 theta)(* (* -0.0185 x)x))))x))(+ (* (* -0.0185 x)(+ (* -0.0185 x)(* (* -0.0185 (* -0.0185 x))-0.0185)(* -0.0185 theta)0.0))(* -0.0322 u)(+ (* -0.0185 x)(* -0.0322 u)(* 0.7904 theta)(* -0.0185 x))(* -0.0185 x))(* (* 0.7904 (+ (* -0.0185 x)(* -0.0185 x)(* 0.7904 x)(+ (* -0.0185 x)(* -0.0322 u)(* 0.7904 theta)(* (* -0.0185 x)x))))x))theta)(* -0.0185 x))(+ (* -0.0185 (+ (* -0.0185 x)(* -0.0322 u)(* 0.7904 theta)(+ (* -0.0185 x)(* (* -0.0322 u)(* 0.7904 u))(* x -0.0322)-0.0322)))(* -0.0322 u)(+ (* -0.0185 x)(+ (+ (* -0.0185 x)(* -0.0185 x)(+ (* -0.0185 x)(+ (* -0.0322 x)(* (* 0.7904 (* u u))u)(+ (* (* -0.0185 x)(* -0.0322 theta))(* -0.0322 u)(+ (* -0.0185 x)(* -0.0322 u)theta (* -0.0185 x))0.0)(+ (* -0.0185 (+ (* -0.0185 x)(* (+ (* -0.0185 x)(* -0.0322 u)(* 0.7904 theta)(* -0.0185 x))(* -0.0185 x))(* 0.7904 theta)(+ (* -0.0185 x)(* (* 0.7904 (* -0.0322 x))-0.0185)(* -0.0322 u)-0.0322)))(* -0.0322 -0.0185)(* 0.7904 theta)(* (* -0.0322 u)-0.0322)))(+ (* -0.0185 (+ (* u x)(* -0.0185 u)(* 0.7904 theta)0.0))(* -0.0185 x)(* x theta)(* -0.0185 x))(* x x))(* (* -0.0185 -0.0185)theta))(* u u)(* x u)(+ (* -0.0185 (* u x))(* (* x theta)u)(* 0.7904 (* (* -0.0185 theta)u))(* -0.0185 u)))(+ (* -0.0185 theta)(* -0.0322 u)(* x x)(* 0.0 x))(* -0.0185 x))(* -0.0185 (* -0.0322 u))))(+ (* (* -0.0185 x)(+ (* -0.0185 x)(* -0.0322 u)-0.0185 0.0))(* -0.0322 u)(* x theta)(* -0.0185 x))(* -0.0185 x))(* -0.0185 x))

α_3 - `rudder`:

(* (/ (* (CUB (ABS omega))u u u)(* (ABS x)(ABS x)theta omega))(* (CUB (ABS w))(* x theta (CUB u))(* (ABS (/ (CUB w)(ABS x)))(ABS (CUB (ABS x)))(* u u w w))theta)u)

α_4 - `coll`:

(+ z (* (+ (* 0.0513 (+ (+ (* 0.1348 w)(+ (+ 0.1348 (+ z (* z z)w)(+ (* (* z 0.1348)(* (+ (* 0.0513 z)(* 0.1348 w)0.1348)w))(* 0.1348 w)w))(+ (* 0.1348 z)z (+ z (* 0.1348 z)(+ (+ z (* z w)(* w w))0.1348 (+ 0.0513 (* 0.1348 w)w))))(+ (* 0.0513 z)(* (* 0.1348 0.1348)z)(* z (+ z 0.1348 (+ z (* 0.0513 (* 0.1348 z))z)))))(+ 0.1348 0.0513 0.0513))(* z z)w))(* 0.1348 (+ (* 0.1348 z)(* 0.23 w)(* z 0.1348)))(+ 0.1348 0.1348 0.0513))w)0.23)

References

1. Dracopoulos, D.C., Piccoli, R.: Bioreactor Control by Genetic Programming. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part II. LNCS, vol. 6239, pp. 181–188. Springer, Heidelberg (2010)
2. Si, J., Barto, A.G., Powell, W.B., Wunch II, D. (eds.): Handbook of Learning and Approximate Dynamic Programming. Wiley (2004)
3. Werbos, P.J.: Foreword - ADP: The key direction for future research in intelligent control and understanding brain intelligence. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 38(4), 898–900 (2008)
4. Koppejan, R., Whiteson, S.: Neuroevolutionary reinforcement learning for generalized helicopter control. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 145–152. ACM (2009)
5. Ng, A., Kim, H., Jordan, M., Sastry, S., Ballianda, S.: Autonomous helicopter flight via reinforcement learning. In: Advances in Neural Information Processing Systems, pp. 799–806. MIT Press (2004)
6. RLC: Reinforcement learning competition (2009), http://www.rl-competition.org
7. Gonzalez, A., Mahtani, R., Bejar, M., Ollero, A.: Control and stability analysis of an autonomous helicopter. In: Proceedings of World Automation Congress, vol. 15, pp. 399–404 (June-July 2004)
8. Koo, T.J., Ma, Y., Sastry, S.S.: Nonlinear control of a helicopter based unmanned aerial vehicle model (2001), http://citeseer.ist.psu.edu/417459.html
9. Abbeel, P., Ganapathi, V., Ng, A.: Learning vehicular dynamics, with application to modeling helicopters. In: Advances in Neural Information Processing Systems, vol. 18, pp. 1–8 (2006)
10. Koza, J.R.: Genetic Programming On the Programming of Computers by Means of Natural Selection. MIT Press (1992)

Cartesian Genetic Programming for Memristive Logic Circuits

Gerard David Howard, Larry Bull, and Andrew Adamatzky

University of the West of England, BS16 1QY, UK
{david4.howard,larry.bull,andrew.adamatzky}@uwe.ac.uk

Abstract. In this paper memristive logic circuits are evolved using Cartesian Genetic Programming. Graphs comprised of implication logic (IMP) nodes are compared to more ubiquitous NAND circuitry on a number of logic circuit problems and a robotic control task. Self-adaptive search parameters are used to provide each graph with autonomy with respect to its relative mutation rates. Results demonstrate that, although NAND-logic graphs are easier to evolve, IMP graphs carry benefits in terms of (i) numbers of memristors required (ii) the time required to process the graphs.

Keywords: Cartesian genetic programming, Self-adaptation, Nanotechnology, Boolean logic, Memristors, Robotics.

1 Introduction

The advent of nanoscale fabrication has given rise to a need for novel manufacturing paradigms. One such change involves the projected adoption of memristor (memory-resistor) technology as a processing/memory medium. The memristor is a fundamental passive two-terminal device first theoretically characterized by Chua [3] and manufactured at the nano scale by HP labs in 2008 [23]. A memristors state (memristance) is nonvolative and dependent upon past activity. Nonvolative memory [7] is perfect for low-heat, low-power storage, and the device's dynamic internal state facilitates information processing. These properties make the memristor an ideal candidate for use in nanoscale architectures [13]. Moreover, when overdriven by voltage they function as a binary latch (e.g. [22]).

Borghetti et al. [2] have recently described how memristive latches can realise Boolean logic. By connecting two memristors to a load resistor, they were able to implement two-input material implication (IMP) logic. Adding a further memristor to the circuit allowed two-input NAND to be synthesised from two IMP operations in serial.

Cartesian Genetic Programming (CGP) [19] is a modern derivative of Genetic Programming (GP) [10] with its origins in evolutionary circuit design. In its canonical form, bounded-size (bloat-free) arrangements of sequentially-executed logical processing elements are evolved, which can then be implemented as physical circuits. In this paper we use CGP to evolve circuits of exclusively IMP or

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 37–48, 2012.
© Springer-Verlag Berlin Heidelberg 2012

exclusively NAND logic; our hypothesis is that it is advantageous to implement directly in IMP rather than constructing NAND from IMP. Comparisons are performed on circuit design and robotics test problems.

2 Background

2.1 Memristors

Memristors (memory-resistors) are the fourth fundamental circuit element, joining the capacitor, inductor and resistor. A memristor can be defined as a resistor whose current resistance value (a) depends on the previous charge that has passed through it (b) is nonvolatile. Formally, a memristor is a passive two-terminal electronic device that is described by the non-linear relation between the device terminal voltage, v, terminal current, i (which is related to the charge q transferred onto the device), and magnetic flux, φ, as (1) shows. Resistance increases or decreases depending on the direction of the current.

$$v = M(q)i \quad \text{or} \quad i = W(\varphi)v \tag{1}$$

The memristance (M) and memductance (W) properties are both nonlinear functions, defined in (2) as:

$$M(q) = d\varphi(q)/dq \quad \text{and} \quad W(\varphi) = dq(\varphi)/d\varphi \tag{2}$$

Physical applications of memristors include the manufacture of nanoscale neural crossbars [22] and memristor-transistor self-programming switching logic circuits [1], both using HP memristors [23]. In particular, [1] highlights the prospect of reconfigurable circuits, such that the usage of various parts of the chip (e.g. storage, processing) can be dynamically altered as required.

The state of a memristive latch can be described by its instantaneous resistance value (switch open = logical 0 = high-resistance, switch closed = logical 1 = low-resistance). Switching behaviour is elicited from the memristor by overdriving the device with a voltage of a certain polarity such that it switches from one state to the other in a single step; applying a voltage of the opposite polarity will switch the memristor back to its original state. This behaviour makes the memristor amenable to use as a Boolean processing element.

Any two-input Boolean function can be implemented with memristors [12]. Material implication has been identified as the 'natural' form of logic for memristors [12]. It has recently been shown [2] that an IMP function can be created from two overdriven memristors (p and q) and a load resistor in a single processing step, whilst a group of three overdriven memristors (p, q and s) and a load resistor is shown to use two IMP operations to perform NAND ($s'' \leftarrow pNANDq$) in two processing steps (assuming $s = 0$, (i)$s' \leftarrow pIMPs$ (ii)$s'' \leftarrow qIMPs'$). Memristive IMP operations are *stateful* in that the state of the latch can be used for processing and memory; experimental details in [2].

It should be highlighted that memristors can be characterised in various ways, from dynamical continuous-valued components operating in continuous time (e.g. for neural systems [9]) to binary latches [11]. Here we take a higher-level view and characterise groups of memristors by the logic function they implement (IMP = 2 memristors, NAND = 3 memristors) whereby a single CGP node implements one of the two functions.

Due to the computational completeness of NAND-based circuitry, it may be tempting for circuit designers to generate memristive circuits based on NAND operations. In this paper we show that IMP is an attractive alternative to NAND, specifically for memristive implementations.

2.2 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is a modern form of GP in which a string of integers is used to encode a two-dimensional feedforward connected graph of nodes with dimensions $n_r \times n_c$ (where n_r and n_c are the number of nodes in a row / column respectively). Each node is an n-tuple specifying $n-1$ inputs and an index to a function that operates on those inputs to produce an output; a complete genome comprises a concatentation of nodes along with a problem-dependant number of program inputs and output indices. The n_i program inputs are assumed to be the first n_i nodes of the genome, the n_o program outputs are randomly selected within an upper bound of nodes that are part of the program specification ($n_i + n_r \times n_c$). The number of inputs to a node (the node arity) is set to the highest arity function in the function list.

CGP nodes always take their inputs from nodes in columns preceding their own; a "levels-back" parameter, l, determines the maximum number of columns that a connection can span. It should be noted that (i) program inputs may input to any node in the graph, (ii) a program input may also be a program output, (iii) a given node may be disconnected, in which case it does not affect the function of the graph. An example CGP genotype-phenotype mapping is given in Fig. 1. Execution of a graph occurs in two stages: decoding and processing. Decoding starts from the right hand side of the graph (e.g. the final output node) and constructs a connectivity map from each node to its predecessors to identify which genotypic nodes are active in the phenotype. Once the active nodes are delineated, processing applies the current problem input to each of the n_i inputs and processes the graph in a feedforward manner, reading the output from the output nodes.

Evolution involves a Genetic Algorithm (GA) [8] combined with either hill-climbing (as used in early Evolutionary Strategies (ES) [21]) or tournament selection [4]. Deterministic amounts of point mutation are used to alter any allele (input, function or output index) in the genome, set to some percentage of the total number of allelles. Inputs are constrained to connect to program inputs or nodes in previous columns. Function mutations are limited by the number of functions in the function list. For typical implementation details see [15].

Related CGP applications include both circuit design [16] and robot control [6]. It has also been used in a modified form to find generalised online solutions

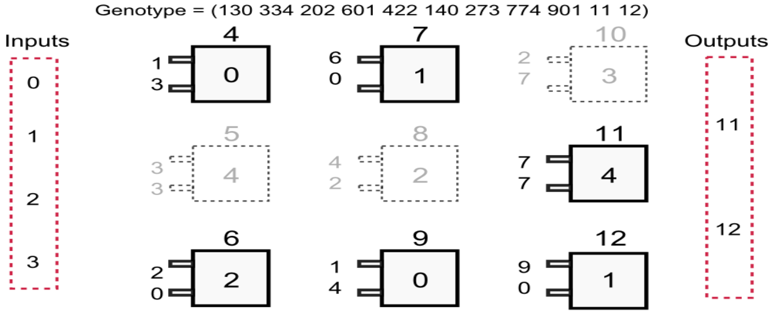


Fig. 1. Displaying a decoded CGP graph and its genotype. Parameters are n_i=4, n_o=2, n_r=3, n_c=3, l=1. Functions range from 0 to 4. Redundant nodes are shown with a faded dashed line, functions are shown inside each node and inputs to the left of it. Indices appear above the node; outputs are read from nodes 11 and 12.

to the parity problem [5]. GP has previously been used to evolve circuits with NAND gates [20]. In summary, we have proposed memristors as a candidate for implementing logic at the nano scale, and CGP as an suitable medium for evolving memristive logic circuits.

3 Genetic Algorithm

In our steady-state GA, offspring are chosen via binary tournament selection with a tournament size of 4. The offspring are inserted into the population and the network with the lowest fitness deleted to respect the maximum population size parameter.

3.1 Self-Adaptive Mutation

We utilise self-adaptive (SA) mutation rates in place of deterministic mutation events. SA mutation is implemented as in ESs [21] to dynamically control the frequency and magnitude of mutation events taking place in each genome, potentially allowing increased structural stability in highly fit graphs whilst allowing less fit graphs to vary more strongly per GA application. Each SA parameter is initialized uniform-randomly in the range [0.005, 1]. During a GA application, each SA parameter is modified as in (3), where p represents the SA parameter value and N is a normal distribution. The offspring then adopts this new value and applies it to its own genome, before being inserted into the population.

$$p \rightarrow p \exp^{N(0,1)} \tag{3}$$

Two parameters were used, μ (controlling mutation of node inputs) and ψ (controlling mutation of output indices). As each graph consists of only one function type, function alleles were not mutated. Input mutation affected all inputs in the

genome, and randomly reassigned a node input (respecting certain constraints, see e.g. [19]) upon satisfaction of the μ parameter. Similarly, satisfaction of the ψ parameter at any output index allele randomly reassigned the output index to any node in the genome.

Initial investigations compared this dual-rate mutation strategy to (i) a single rate strategy where μ controlled both input and output allele mutation with ψ being unused and (ii) deterministic amounts of point mutation (10%). Using the 6-multiplexer problem (results not shown) it was found that the dual-rate strategy more frequently found solutions with optimal fitness and low phenotypic node numbers. The dual-rate strategy was therefore employed throughout the remainder of the experiments.

4 Experimental Setup

All experiments had a population size of 20. CGP parameters were n_r=1, n_c=2000 and l=n_c (as this allows the most freedom with respect to connectivity patterns see e.g. [18]). Numbers of inputs and outputs were problem-dependent and summarised in Sections 5 and 6. Each graph also has access to constant 0 and 1 inputs, which are treated as program inputs. As we supply a constant 0 input to the circuits, IMP graphs have the potential to be computationally complete, as noted in [2].

A trial began with decoding a graph to delineate the active phenotype, followed by presentation of the first problem instance to the graph. Each active node was then sequentially processed by order of its index and its output stored for potential use by nodes in later columns. After every active node was processed, the value on the output node(s) was taken as the response. This was repeated for every problem input, at the culmination of which the trial ended. A graph that produced the correct output for every instance was said to have *solved* the problem.

Fitness assignment is covered in Section 5 for circuits and Section 6 for robotic experiments. In the event that multiple candidates possessed identical fitness values, tournament winners were selected randomly from amongst the candidates up to the trial in which the problem was solved. After this point, the candidate with the lowest phenotypic size was taken. If multiple candidates possessed the same fitness and phenotypic size, a random selection was made amongst these tied parents.

Each experiment was repeated ten times, the statistics recorded were the averages of those ten runs. The current state of the system was stored intermittently (intervals shown in Table 1) throughout the experiments and used to create the results that follow. In the following tables, "Performance" is the first generation in which all instances of the test problem are solved. If a problem was not solved within the allocated timeframe, its performance was set to the maximum number of trials. "High fitness" refers to the average fitness of the highest-fitness network in each run. "Active nodes" is the average non-redundant nodes (not including program inputs) in the population.

5 Logic Circuits

A number of common circuit design problems were initially used for evaluation, with parameters shown in Table 1. As every problem studied represented multiple problem instances, each graph was trialed on every instance. A graph received a fitness increment of 1 in the event of the desired output being achieved for that instance, 0 otherwise - the total fitness achievable by a graph equalled the total number of problem instances. Five-, six-, and seven-bit parity problems were trialed but not reported on as IMP circuits could not find a solution within the trial limit with the parameters used. It should be noted that standard circuit design tools could be used to (optimally) create the circuits tested; these problems are used to explore the basic characteristics of the approach relatively systematically before moving to more complex tasks, such as robotics.

Table 1. Parameter values for all circuit experiments. Number of inputs does not include constant inputs.

Problem	Inputs	Outputs	Trials	Test interval	Max fitness
6 MUX	6	1	300000	1000	64
11 MUX	11	1	10000000	10000	2048
6 DEMUX	3	4	10000000	10000	8
2 PARITY	2	1	10000	100	4
3 PARITY	3	1	1000000	1000	8
4 PARITY	4	1	10000000	10000	16

5.1 Results

Evolvability. Evolvability is here defined as "the ease at which a maximally-fit solution is evolved". Table 2 shows NAND graphs to be more amenable to evolution, producing 53/60 total successful experiments to 30/60 successful IMP experiments. Similarly, NAND graphs were statistically higher performing on all problems (t-test $p=4.97\times10^{-4}$ for 6 MUX, $p=7.1\times10^{-3}$ for 11 MUX, p=0.015 for 6 DEMUX, p=0.021 for 2 PARITY, p= 6.73×10^{-5} for 3 PARITY and $p=4.67\times10^{-3}$ for 4 PARITY). In terms of mean highest fitess, NAND graphs were either equal (p=1 for 6 MUX) or better (p=0.048 for 11MUX, $p=1.89\times10^{-3}$ for 6 DEMUX, p=0.17 for 2 PARITY, p=0.01 for 3 PARITY, $p=1.11\times10^{-3}$ for 4 PARITY) than their IMP counterparts. Due to the single function node representation, comparisons with other CGP implementations were not made, although our approach is almost certainly slower (see e.g. [17]). Lack of scalability of IMP circuits could be circumvented by (i) running for an exhaustive number of generations (ii) evolving smaller, modular circuit designs.

It is interesting to note that the SA parameters are context-sensitive and vary depending on the logic function used in the graph, although without statistical significance; p= 0.121 to 0.619 for μ and p= 0.231 to 0.839 for ψ. Statistically significant differences ($p<0.05$) between the two rates of mutation (e.g. μ vs. ψ

Table 2. Results of evolving IMP and NAND based CGP logic circuits on the test problems. Node and SA parameter statistics taken from successful runs only.

	Problem	Perf	Avg. fit	Solved	Avg. nodes	Min. nodes	μ	ψ
IMP	6 MUX	1.27×10^5	64	10	13.4	12	0.01	0.047
	11 MUX	6.04×10^6	2024	5	28	27	0.009	0.045
	6 DEMUX	9.0×10^6	6.3	1	12	12	0.097	0.025
	2 PARITY	3516	3.8	8	4.3	4	0.02	0.049
	3 PARITY	7.6×10^5	6.5	5	10	8	0.013	0.035
	4 PARITY	9.28×10^6	14	1	16	16	0.011	0.053
NAND	6 MUX	2.54×10^4	64	10	13.5	12	0.012	0.042
	11 MUX	5.46×10^5	2048	10	29.8	26	0.008	0.044
	6 DEMUX	4.3×10^6	7.6	6	12.5	12	0.011	0.028
	2 PARITY	92.4	4	10	5.4	4	0.017	0.074
	3 PARITY	1.61×10^4	8	10	9.1	8	0.015	0.043
	4 PARITY	3.28×10^6	15.6	7	13.9	12	0.01	0.037

on each problem are observed except for IMP circuits solving 6 DEMUX and 2 PARITY. This disparity somewhat justifies the use of a dual-rate mutation strategy.

Solution Size and Efficiency. IMP takes two memristors a single clock tick to realise, whereas NAND takes three memristors and two clock ticks [2]. We employ two methods to evaluate the circuits; *cost* and *efficiency*. Cost is defined in (4), where n is the number of nodes in the circuit and m the number of memristors per node.

$$cost = n \times m \tag{4}$$

T(A), defined in (5), is the time taken to process the circuit and *ct* is the number of clock ticks required per node. The efficiency ratio (*eff*) of two circuits A and B is then calculated (6).

$$T(A) = n(A) \times ct(A) \tag{5}$$

$$eff(B) = T(A) \times n(A)/T(B) \times n(B) \tag{6}$$

Table 3 shows that IMP circuits are approximately twice as efficient when compared the equivalent NAND circuits on all test problems, apart from 4 PARITY which is more equal yet still favours IMP. This indicates that the evolved IMP circuits can be processed more expediently in a given hardware implementation. The minimum cost per circuit is also universally lower for IMP circuits, meaning a smaller, potentially denser, hardware footprint.

Overall these results demonstrate that the initial difficulty of evolving highly fit IMP-based logic circuits eventually yields designs that are both faster to process and require fewer memristors. It should be noted that solution size and efficiency are both more important than evolvability given prospective hardware implementations, as such designs would typically be evolved offline before realisation. An example evolved circuit is shown in Fig. 2.

Table 3. Cost, time taken and efficiency of the best evolved IMP and NAND based CGP logic circuits for each test problem.

	Problem	Min. Cost	Min. T	Efficiency
IMP	6 MUX	24	12	2
	11 MUX	54	27	1.925
	6 DEMUX	24	12	2
	2 PARITY	8	4	2
	3 PARITY	16	8	2
	4 PARITY	32	16	1.125
NAND	6 MUX	36	24	0.5
	11 MUX	78	54	0.519
	6 DEMUX	36	24	0.5
	2 PARITY	12	8	0.5
	3 PARITY	24	16	0.5
	4 PARITY	36	24	0.89

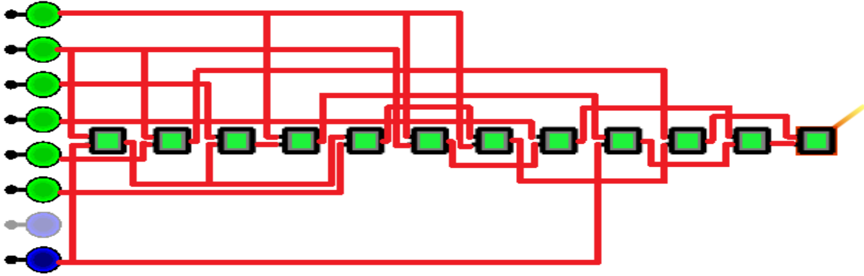


Fig. 2. An evolved IMP phenotype that solves the 6MUX problem. Blue inputs are constant, constant "true" is not used. The rightmost node is the output.

6 Robot Control

We now demonstrate the same approach, this time solving a multi-step control problem in which a robot must navigate obstacles to reach a light source over a number of time steps. The chosen robotics simulator was Webots [14].

6.1 Environmental Setup

Agent. The agent was a simulated Khepera II robot with eight light sensors and eight IR distance sensors; three of each sensor type were used as input (sensors at positions 0, 2 and 5 as shown in Fig. 3(a)). At each step (64ms in simulation time), the agent sampled its light and IR sensors, whose scaled response values ranged from 0 (no light / no object detected) to 1 (fully illuminated / object very close). To make this continuous-valued input amenable to the CGP representation, each of the 6 sensors was encoded as two-bit binary (sensor value

<0.25 = 00, between 0.25 and 0.5 = 01, 0.5 to 0.75 = 10 and >0.75 = 11). Two further input nodes carry constant 1 and 0 signals as before. Three actions were possible: forward, (1,1 or 0, 0 at the two output nodes) and continuous turns to both the left (0, 1) and right (1, 0). Additionally, two bump sensors were added to the front-left and front-right of the agent to prevent it from becoming stuck against an object. If either bump sensor was activated, an interrupt was sent causing the agent to reverse 10cm and the agent to be penalised by 10 steps.

Environment. The agent was randomly located within a walled arena which it could not leave, with coordinates ranging from [-1,1] in both x and y directions. The agents start position was constrained so that an obstacle was initially always between the agent and the light source (initial x+y < -1.5), forcing the agent to learn obstacle avoidance behaviour in addition to phototaxis. Adding to the complexity of the environment, a three-dimensional box was placed centrally in the arena, with vertices on ground level at (x=-0.4, y=-0.4), (-0.4, 0.4), (0.4,0.4), and (0.4, -0.4), and raised to a height of z=0.15. A light source, modelled on a 15 Watt bulb, was placed at the top-right hand corner of the arena (x=1, y=1), which the agent must navigate to. The environment is shown in Fig. 3(b). When the agent reached the goal state (where x+y >1.6), the responsible graph received a fitness bonus of 2500, which was added to the fitness function f (7).

$$f = \frac{1}{1.6 - (|c_x - c_y|)} * 1000 - ts \tag{7}$$

The denominator in the equation expresses the difference between the position of the goal state (1.6) and the current agent position (c_x and c_y), and st is the number of timesteps taken to solve. The minimum value of this function is capped so that $f > 0$. Optimal performance gives f=11800, which corresponds to 700 steps from start to goal state with no collisions.

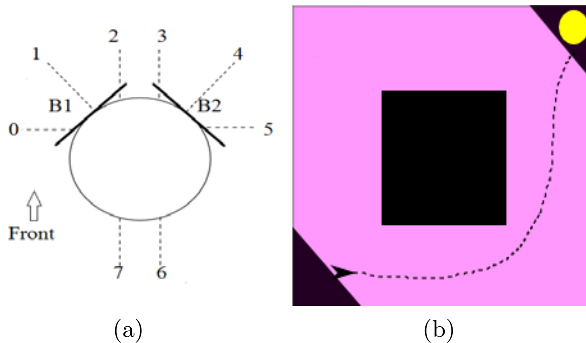


Fig. 3. (a) Khepera sensory arrangement. 3 light sensors and 3 IR sensors share positions 0, 2 and 5. Bump sensors B1 and B2 are attached at 45 degree angles to the front-left and front-right of the robot.(b) The test environment. The agent begins in the lower-left and must reach a light source (circle) in the upper-right, circumnavigating the central obstacle. An example agent path is shown (dotted line).

Table 4. Results of evolving IMP and NAND based CGP logic circuits on the robotics problem. Node and SA parameter statistics taken from successful runs only.

	Perf	Avg. fit	Solved	Avg. nodes	Min. nodes	μ	ψ
IMP	331.9	9288.55	8	32.25	16	0.018	0.06
NAND	118.6	11782.44	10	38.6	18	0.026	0.07

Procedure. A trial began with the placement of the agent in the arena and ended with either (a) location of the reward or (b) a time out after 4000 steps. Each step of the trial consisted of the receipt of the current state at the program inputs and ended with performance of an action based on the states of the output nodes. After all initial networks were trialed, 1000 generations of GA application took place; the systems' state was stored every 20 generations. Newly-generated networks were trialed on the test problem, replacing a member of the population as before.

6.2 Results

As the robot's start location was tightly constrained, we were able to compare "Performance", defined as the first generation in which any graph in the population found the goal state. "High Fitness" was altered to use (7), no other changes were made.

Evolvability. In terms of evolvability, 10/10 NAND runs and 8/10 IMP runs solved the problem. Combined with results seen in Table 4 confirms the general pattern seen in the previous experiments; IMP is harder to evolve than NAND. It is interesting to note that, for the solved cases, none of the differences are statistically significant in the robotics task (performance p=0.127, mean high fitness p=0.143, mean average fitness p=0.142), in other words circuit performance and fitnesses are less divergent. Low average fitness is observed for IMP as two of the runs failed. Self-adaptive parameters do not vary between the circuits (p=0.612 for μ, p=0.74 for ψ), although statistically significant differences are observed between the two mutation rates within the same same circuit type (p=0.048 in IMP, p=0.034 in NAND).

Table 5. Cost, time taken and efficiency of the best evolved IMP and NAND based CGP logic circuits for the robotics test problem

	Min. Cost	Min. T	Efficiency
IMP	32	16	2.53
NAND	54	36	0.40

Solution Size and Efficiency. As before, we use equations 4, 5 and 6 to calculate the efficiency and cost of the best evolved circuit of each gate type. Table 5 shows that IMP circuits are yet again more efficient than NAND. An efficiency ratio of 2.53 in favour of IMP is the highest observed, surpassing the

previous best ratio of 2 observed in 6 MUX, 6 DEMUX and 2- and 3- PARITY. It is worth noting that physical NAND circuits will require more processing steps to run than their IMP counterparts.

7 Conclusions

In this paper we have compared the evolvability and efficiency of two possible implementations for memristive logic circuits. It has been shown across all problems considered that IMP is more efficient than NAND in terms of the number of components required to implement the best evolved circuits. The efficiency ratio between the two types increases for the robotics problem. SA mutation allows for circuits to be successfully evolved for two different problem types without having to manually tune GA parameters. We also show that the GA has more difficulty working with IMP gates, in other words NAND is more evolvable than IMP. However, any physical implementation can be run offline for many generations before realisation as a circuit so this shortcoming is somewhat mitigated. The onus is therefore firmly on solution size rather than the number of generations required to generate working solutions. Future research will focus on evolving more complex, low-energy embedded controllers where the statefulness of the memristive circuits can be utilised.

References

1. Borghetti, J., Li, Z., Straznicky, J., Li, X., Ohlberg, D.A.A., Wu, W., Stewart, D.R., Williams, R.S.: A hybrid nanomemristor/transistor logic circuit capable of self-programming. Proceedings of the National Academy of Sciences 106(6), 1699–1703 (2009)
2. Borghetti, J., Snider, G.S., Kuekes, P.J., Yang, J.J., Stewart, D.R., Williams, R.S.: 'Memristive' switches enable 'stateful' logic operations via material implication. Nature 464(7290), 873–876 (2010)
3. Chua, L.: Memristor-the missing circuit element. IEEE Transactions on Circuit Theory 18(5), 507–519 (1971)
4. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1989)
5. Harding, S., Miller, J.F., Banzhaf, W.: Self modifying cartesian genetic programming: Parity. In: Tyrrell, A. (ed.) 2009 IEEE Congress on Evolutionary Computation, May 18-21, pp. 285–292. IEEE Computational Intelligence Society, IEEE Press, Trondheim, Norway (2009)
6. Harding, S., Miller, J.F.: Evolution of Robot Controller Using Cartesian Genetic Programming. In: Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 62–73. Springer, Heidelberg (2005)
7. Ho, Y., Huang, G.M., Li, P.: Nonvolatile memristor memory: device characteristics and design implications. In: Proceedings of the 2009 International Conference on Computer-Aided Design, ICCAD 2009, pp. 485–490. ACM, New York (2009)
8. Holland, J.H.: Adaptation. In: Rosen, R., Snell, F.M. (eds.) Progress in Theoretical Biology IV, pp. 263–293. Academic Press, New York (1976)

9. Howard, G.D., Gale, E., Bull, L., de Lacy Costello, B., Adamatzky, A.: Evolution of plastic learning in spiking networks via memristive connections. IEEE Transactions on Evolutionary Computing (to appear)
10. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
11. Kuekes, P.J., Stewart, D.R., Williams, R.S.: The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. Journal of Applied Physics 97(3), 034301 (2005)
12. Lehtonen, E., Poikonen, J., Laiho, M.: Two memristors suffice to compute all boolean functions. Electronics Letters 46(3), 230–231 (2010)
13. Mead, C.: Neuromorphic electronic systems. Proceedings of the IEEE 78(10), 1629–1636 (1990)
14. Michel, O.: Webots: Professional mobile robot simulation. International Journal of Advanced Robotic Systems 1(1), 39–42 (2004)
15. Miller, J.F.: Cartesian Genetic Programming. Springer, Heidelberg (2011)
16. Miller, J.F.: Digital filter design at gate-level using evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999, pp. 1127–1134. Morgan Kaufmann (1999)
17. Miller, J.F.: An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, July 13-17, vol. 2, pp. 1135–1142. Morgan Kaufmann, Orlando (1999)
18. Miller, J.F., Smith, S.L.: Redundancy and computational efficiency in cartesian genetic programming. IEEE Transactions on Evolutionary Computation 10(2), 167–174 (2006)
19. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
20. Rajaei, A., Houshmand, M., Rouhani, M.: Optimization of Combinational Logic Circuits Using NAND Gates and Genetic Programming. In: Gaspar-Cunha, A., Takahashi, R., Schaefer, G., Costa, L. (eds.) Soft Computing in Industrial Applications. AISC, vol. 96, pp. 405–414. Springer, Heidelberg (2011)
21. Rechenberg, I.: Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog, Stuttgart (1973)
22. Snider, G.: Computing with hysteretic resistor crossbars. Applied Physics A: Materials Science and Processing 80, 1165–1172 (2005)
23. Strukov, D.B., Snider, G.S., Stewart, D.R., Williams, R.S.: The missing memristor found. Nature 453, 80–83 (2008)

A New, Node-Focused Model for Genetic Programming

David Jackson

Department of Computer Science, University of Liverpool
Liverpool L69 3BX, United Kingdom
djackson@liverpool.ac.uk

Abstract. We introduce Single Node Genetic Programming (SNGP), a new graph-based model for genetic programming in which every individual in the population consists of a single program node. Function operands are other individuals, meaning that the graph structure is imposed externally on the population as a whole, rather than existing within its members. Evolution is via a hill-climbing mechanism using a single reversible operator. Experimental results indicate substantial improvements over conventional GP in terms of solution rates, efficiency and program sizes.

Keywords: Genetic programming, Graph-based representation.

1 Introduction

In genetic programming (GP), a variety of representations are available to encode the programs representing individuals of the population. Most commonly, programs are stored as tree structures [1], in which leaf nodes are taken from a set of terminals, and internal nodes are drawn from a set of functions appropriate to the problem. Evolutionary operators work on such representations by swapping subtrees or replacing them with new, randomly-generated subtrees.

In linear GP [2] the representation has no such structure imposed on it. Instead, programs are simply sequences of individual instructions. Whereas tree-based GP takes a functional view of programs, in which calculations are passed up a tree as it is evaluated, linear GP is more akin to conventional imperative programming, with intermediate and final results being stored in registers or memory variables. The representation language used can be either real or abstract: when machine code instructions are directly manipulated, the efficiency gains can be substantial [3].

A tree is merely one form of a graph, and so it is perhaps not surprising that it is not the only such graph structure that has been tried for GP. One of the first systems to explore this was PADO (Parallel Algorithm Discovery and Orchestration) [4]. PADO makes use of stack memory and indexed memory, and a graph may contain action nodes and branch-decision nodes. The system was used to evolve parallel programs for classifying images.

Taking inspiration from the parallel processing performed in neural networks, Poli's PDGP (Parallel Distributed GP) [5] uses a grid representation to hold graph-structured programs. Individuals are still subject to (suitably modified) crossover and

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 49–60, 2012.
© Springer-Verlag Berlin Heidelberg 2012

mutation, but programs are more compact than tree-based equivalents, and offer opportunities for concurrent execution. A similar grid-based approach is employed in Cartesian Genetic Programming (CGP) [6], in which the number of rows and columns, and the amount of feed-forward, are all parameters to the system. Originally developed to evolve digital logic designs, the approach made use exclusively of mutation to generate new candidates which took part in a (1+λ) evolutionary strategy, but more recent research has explored the advantages of a new crossover operator [7]. In the GRAPE (GRAph structured Program Evolution) approach [8], graphs contain arbitrarily directed links, and both calculations and node sequencing are determined by a separate data set.

Other researchers have taken conventional tree-based or linear GP and augmented them with additional structures. Often, this is done as a way of introducing modules or other forms of hierarchy into programs [1,9-11]. In linear-tree GP [12], each node of a tree consists of a linear program and a branching node which determines the next node in the tree to be executed. The idea was later extended to more general graph structures [13]. In the MIOST system [14], program trees may contain additional links both to provide more sophisticated interaction between nodes and also to allow multiple outputs from individuals.

In Multi-Expression Programming (MEP) [15,16], each individual has a structure similar to that of single-row CGP, with each node of the graph having links to operands further back in the graph. The main difference is that execution results are computed not only for a program graph as a whole, but also for each of its sub-graphs. The overall fitness of the individual is defined to be the fitness of the best sub-expression. Mutation and crossover are the primary evolutionary operators.

In our own proposed approach, which we call Single Node Genetic Programming (SNGP) we take this idea of associating a fitness with every node a stage further. In our model, every individual in the population has just one fitness value, rather than several as in MEP, but that is because each individual consists of only one node. Another key difference between our approach and others is that individuals are not entirely distinct: they are interlinked in a graph structure similar to that of MEP or CGP, with population members acting as operands of other members. In a sense, then, an entire SNGP population can be viewed as being very similar to a single MEP individual, although, as we shall see, the mechanics of evolution are very different.

2 The SNGP Model

An SNGP population is a set of N members

$$M = \{m_0, m_1, \ldots, m_{N-1}\}.$$

Each member is a tuple of the form:

$$m_i = < u_i, r_i, S_i, P_i, O_i >$$

where:

$u_i \in \{T \cup F\}$ is a single graph node taken from either the function set F or the terminal set T of the problem;

r_i is the rating of fitness for the individual;

S_i is a set of successors of this node;
P_i is a set of predecessors of the node;
O_i is a vector of outputs generated when this node is evaluated.

During initialisation, the population is partitioned in such a way that:

$u_i \in T$ if $i < TNUM$
$u_i \in F$ otherwise

where *TNUM* is the number of terminals in the terminal set. Moreover, for any u_i, u_j such that $i, j < TNUM$ and $i \neq j$, we have $u_i \neq u_j$.

In other words, the first *TNUM* members of the population are initialised to represent the members of the terminal set, with each terminal appearing exactly once. All other members contain nodes drawn from the function set. These are allocated at random, and so may be replicated in the population.

For a population member which represents a function, the operands of that function are drawn from other members of the population. The successor set of the node is a list of the population members acting as operands, represented by their position in the population. We make the restriction that for each $s \in S_i$ we have $0 \leq s < i$, i.e. the operands of a function must be 'lower down' in the population (towards position zero).

Similarly, the predecessors of an individual are those population members for which the individual is used directly as an operand, i.e. they take us to the next higher expression level. This means that for each $p \in P_i$ we have $i < p < N$.

Note that for terminal nodes the successor sets are empty. Moreover, as these nodes cannot change during evolution (see later), their predecessor sets are not needed and are also left empty.

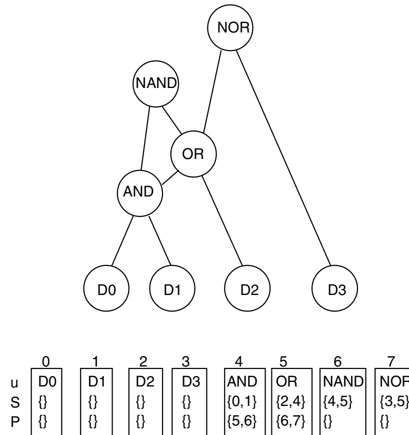


Fig. 1. Small (8-member) SNGP population and corresponding graph structure

Fig. 1 shows how a population of just 8 members might be initialised, together with the corresponding graph. The first four positions in the population are occupied

by terminals, the remainder by functions. For ease of explanation the functions shown here are all different, although in reality functions could be replicated, and certainly will be with larger population sizes. Note that the AND node and the OR node both have two predecessors, i.e. they appear as immediate operands of two other function nodes. This form of reuse is characteristic of SNGP programs, and therefore differs from conventional tree-based GP.

The graph shown here contains eight different expressions, one per node. The simplest expressions are the single-node terminals: D0, D1, D2 and D3. The other expressions are those rooted at the remaining nodes:

AND(D0, D1)
OR(AND(D0, D1), D2)
NAND(AND(D0, D1), OR(AND(D0, D1), D2))
NOR(OR(AND(D0, D1), D2), D3)

It can be seen that, even with only eight nodes, a range of reasonably complex expressions can be encoded. This complexity can rise dramatically when hundreds of nodes are used.

Key to the efficiency of SNGP is the way in which the fitness of each individual is calculated. This is achieved using a form of dynamic programming. During initialisation, each terminal is evaluated across all test input cases, and the outputs generated are stored in O_i. These outputs are used to calculate the fitness values r_i. As initialisation continues, and each randomly selected function is inserted into the population, outputs and fitnesses continue to be computed, but making use of the values already stored for the operands forming the successor set. In this way, the fitness calculation for an individual is highly efficient, involving the application of only one operator or function per test case.

In SNGP there is only one evolutionary operator, called *smut* (successor mutate). The way that *smut* works is that a member of the population is chosen at random, and then one of its operands (i.e. a member of its successor set) is replaced by a reference to a different member of the population (but still lower down in the position order). Figure 2 shows how this might work for the small program graph that was given previously in Figure 1.

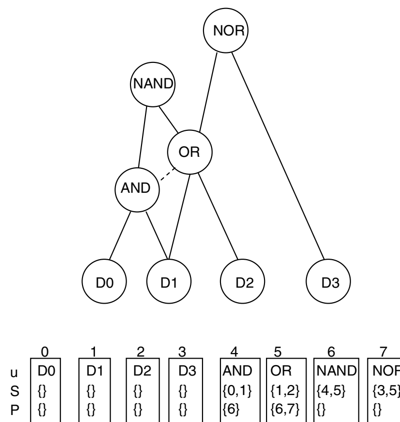


Fig. 2. Effects of the *smut* operator on the population

Here, the first operand of the OR node is being changed from population member number 4 (the AND node) to member number 1 (the terminal D1). Hence, the successor set of node 5 must be changed to reflect this, and node 5 must therefore be deleted from the predecessor set of the AND node. In this example, the new operand is a terminal, and so nothing more needs to be done to the graph structure; when the new operand is a function, its predecessor set must also be updated to add in the new parent.

A modification such as this means that the individual which has been changed must be re-evaluated to determine its new outputs and fitness rating. In our example, the expression OR(D1, D2) must be computed for all test cases. However, this will also have an effect on individuals higher up in the population. Exactly which individuals are affected is determined by the predecessor sets. In Figure 2, the predecessors of the OR node are the NAND node and the NOR node, and so these must be re-executed. In larger graphs, it may be necessary to continue this chain of execution by pursuing the predecessor references until all affected individuals have been re-assessed.

The order in which evaluations proceed up the population can have a great impact on efficiency. Returning to Figure 1, a change to the operands of the AND node might cause the immediate predecessors NAND and OR to be evaluated next. Then, because the OR outputs have changed, the NAND node might be invoked once again. In general, there may be many unnecessary evaluations that take place before the population eventually settles to its final values. To circumvent this, we implement a mechanism in which the predecessor sets are followed to build an ordered 'update list' of all affected individuals. We then execute each member of the list in turn, from the lowest to the highest position in the population, thus ensuring that no node is visited more than once.

Evolution of an SNGP population is driven using a hill-climbing approach. This is based on fitness measurements across the whole population, rather than on single individuals. More formally, and assuming that lower fitness values are better, the aim is to minimize Σr_i. Whenever the *smut* operator is applied, this summation is computed. If the result is worse than the value of the summation before *smut* was invoked, and if no solution has been found, then the modifications made by *smut* are reversed. To make this more efficient, the old outputs and fitness values of each member of the update list are recorded by *smut*, so that they can be put back in place if necessary by a single *restore* operation. Note that worse overall fitness will cause the reversal to be activated even if it means discarding individuals that are fitter than any of those previously present in the population.

3 Experimentation

In evaluating SNGP against conventional GP we have made use of three standard benchmark problems: 6-multiplexer, even-parity, and symbolic regression.

In the 6-mux problem, the aim is to evolve a program that interprets the binary value on two address inputs (A0 and A1) in order to select which of the four data inputs (D0-D3) to pass onto the output. The function set is {AND, OR, NOT, IF}. Fitness evaluation is exhaustive over all 64 combinations of input values, with an

individual's fitness being given in terms of the number of mismatches with expected outputs.

In the even parity problem we search for a program which returns TRUE if the number of inputs set to logic 1 is even, and FALSE otherwise. The function set is {AND, OR, NAND, NOR}. In this paper, we will first consider the 4-input version of the problem, with fitness being given in the range 0-16; later, we will discuss higher level parity problems.

Our final problem is symbolic regression of a polynomial. In our version of this, the polynomial we attempt to match through evolution is $4x^4 - 3x^3 + 2x^2 - x$. The only terminal is x, and the function set is {+, -, *, /}, with the division operator being protected to ensure that divide-by-zero does not occur. The fitness cases consist of 32 x-values in the range [0,1), starting at 0.0 and increasing in steps of 1/32, plus the corresponding y-values. Fitness is calculated as the sum of absolute errors in the y-values computed by an individual, whilst success is measured in terms of the number of 'hits' – a hit being a y-value that differs from the expected output by no more than 0.01 in magnitude.

The problem parameters as they apply to the use of standard GP in all these problems is given in Table 1. For SNGP there are really only two parameters. The first is the population size (number of nodes), which we have arbitrarily set to 100, although later we will discuss the effects of altering this. The second parameter is the 'length' of a run, which we will refer to as *L*. SNGP does not have generations as such; we can think instead in terms of the number of evolutionary operations performed. Since standard GP with a population size of 500 running over 50 generations generates 25,000 individuals via crossover or reproduction, we will set the upper limit on the number of *smut* applications to 25,000. Again, however, we will examine the effects of changing this parameter.

Table 1. GP system parameters common to all experiments

Population size	500
Initialisation method	Ramped half-and-half
Evolutionary process	Steady state
Selection	5-candidate tournament
No. generations	51 generational equivalents (initial+50)
No. runs	100
Prob. crossover	0.9
Mutation	None
Prob. internal node used as crossover point	0.9

In comparing SNGP with conventional GP we consider three factors: solution rate, efficiency, and solution size. The solution rate is given simply in terms of the number of solutions to the problem found in 100 runs. Comparisons of efficiency can be a little more difficult to make fair. For example, it would be possible to compare the number of fitness evaluations performed in each case. However, the nature of a fitness

evaluation performed in standard GP is so different from that of SNGP that comparing the two becomes meaningless. Instead, we use the more uniform measure of standard wall-clock time. The timings we give are for a PC with an Intel Core i7 quad-core processor running at 2.8GHz. The GP systems are compiled using Microsoft Visual Studio as single-threaded processes executing under identical load conditions. Each timing figure is for the number of seconds required to perform 100 runs. Table 2 summarises the results for our three problems.

Table 2. Comparison of SNGP with standard GP

	Even-4		**6-mux**		**Regression**	
	GP	**SNGP**	**GP**	**SNGP**	**GP**	**SNGP**
Soln. rate (%)	14	95	66	100	11	43
Time 100 runs (secs)	16	7	15	7	59	21
Av. soln size	278	38	83	31	223	24
Max soln size	709	56	449	54	1011	48
Min soln size	59	22	10	15	29	12

We can see from this table that SNGP substantially out-performs conventional GP. For the even-4 parity problem, SNGP finds almost seven times as many solutions as standard GP, and in less than half the time. Standard GP improves on the multiplexer problem, but now SNGP finds a solution on every run, again in less than half the time. For the symbolic regression problem, SNGP discovers four times as many solutions in one third of the time of standard GP. It should be pointed out that all the comparative performance figures in this paper have been established as statistically significant using a t-test at the 95% confidence level.

Turning our attention to the program sizes in Table 2, we see that SNGP again comes off best. The explanation for this is of course that, with the parameters we have used, it is impossible for the SNGP programs to grow beyond 100 nodes, whereas standard GP is effectively unbounded. Although it could be argued that this frees standard GP to explore a larger search space, it is clear that it does not hamper SNGP's ability to find solutions. Conversely, standard GP would find it very difficult to find solutions when using such small population sizes.

The only figure that could be regarded as a victory for conventional GP in Table 2 is that of the minimum solution size found for the multiplexer problem. Standard GP finds the following 10-node solution:

IF (A1 IF (A0 D3 IF (A0 D1 D2)) (IF (A0 D1 D0)))

whilst the best that SNGP can find is the following 15-node solution:

```
26: IF 5 19 7
19: IF 4 3 18
18: AND 2 16
16: IF 6 14 10
14: OR 7 5
10: OR 9 4
```

9: OR 0 5
7: IF 4 1 0
6: NOT 5
5: A1
4: A0
3: D3
2: D2
1: D1
0: D0

In this linearised form of the program graph, each line represents a node, and the number at the beginning of the line is the index number of the node in the population. As can be seen, population members 0 to 5 represent the terminals of the problem. The way to interpret the graph as a program is to read it in a top-down fashion. The first node (node 26 in this example) is the one at which the solution is rooted. Hence, the top-level function in this program is an IF construct which will test the value of A1 (node 5); if A1 is true, execution will be directed to node 19 (another IF statement), otherwise it will go to node 7 (also an IF statement, corresponding to 'IF A0 then D1 else D0').

That said, the sizes of the programs evolved by SNGP are constrained by the population size. This suggests that reducing the population size might have the effect of encouraging the evolution of smaller programs. However, the question is by how much we can do this while still providing the evolutionary process with enough genetic material to discover solutions. The graph of Figure 3 charts what happens when the population size is altered for the 6-mux problem. Similar graphs are obtained for the symbolic regression and parity problems.

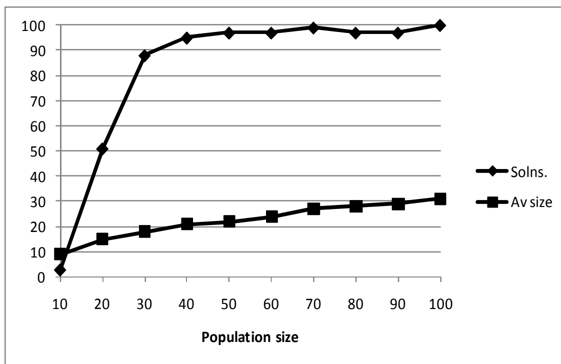


Fig. 3. 6-mux soln rate and av. soln size for varying population sizes

We can see that, as the population size is gradually reduced, we do indeed see the desired reduction in the average solution size. As might be expected, the number of solutions obtained also decreases, but what is surprising is that solutions continue to be found even when the population size becomes quite small. For the 6-multiplexer problem, we require only 10 individuals (and therefore just 10 nodes across the whole

population) to discover solutions, whilst anything greater than 40 individuals is enough to give us solution rates close to the 100% level. Similarly, 20 individuals are sufficient to evolve solutions for the symbolic regression problem, and 30 individuals for the even-4 parity problem.

The population size parameter therefore acts as a useful mechanism for tuning the results we wish to obtain from evolution. Higher values give us lots of solutions; lower values provide fewer solutions, but they are smaller in size and can be generated much quicker. When just 10 individuals are used in the 6-mux problem, we get a solution rate of just 3%. However, the solutions we obtain can be no bigger than 10 nodes (because they are bounded by the population size), and the complete set of 100 runs executes in only 2 seconds.

Hence, the inability of SNGP to match standard GP on minimum program size for 6-mux in our earlier experiments is easily remedied. With a population size of 10, we obtain two solutions of size 9, and one of size 10. In general, the extensive re-use of nodes via multiple pointers in SNGP enables much more compact solutions than the conventional GP equivalents. For example, one 14-node SNGP solution to the 6-mux problem takes up 59 nodes when written out as a standard expression tree such as would be used in standard GP.

The other SNGP parameter – the maximum number of operations per run (L) – can, of course, also be altered. In conventional GP, increasing the duration of runs generally has little effect: populations which converge without finding a solution usually do not recover no matter how much extra evolutionary time they are granted. In contrast, our observations of SNGP runs suggest that many runs are still continuing to evolve when they are terminated. The parameters we have used thus far for SNGP have been sufficient to obtain a 100% solution rate for the 6-mux problem. Table 3 shows what happens in the even-parity and regression problems if the maximum length of each run is increased from 25,000 to 100,000 operations.

Table 3. Effects on even-4 parity and symbolic regression problems when SNGP max run length increased to 100,000 ops

	Even-4	**Regression**
Soln. Rate (%)	99	54
Time 100 runs (secs)	10	61

The results we have presented above are encouraging for these simple benchmark problems, but suppose we increase the problem difficulty? A simple and effective way of investigating this is to apply SNGP to the solution of higher-order parity problems, which are known to be difficult for conventional GP to solve. Table 4 presents the results of these experiments, with N kept at 100 and L maintained at 25,000. By way of contrast, conventional GP is unable to find any solutions to the even-5 parity problem, even when the population size is increased to 4,000 (equivalent to 200,000 evolutionary operations over 50 generations) [1].

Table 4. Performance of SNGP for higher-order even parity problems (N=100, L=25,000)

	Even-5	Even-6	Even-7
Soln rate (%)	58	14	2
Time 100 runs (secs)	47	67	82
Av. soln size	48	52	58
Max. soln size	67	64	59
Min. soln size	31	40	58

The results presented here for even parity also compare well against those reported for Multi-Expression Programming (MEP) [16]. For MEP populations up to size 500, each member having 200 genes, the best success rate for even-4 parity is less than 50% (cf. SNGP's rate of 95-99%). Similarly, MEP with a population of 1000 individuals having 600 genes each has a success rate of just 16.66% for even-5 parity.

It should also be borne in mind that the SNGP results have been obtained using our standard parameter values. As before, these can be manipulated to tune the solution rate, solution sizes, and speed of solution discovery. For example, halving the population size to 50 for the even-5 parity problem provides us with only 19 solutions in 100 runs, but execution of this set of runs is completed in just 13 seconds. At the other end of the scale, increasing *L* to 100,000 operations whilst keeping *N* at 100 leads to a 12% solution rate for the even-7 parity problem, found in just over 11 minutes. Moving up to even-8 parity, the parameter values *N*=200 and *L*=200,000 gave us 2 solutions in the 5 runs we attempted, one with 100 nodes, the other with 110. In the other 3 runs, the best programs found had fitness values of either 2 or 3.

4 Conclusions

In this paper we have introduced a new graph-based model for genetic programming. It has several key differences from existing representations, perhaps the most striking being that each individual in the population consists of just a single program node. Moreover, the graph structure to which we refer is not contained within individuals, as it is in most other representations, but is external to them: it is imposed on the population as a whole, linking them into a network of functions and their operands. Indeed, it could be argued that an SNGP population is not a collection of members at all, but merely a single individual. However, the complexity of each node, comprising not only the function to be applied, but also its predecessor and successor sets, its output values, and in particular its own identifiable fitness value, makes it worthy of consideration as a distinct individual. This is really just a question of semantics; what really matters is whether the approach has any merit.

Another important difference is the way in which evolution is carried out. We do not use crossover, and the form of mutation we employ is non-standard in the sense that the function or terminal held at a particular node never alters: once a population has been randomly initialized, each node will retain its given operation for the life-time of the run. What does change are the references to other individuals acting as operands of a given function. In this way, we view the dynamics of SNGP evolution

as a search for an optimal set of connections between functions and terminals which are present in sufficient number to solve the problem at hand.

Evolution in SNGP is a lot more collaborative and altruistic than it is in other approaches. The hill climbing approach we use is based on the good of the whole population, not just individuals. If a given operation does not lead to better fitness across the whole population, then its actions are reversed, even if there exist particular individuals which would have benefited greatly from the change. The only exception to this is when the operation leads to a solution being discovered.

Despite the simplicity of its representation, SNGP copes extraordinarily well with the benchmark problems we have thrown at it. Its solution-finding performance is superior in terms of both the numbers of solutions obtained and in the times taken to discover them. It readily finds solutions to higher-order parity problems that are beyond the reach of conventional GP and many other approaches, and it does so using populations that are comparatively minute.

A further advantage to be gained is that these solutions are significantly smaller than those evolved in other approaches. A key factor in this compactness is the ability to re-use nodes as operands of numerous individuals simultaneously, effectively converting them into program modules.

A serious problem in conventional GP is that of bloat – the rapid explosion in program sizes as evolution proceeds. SNGP does not have this problem. This is partly due, of course, to the restrictions on program growth imposed by a fixed population size. But it is also because SNGP does not have the equivalent notion of introns, which are often a major contributing factor in bloat. In SNGP, every node is evaluated, and therefore has its own intrinsic worth in addition to the value it may offer in its role as an operand of other individuals.

All of these findings are highly encouraging. There is, however, still much research to be done on SNGP. Some of the questions which immediately jump to mind include: What are the dynamics of SNGP, such that it is able to find solutions so readily with such small populations? What is the potential for exploiting the parallelism inherent in both the SNGP system and in the programs it evolves? Are we using the best evolutionary operator, or should it be modified or others introduced? Are we making too much of randomness in the initialization phase and in the way changes are made during evolution? How well does SNGP deal with problems in which functions have side-effects and which are therefore not amenable to dynamic programming?

We hope to address these questions and others in future work.

References

1. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
2. Brameier, M., Banzhaf, W.: Linear Genetic Programming. Springer, Heidelberg (2007)
3. Nordin, P., Banzhaf, W., Francone, F.D.: Efficient Evolution of Machine Code for CISC Architectures Using Instruction Blocks and Homologous Crossover. In: Spector, L., et al. (eds.) Advances in Genetic Programming, vol. 3, pp. 275–299. MIT Press, Cambridge (1999)

4. Teller, A., Veloso, M.: PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System. Technical Report CS-95-101, Department of Computer Science, Carnegie-Mellon University, USA (1995)
5. Poli, R.: Parallel Distributed Genetic Programming. In: Corne, D., et al. (eds.) New Ideas in Optimization, pp. 779–805. McGraw-Hill Ltd., UK (1999)
6. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
7. Clegg, J., Walker, J.A., Miller, J.F.: A New Crossover Technique for Cartesian Genetic Programming. In: Thierens, D., et al. (eds.) Proc. Genetic and Evolutionary Computing Conf (GECCO 2007), London, England, UK, pp. 1580–1587 (2007)
8. Shirakawa, S., Ogino, S., Nagao, T.: Graph Structured Program Evolution. In: Thierens, D., et al. (eds.) Proc. Genetic and Evolutionary Computing Conf. (GECCO 2007), London, England, UK, pp. 1686–1693 (2007)
9. Angeline, P.J., Pollack, J.: Evolutionary Module Acquisition. In: Proc. 2nd Annual Conf. on Evolutionary Programming, La Jolla, CA, pp. 154–163 (1993)
10. Jackson, D.: The Performance of a Selection Architecture for Genetic Programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 170–181. Springer, Heidelberg (2008)
11. Rosca, J.P., Ballard, D.H.: Discovery of Subroutines in Genetic Programming. In: Angeline, P., Kinnear Jr., K.E. (eds.) Advances in Genetic Programming, ch. 9, pp. 177–202. MIT Press, Cambridge (1996)
12. Kantschik, W., Banzhaf, W.: Linear-Tree GP and Its Comparison with Other GP Structures. In: Miller, J., Tomassini, M., Lanzi, P.L., Ryan, C., Tetamanzi, A.G.B., Langdon, W.B. (eds.) EuroGP 2001. LNCS, vol. 2038, pp. 302–312. Springer, Heidelberg (2001)
13. Kantschik, W., Banzhaf, W.: Linear-Graph GP - A New GP Structure. In: Foster, J.A., Lutton, E., Miller, J., Ryan, C., Tettamanzi, A.G.B. (eds.) EuroGP 2002. LNCS, vol. 2278, pp. 83–92. Springer, Heidelberg (2002)
14. Galvan-Lopez, E.: Efficient Graph-Based Genetic Programming Representation with Multiple Outputs. International Journal of Automation and Computing 5(1), 81–89 (2008)
15. Oltean, M.: Evolving Digital Circuits using Multi-Expression Programming. In: Zebulum, R.S., et al. (eds.) Proc. 2004 NASA/DoD Conf. on Evolvable Hardware, Seattle, USA, pp. 87–97 (2004)
16. Oltean, M.: Solving Even-Parity Problems using Multi-Expression Programming. In: Chen, C., et al. (eds.) Proc. 7th Joint Conf. on Information Sciences, North Carolina, USA, vol. 1, pp. 295–298 (2003)

Medial Crossovers for Genetic Programming

Krzysztof Krawiec

Institute of Computing Science, Poznan University of Technology, Poznań, Poland
krawiec@cs.put.poznan.pl

Abstract. We propose a class of crossover operators for genetic programming that aim at making offspring programs semantically intermediate (medial) with respect to parent programs by modifying short fragments of code (subprograms). The approach is applicable to problems that define fitness as a distance between program output and the desired output. Based on that metric, we define two measures of semantic 'mediality', which we employ to design two crossover operators: one aimed at making the semantic of offsprings geometric with respect to the semantic of parents, and the other aimed at making them equidistant to parents' semantics. The operators act only on randomly selected fragments of parents' code, which makes them computationally efficient. When compared experimentally with four other crossover operators, both operators lead to success ratio at least as good as for the non-semantic crossovers, and the operator based on equidistance proves superior to all others.

Keywords: Genetic programming, Program semantic, Semantic crossover.

1 Introduction

The function of crossover in evolutionary computation is to produce new candidate solutions (offspring) that fuse some of the features of existing candidate solutions (parents). This principle is typically implemented by following the biological paradigm, i.e., by recombining the genotypes of parents. An implicit assumption hidden in this line of reasoning is that recombination effects propagate to the phenotypes in an analogous way, and the offspring is expected to *behave* in a way that mixes, to some extent, the behaviors of its parents.

This renders invalid if the elements of the genotype do not map one-to-one to the elements of phenotype, which is unfortunately always the case for nontrivial problems. Most problems considered in genetic programming (GP, [1]) belong here too, as the interactions between the elements of genotype (instructions) are usually strong, and influence the output of program (and subsequently its fitness) in a complex way that is hard to predict and model. As a result, an operator that recombines parents' programs cannot be in general expected to always recombine their behaviors.

Trying to circumvent this difficulty, we propose a family of operators, *medial crossovers*, designed to produce programs that inherit not only parts of parents' code, but also some elements of parents' semantics. To this aim, we exploit the

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 61–72, 2012.
© Springer-Verlag Berlin Heidelberg 2012

fact that many genetic programming tasks define fitness using a metric that compares the actual program behavior with the desired one. In the proposed operators, that very metric is employed to alter subprograms within parents' code so that their outcome at certain execution stage become more similar to each other, which indirectly affects the semantic of entire offspring programs in a way that gives chance to exploit certain properties of the fitness landscape.

Although research on semantic in GP intensified only recently, there are a few related contributions. McPhee *et al.* were probably one of the first to study the impact of crossover on program semantics and so-called semantic building blocks [2]. In [3], Moraglio *et al.* considered properties of semantic spaces for different metrics and provided guidelines for designing semantically geometric crossovers. The semantically-aware crossover by Quang *et al.* [4] swaps a pair of subtrees in parent solutions that have similar, yet not too similar, semantics. To author's knowledge however, this is the first study on semantic crossovers that act in a semantically medial way on the level of subprograms.

2 Metric-Based Crossover Operators

We consider here the class of problems for which the objective function captures the divergence between the program output and the desired output, given as a part of problem formulation. This class embraces the prevailing part of GP applications, where individuals are typically tested on a set of fitness cases, and fitness is some form of error built upon the outcome of these tests (cf. symbolic regression and boolean function synthesis).

Formally, we assume that the (minimized) fitness of a candidate solution p is defined as $f(p) = ||p, t||$, where t is the (known) desired output (target) determined by *problem instance*, and $|| \, ||$ is a metric measures the distance between t and the output produced by p. The metric imposes a structure on the set of programs (potential solutions), turning it into a *space*. It is essential to emphasize that, throughout this paper, the metric operates on the phenotypic level, ignoring program code (syntax) and taking into account only its *output*, which within this paper we identify with *program semantics* [2,5,6]. We call a fitness function defined in this way *metric-based fitness function*.

Crossover is a search operator that produces a new search point (offspring) o (or a pair of offspring) based on a pair of search points p_1, p_2 (parents). In the following, we consider only *nontrivial* offspring $o \neq p_1, p_2$.

The metric allows us to express some properties of the offspring in the context of its parents. An offspring o that fulfills

$$||o, p_1|| + ||o, p_2|| = ||p_1, p_2||, \tag{1}$$

will be referred to as *geometric offspring,* and a crossover operator that produces such offspring *geometric crossover* (a.k.a. topological crossover [7]).

To demonstrate the potential that dwells in geometric operators, let us consider a special case, a geometric crossover for which the distributions of $||o, p_1||$ and $||o, p_2||$ are uniform and the city-block distance as the metric. In such

a case, the expected fitness of offspring is $(f(p_1) + f(p_2))/2$. The sketch of proof is as follows. The metric is a norm here and we switch to vector spaces, so let x^i denote the ith coordinate of point in the space. A geometric offspring o has to fulfill $o^i = \alpha^i p_1^i + (1 - \alpha^i p_2^i)$, where $\alpha^i \in (0, 1)$. Let $d^i(x)$ denote the difference between the location of x and t on coordinate i. Then, $d^i(o) = t^i - o^i = t^i - [\alpha^i p_1^i + (1 - \alpha^i p_2^i)]$. It is easy to show that, because for uniform distribution $\mathbb{E}(\alpha^i) = \mathbb{E}(1 - \alpha^i) = \frac{1}{2}$, it must hold:

$$\mathbb{E}(d^i(o)) = t^i - \mathbb{E}(\alpha^i)p_1^i - \mathbb{E}(1 - \alpha^i)p_2^i = t^i - \frac{p_1^i}{2} - \frac{p_2^i}{2} = \frac{d(p_1) + d(p_2)}{2}$$

Without loss of generality, we can ignore the sign of $\mathbb{E}(d^i(o))$, in which case it becomes the expected contribution to o's distance from t (fitness) on the ith dimension. The expected fitness of o for city-block fitness is then

$$\mathbb{E}(f(o)) = \mathbb{E}(\sum_i d^i(o)) = \sum_i \mathbb{E}(d^i(o)) = \frac{\sum_i d^i(p_1) + \sum_i d^i(p_1)}{2} = \frac{f(p_1) + f(p_2)}{2}$$

Verification of this property for other norms is beyond the scope of this paper. Nevertheless, under all metrics, the offspring cannot be worse than the worse of the parents.

As another important property, let us notice that, under all metrics, the expected fitness of a geometric offspring is minimized when it fulfills also the condition of *equidistance*:

$$||o, p_1|| = ||o, p_2|| \tag{2}$$

Without providing formal proof, let us notice that, under any finite-support distribution of targets t, given random locations p_1 and p_2, a point that is equidistant from them is expected to be the closest to t.

Producing a geometric offspring that is simultaneously equidistant can be difficult (not mentioning the challenge of designing a geometric crossover alone, which we discuss later). The reason for this is twofold: (i) for discrete spaces such a point may not exist, and (ii) a program o with the output that fulfills (2) may not exist (an equidistant semantics cannot be expressed within the assumed programming language). A possible solution to this problem is to relax condition (2) and produce a geometric offspring that is as equidistant as possible. However, a question arises: how much a program should be allowed to diverge from equidistance to be still considered a useful offspring for a particular pair of parents? As $||o, p_1||$ and $||o, p_2||$ diverge from each other, the offspring becomes more and more similar to one of the parents, which makes the search less effective. Note also that, in general, there is no guarantee of existence of even a single nontrivial geometric offspring for a pair of programs.

An analogous problem arises when one takes an alternative path, i.e., relaxing geometricity condition (1) while requiring perfect equidistance (2): again, a perfectly equidistant offspring may simply not exist at all (i.e., even if geometricity is completely ignored), and it is hard to tell what is the acceptable divergence from geometricity.

An offspring that meets even one of the above conditions may therefore not exist. This obliges us to simultaneously relax both of them to design an operator that works in practice. This can be formalized by defining analogous criteria: *divergence from geometricity* d_G and *divergence from equidistance* d_E:

$$d_G(o, p_1, p_2) = ||o, p_1|| + ||o, p_2|| - ||p_1, p_2|| \quad (3)$$
$$d_E(o, p_1, p_2) = \big| \, ||o, p_1|| - ||o, p_2|| \, \big|$$

In the following, we consider operators that attempt to minimize d_G or d_E, which we refer to as (semantically) *medial* crossovers.

An exact implementation of medial crossover is in most cases technically infeasible. The primary cause is the complexity (and, typically, irreversibility) of genotype-phenotype mapping, which makes direct synthesis of an offspring that minimizes one or both of the above criteria impossible or at least computationally intractable. Apart from certain special cases [3], there is no direct way of constructing an offspring program that exhibits intermediate behavior with respect to the behaviors of parents programs (intermediate in the sense of the assumed metric $|| \cdot ||$). Standard crossover operators typically ignore that fact and produce offspring that are intermediate in purely syntactical terms, but this does not translate into analogous intermediacy in the space of program behaviors. For instance, tree-swapping crossover tends to produce offspring that are semantically very different from the parents [8].

In theory, one could consider *all* potential offspring (i.e., all possible programs), and pick the one that minimizes the criterion (criteria). But such procedure is computationally impractical (exponential time complexity w.r.t. program length), and can be only approximated via sampling, which we studied in past [5]. Moreover, if all programs were to be generated *and* run (the latter required to know the program output), then also the optimal solution ($o = t$) would be among them. A search algorithm that has to consider *all* solutions to proceed with a single iteration has limited usefulness, to say the least.

In general then, it is impossible to generate 'mixtures' of parents that are optimal in the sense of d_G or d_E. Can we at least approximate such behavior? We investigate this possibility in the subsequent section.

3 Partially Medial Crossover

In this section, we come up with a family of crossover operators that are based on the introduced criteria and are technically realizable. The key idea is to port the concepts from Section 2, where they applied to entire programs, to *subprograms*. Our hypothesis is that by making programs semantically more similar to each other (*medial*) at intermediate execution steps (*partially*), we have a chance of making them overall more similar.

The definition of subprogram depends on the assumed program representation. For simplicity, we represent here programs as linear sequences of *instructions*, in which case a subprogram is a continuous subsequence of instructions.

Let $p[i..j]$, $i \leq j$ denote the subprogram of program p composed of instructions from i^{th} to j^{th} inclusive. We also assume that concatenation of any programs p_1 and p_2 is a valid program and denote it as $p_1 p_2$. Finally, we limit our considerations to domains with Markov property: the result of an instruction (the memory state it produces) depends only on the current memory state. $|p|$ is the length of program p (number of instructions).

The standard two-point crossover for this program representation has the natural interpretation of swapping parents' subprograms located between two randomly drawn loci i and j. The *partially medial crossover* (PMX) we propose is also homologous, and also affects subprograms in the parent solutions, however, the pieces of code pasted into the offspring result from analysis of semantic properties of the corresponding parents' subprograms. The choice of the code to be pasted between loci i and j can follow a simple principle: paste the subprogram that makes the resulting offspring possibly medial with respect to the parents *at locus j*.

More formally, the subprogram p that replaces the instructions from i to j ($|p| = j - i + 1$) in the kth offspring ($k = 1, 2$) to be created from parents p_1 and p_2 is the one that minimizes:

$$\arg\min_{p} d(p_k[1..i-1]p,\, p_1[1..j],\, p_2[1..j]) \tag{4}$$

where d is one of the divergence criteria (Eq. 3). Thus, when minimizing the divergence, PMX takes into account only the semantic effects of the first j instructions. The kth offspring is a program of the form $p_k[1..i-1]pp_k[j+1..|p_k|]$, where the 'head' $p_k[1..i-1]$ and the 'tail' $p_k[j+1..|p_k|]$ are copied from the kth parent.

PMX considers *all* potential subprograms p that can be pasted between loci i and j. As the number of such instruction sequences is exponential in function of $|p|$, we limit the span of loci i and j: only i is drawn at random, and j is set to $i + l - 1$, where l is a parameter that limits the length of considered subprograms. The number of instructions we need to execute in order to calculate (4) is still exponential in function of l, but can be significantly reduced, so that the computational overhead for small l is reasonable (see analysis of computational complexity in Section 5).

The ties that may occur when minimizing (4) are resolved at random. In this way, we avoid unnecessary bias, and the outcome of crossover becomes partially indeterministic.

4 The Experiment

The objective of the experiment is to compare the partially medial crossover in its two variants, minimizing the divergence from geometricity d_G (PMX$_G$) and minimizing the divergence from equidistance d_E (PMX$_E$). As control approach we use macromutation (MM), which overwrites the affected instructions with randomly generated instructions, and two-point crossover (2PX), that swaps

the affected subprograms between parents. To make comparison fair, they both affect a randomly selected continuous subsequence of instructions of length l, so the fraction of code they are allowed to modify is the same as for PMX.

We employ also one-point crossover (1PX) that draws a locus at random, splits each parent into head and tail, and swaps the tails, and a 'reset' operator RND that produces a random offspring (the *entire* offspring's code is randomized). All operators produce two offspring when applied to a pair of parents.

We conduct two experiments, one to quantitatively characterize the semantic impact of considered operators (Section 4.2), and one to assess the performance of evolutionary search (Section 4.3).

4.1 The Puzzle World

We adopt the task of solving the sliding puzzle as an experimental framework. Consider the 3×3 sliding puzzle with 8 movable pieces. The puzzle can be in one of $9! = 362,880$ states, which can change as a result of four possible moves L, R, U, D, where we assume the moves to shift the empty space (and thus also a piece). Any finite sequence composed of moves can be considered as a program, with moves playing roles of instructions, and the state of the puzzle corresponding to a memory state of a virtual machine that executes the program.

We define a *puzzle task* as follows: given a starting state s_0 and a target state t, find a program of length m that transforms the former one into the latter. Formally, let $s(p)$ denote the final memory state produced by program p that started execution from memory state s_0. The puzzle task is then to find p, $|p| = m$ such that $s(p) = t$. Although insisting on finding a program of length *exactly* m may sound too specific, let us note that almost every shorter program that solves a task can be extended to length m by inserting ineffective instructions that shift the space back and forth. Thus, a task can be solved by a program which effective length is less than m.

To solve the sliding puzzle task with an evolutionary approach, we evolve individuals that encode programs as fixed-length sequences (vectors) of instructions. Evolution will be driven by a (minimized) fitness function f defined as the total city-block distance between the locations of the 8 pieces and the empty space (_) in $s(p)$ and locations of corresponding elements in t, which is always an even number. Such definition of f is not accidental, it is used as a heuristic path cost estimate to solve this type of problems with exact algorithms like A*.

Formally, $f(p) = ||s(p), t||$, and f is thus a metric-based fitness function in the sense introduced in Section 2. For instance, if $t = (1,2,3,4,5,6,7,8,_)$, then

$$f\left(\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&6&\\\hline 7&8&5\\\hline\end{array}\right) = \left\|\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&6&\\\hline 7&8&5\\\hline\end{array}, \begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5&6\\\hline 7&8&\\\hline\end{array}\right\| = 0+0+0+0+1+1+0+0+2 = 4$$

Despite apparent simplicity, sliding puzzle captures the important features of programming task. It is *contextual*, i.e., the effect of an instruction depends on the current memory state. In particular, an instruction or instruction sequence can be ineffective when applied to a specific state (like the move R applied to

Table 1. The statistics describing average spatial relationships between the semantics of parents and offspring for different types of crossover operators

	Statistics	RND	MM	2PX	1PX	PMX$_E$	PMX$_G$
1	$\overline{\|\|p_1, p_2\|\| - \|\|o_1, o_2\|\|}$	0.00	0.00	0.00	0.00	0.83	0.78
2	$\overline{d_G(o, p_1, p_2)}$	11.53	3.83	3.76	4.61	4.91	2.83
3	$\overline{d_E(o, p_1, p_2)}$	3.54	7.94	7.88	5.07	5.87	7.98
4	$\overline{\|\|p, o\|\|}$	11.52	3.83	3.83	8.71	5.51	3.31
5	$\Pr(d_G(o, p_1, p_2) = 0 \| o \neq p_1, p_2)$	0.02	0.05	0.06	0.11	0.08	0.09

the state evaluated above). It is *compositional*: new programs can be created by composing (concatenating) other programs. Finally, the memory is composed of *multiple* elements and a single instruction changes only some of them.

Apart from these features, sliding puzzle exhibits also some features characteristic for *genetic* programming tasks. Firstly, programs are evaluated by *running* them (testing) on input data. Secondly, the performance of a program is a function of a *distance* between its output and the desired output.

4.2 Experiment 1: Properties of Search Operators

In this experiment, we analyze properties of the considered crossover operators by applying them to random programs. For each operator, we repeat $1,000,000$ times the following steps: (i) generate a random starting state s_0, (ii) generate two random parents p_1, p_2, (iii) apply the operator to p_1 and p_2, producing offspring o_1 and o_2, (iv) run the parent and offspring programs, starting with memory state s_0, and (v) measure the spatial relationships between the outputs (semantics) of parent and offspring programs. Note that this analysis abstracts from the target t, and that all these measurements concern *entire* programs (the final program outcomes).

For brevity we report the results only for program length $m = 40$ and $l = 3$, (the number of instructions affected by the PMX, MM, and 2PX), but other lengths led to consistent conclusions. Table 1 shows the averaged statistics obtained in step (v) of the above procedure, including 1) the reduction of distance between offspring compared to the distance between parents $\overline{||p_1, p_2|| - ||o_1, o_2||}$, 2) mean divergence from geometricity d_G, 3) mean divergence from equidistance d_E, 4) mean distance between the parent and the offspring $\overline{||p, o||}$, and 5) probability of generating a perfectly geometric offspring, excluding the trivial cases, i.e., such that the offspring is a copy of one of the parents.

We start with noting that the non-semantic operators (RND, MM, 2PX, 1PX) are incapable to produce offspring that is more semantically similar than the parents (row 1). The offspring of PMX operators, on the other hand, is typically substantially more similar to each other.

Some non-semantic operators, despite their simplicity, turn out to be quite good at producing medial solutions (rows 2 and 3). MM, 1PX, and 2PX diverge

from geometricity (d_G) more than PMX$_G$, which is the best in that respect, but less than PMX$_E$. However, considering the rate of non-trivial offspring that are perfectly geometric (row 5), only 1PX turns out to be better than PMX$_G$.

The offspring of PMX$_E$ is not the most equidistant from parents, yielding to 1PX and RND. The fact that RND attains the lowest d_E may be surprising at first, but can be explained by the structure of the space. For the considered sample of programs, the maximum distance between any pair of parents is 24, and the median of distance is 12. As the metric assumes only even values, a completely random offspring produced by RND is then quite likely to be equidistant. However, such offspring is typically very different from the parents, having mean parent-offspring distance almost twice as high as for PMX$_E$ (row 4), so it is unlikely to inherit much of their behavior.

The main conclusion we draw from this experiment is that, despite the fact that our medial crossover operators are partial, i.e., affect only short subprograms of parents' code, their effects propagate to the end of program and affect their semantic in expected way (i.e., consistent with the used criterion). Because they also clearly produce offspring that is more similar than the parents (row 1) and semantically not too distant from them (row 4), they are quite likely to prove useful in evolutionary search, which we verify in subsequent section.

One might argue that, from time to time, PMX happens to affect the very last 3 of 40 instructions, and the observed values of indicators are mainly due to such cases. Such events are however rare: there are 37 possible crossover points, so less than 3% of cases fall into this category. Also, we conducted an analogous analysis, not reported here for brevity, where the first crossover point was constrained to the first half of the genotype. The resulting values of indicators, though less extreme, confirmed the above conclusions.

4.3 Experiment 2: Performance in Evolutionary Search

To evaluate usefulness of PMX operators as search tools, we carried out a evolutionary experiment for different program lengths ($m = 20, 40, 60, 80$) and various length of the affected code fragment ($l = 3, 4, 5$).

Each evolutionary run is to find a solution to a specific, randomly generated puzzle task (s_0,t), where s_0 and t are random permutations of the canonical state (1,2,3,4,5,6,7,8,_). For each combination of m and l, 50 independent runs were carried out, each solving a different puzzle task. Note that, as only half of the 9! permutations of pieces are reachable from any given puzzle state [9], not all such tasks are solvable, i.e., there does not exist a sequence of moves of any length that leads from s_0 to t. Moreover, as the worst configuration of 8-puzzle requires 31 moves to solve (see the OEIS integer sequence A087725, [10]), some of the tasks are not solvable for the smallest considered program length $l = 20$. These difficulties however affect equally all the considered methods, so the comparison remains fair.

We use generational evolution driven by the fitness function defined in Section 4.1, i.e., as overall city-block distance between the locations of pieces in the state reached by the program from the locations of pieces in the goal state t. Each run

Table 2. Success ratio [%] for the 3×3 puzzle, for various length of code fragment affected by crossover (l) and total program length (m)

m	20			40			60			80		
l	3	4	5	3	4	5	3	4	5	3	4	5
RND	0	0	0	2	2	2	10	10	10	10	10	10
MM	2	0	0	22	30	28	40	42	48	46	50	50
2PX	0	0	0	2	4	2	6	6	6	22	16	20
1PX	0	0	0	2	8	6	0	10	10	18	18	16
PMX_E	4	2	6	30	48	46	46	62	60	58	60	62
PMX_G	0	0	0	6	6	6	18	10	20	40	28	34

evolves a population of 1,000 individuals for 1,000 generations, unless a solution is found earlier. The solutions are selected using tournament of size 7, after which they either undergo crossover using one of the aforementioned operators (with probability 0.9), or one-point mutation (with probability 0.1, and probability of affecting a single instruction 0.03).

Note that for RND evolution is effectively a random memoryless search.

Table 2 presents the success rate for different settings. Before comparing the operators, we should notice that all of them perform better when operating on longer programs. As the average task difficulty remains the same, this suggests that finding a program that reaches the target state from the starting state in, say, 40 instructions can be more difficult than finding a program that does the same using 80 instructions.

MM performs remarkably well, especially when confronted with other non-semantic operators. Apparently, introducing completely random modifications in the code is on average more profitable than purely syntactic swapping of code fragments implemented by 1PX and 2PX. However, the semantic-aware manipulation provided by PMX clearly pays off. In particular, for all considered parameter settings, PMX_E finds the optimum more frequently than any other operator. The efficiency of PMX_G as a search tool is much worse, though almost always not worse than 2PX and 1PX. This suggests that, at least for the considered domain of sliding puzzle, it is more important to generate solutions that inherit roughly the same 'fraction' of behavior from both parents, even if that share is low. Minimizing the divergence from geometricity is less effective, which may be due to the slower pace at which the PMX_G traverses the search space (see row 4 of Table 1).

The high performance of MM suggests that the task we consider here is relatively easy. This observation inclined us to consider the harder 4×4, 15-piece puzzle. As the 4×4 puzzle has 20,922,789,888,000 possible states, this time we consider longer programs of length 100, 200, and 300. All other settings, including the method of task generation, remain the same.

The results, presented in Table 3, support earlier conclusions. PMX_E is again superior, and its relative outperformance over MM is even larger than for the 3×3 puzzle: probability of finding the optimum is now often several times greater than for MM. PMX_G is much worse again, but still comparable to MM. Other operators fail completely.

Table 3. Success ratio [%] for the 4 × 4 puzzle, for various length of code fragment affected by crossover (l) and total program length (m)

m	100			200			300		
l	3	4	5	3	4	5	3	4	5
RND	0	0	0	0	0	0	0	0	0
MM	0	0	0	4	2	4	10	8	4
2PX	0	0	0	0	0	0	0	0	0
1PX	0	0	0	0	0	0	0	0	0
PMX_E	0	2	2	4	12	20	26	18	22
PMX_G	0	0	2	0	2	2	4	4	6

5 Discussion

Good performance of PMX suggests that generating programs that share some elements of behavior (semantic) with the earlier visited solutions is an important and desired feature of search operator. In particular, this turns to be more important here than inheriting the genetic material.

However, PMX is incapable to explicitly modify the semantic of programs and operates on short subprograms only. Although, as we have shown in Table 1, such modifications tend to affect the output of entire programs in the expected way, it is the affected loci (instructions from i to j) where the semantic effect of PMX is the strongest. Why then generating programs that exhibit medial semantics at *intermediate* stages of execution should be profitable?

Our working explanation is the modularity of the puzzle task. To solve a task, i.e., to reach the memory state t, one has first to solve a certain subtask, thus reach a certain intermediate memory state s' (subgoal), which is typically unknown prior to solving the task. The partially medial crossovers, by making more similar the memory states visited by the parent programs, may promote convergence to such subgoals. However, how effective this process is can depend on many factors, including the number of solutions that exist for a given puzzle task, the number of such subgoals for a given task, and the structure of the fitness landscape (e.g., how does attaining such a subgoal pay off in terms of fitness). Thus, this hypothesis remains to be verified, possibly with help of the concept of interdependency (see, e.g., [11,12]).

An important difference between PMX and the non-semantic operators is that the former involve partial execution of the considered program fragments. This involves an extra computational overhead, which is not reflected in Tables 2 and 3. PMX needs to run (i) the heads of the parent programs and then (ii) the subprograms to be pasted (Formula 4). The former part requires execution of $i - 1$ instructions per parent, but this has to be done only once: we store the result of head execution (the final memory state), and use it then as the starting memory state for considered subprograms. For a programming language comprising n instructions, PMX considers in the latter part n^l subprograms of length l, which apparently requires executing lm^l instructions. Note however that these subprograms partially overlap and can be represented by a n-ary tree

of depth l, which comprises only $n^l - 1$ nodes (instructions). For instance, for $l = 3$ and $n = 4$ this means reduction from 64 to 15 instructions.

The overall number of instructions executed when crossing over at locus i is then $2(i+n^l-2)$, where factor 2 is due to producing two offspring. The worst-case cost, for crossing over at the very end of parents, amounts to $2(m - l + n^l - 1)$. For short subprograms and small instructions sets this is a moderate computational overhead. Should this overhead become unacceptable, some form of sampling from the space of subprograms may be considered. Given that evolutionary search is stochastic anyway, exact minimization of d_G or d_E in (4) may be unnecessary, and some performance improvements should be possible to attain with approximate approach at lower computational cost.

6 Conclusion

The overall conclusion of this study is that partially medial crossovers, search operators that make the *subprograms* of parent programs *semantically intermediate* with respect to parents, can positively contribute to effectiveness of genetic programming algorithms. The extent of this contribution is undoubtedly problem-dependent, and will vary with instruction set and metric definition. Nevertheless, we hyptothesize that a significant fraction of real-world genetic programming problems can potentially benefit from this approach. Our rationale for this claim is that real-world programming tasks tend to be modular.

The puzzle world seems to constitute a convenient demonstrator for the concept of PMX for several reasons. Firstly, because the programs are intended to solve a specific instance of puzzle task, for a *single* input (starting memory state s_0), the semantic of a (sub)program can be identified with the memory state it produces. For problems that require testing a (sub)program on a set of fitness cases (e.g., symbolic regression), the notion of semantic and the associated metric would have to be adapted (cf. [2,4,5,13]). Secondly, puzzle manipulating has the natural interpretation of *sequential* programs, which implies convenient correspondence of loci in the parent programs. For less regular program representations like trees, PMX would need some extra means to first decide which subprograms (subtrees) to operate on. Such extensions and the above hypothesis concerning modular problems remain to be verified in future research.

Acknowledgment. This work has been supported by NSC grant no. DEC-2011/01/B/ST6/07318.

References

1. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
2. McPhee, N.F., Ohs, B., Hutchison, T.: Semantic Building Blocks in Genetic Programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 134–145. Springer, Heidelberg (2008)

3. Moraglio, A., Krawiec, K., Johnson, C.: Geometric semantic genetic programming. In: Igel, C., Lehre, P.K., Witt, C. (eds.) The 5th Workshop on Theory of Randomized Search Heuristics, ThRaSH 2011, Copenhagen, Denmark (2011)
4. Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Semantic Aware Crossover for Genetic Programming: The Case for Real-Valued Function Regression. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 292–302. Springer, Heidelberg (2009)
5. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Raidl, G., Rothlauf, F., Squillero, G., Drechsler, R., Stuetzle, T., Birattari, M., Congdon, C.B., Middendorf, M., Blum, C., Cotta, C., Bosman, P., Grahl, J., Knowles, J., Corne, D., Beyer, H.G., Stanley, K., Miller, J.F., van Hemert, J., Lenaerts, T., Ebner, M., Bacardit, J., O'Neill, M., Di Penta, M., Doerr, B., Jansen, T., Poli, R., Alba, E. (eds.) GECCO 2009: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, Montreal, pp. 987–994. ACM (2009)
6. Krawiec, K., Wieloch, B.: Analysis of semantic modularity for genetic programming. Foundations of Computing and Decision Sciences 34(4), 265–285 (2009)
7. Moraglio, A., Poli, R.: Topological Interpretation of Crossover. In: Deb, K., Poli, R., Banzhaf, W., Beyer, H.G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J., Harman, M., Holland, O., Lanzi, P.L., Spector, L., Tettamanzi, A., Thierens, D., Tyrrell, A. (eds.) GECCO 2004, Part I. LNCS, vol. 3102, pp. 1377–1388. Springer, Heidelberg (2004)
8. Johnson, C.G.: Genetic Programming Crossover: Does It Cross over? In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 97–108. Springer, Heidelberg (2009)
9. Archer, A.F.: A modern treatment of the 15 puzzle. American Mathematical Monthly 106, 793–799 (1999)
10. The On-line Encyclopedia of Integer Sequences, http://oeis.org
11. Altenberg, L.: Modularity in evolution: Some low-level questions. In: Rasskin-Gutman, D., Callebaut, W. (eds.) Modularity: Understanding the Development and Evolution of Complex Natural Systems, pp. 99–128. MIT Press, Cambridge (2005)
12. Watson, R.A.: Compositional Evolution: The impact of Sex, Symbiosis and Modularity on the Gradualist Framework of Evolution, NA. Vienna series in theoretical biology. MIT Press (February 2006)
13. Krawiec, K.: Semantically embedded genetic programming: automated design of abstract program representations. In: Krasnogor, N., et al. (eds.) GECCO 2011: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin, Ireland, pp. 1379–1386. ACM (2011)

Improving Face Detection

Penousal Machado[1], João Correia[1], and Juan Romero[2]

[1] CISUC, Department of Informatics Engineering, University of Coimbra,
3030 Coimbra, Portugal
{machado,jncor}@dei.uc.pt

[2] Faculty of Computer Science, University of A Coruña, Coruña, Spain
jj@udc.pt

Abstract. A novel Genetic Programming approach for the improvement of the performance of classifier systems through the synthesis of new training instances is presented. The approach relies on the ability of the Genetic Programming engine to identify and exploit shortcomings of classifier systems, and generate instances that are misclassified by them. The addition of these instances to the training set has the potential to improve classifier's performance. The experimental results attained with face detection classifiers are presented and discussed. Overall they indicate the success of the approach.

Keywords: Face detection, Haar cascade.

1 Introduction

Object detection systems, in particular face detection, have become a hot topic of research. Applications that employ this kind of systems are becoming widespread. For instance, they can be found in search engines, social networks, incorporated in cameras, or in applications for smart phones. Like in other example-based learning techniques, the datasets employed are vital, not only for attaining competitive performances, but also for correctly assessing the strengths and shortcomings of the classifiers. As such, developing adequate datasets for training, testing and validation becomes a crucial and complex process.

The use of Evolutionary Computation (EC) techniques in the fields of Computer Vision (CV) and Machine Learning (ML) is widespread. Among other applications, EC has been used for digital filters tuning, parameter optimization and image generation. In the field of ML, EC applications include evolving classifier parameters, thresholds, feature selection for classification, the classifier itself, etc. Works such as [17,7,1,14] combine EC, CV and ML aspects.

This paper explores the use of Genetic Programming (GP) to assess and improve classifier's performance through the synthesis of new training examples. More specifically, the current work focus on: (i) assessing classifier's performance, (ii) using evolutionary algorithms to generate new examples, (iii) using the generated examples to boost the performance of the classifier.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 73–84, 2012.
© Springer-Verlag Berlin Heidelberg 2012

We propose a novel generic evolutionary framework for classifier improvement through the synthesis of new training examples. This framework is then instantiated by combining a GP image generation system with a state of the art face detector [19].

The experimental results show that GP was successful in finding shortcomings of the face detector, generating hundreds of images that were incorrectly classified. They also show that the addition of these images to the training set reduces its shortcomings, promoting detection accuracy, and leading to better classifier's performance.

The paper is organized as follows: Section 2 makes a brief overview of related work; Next, in section 3, we present the proposed framework for classifier's performance; Section 4 describes the experimental setup; The experimental results are presented and analyzed in section 5; Finally, in section 6, overall conclusions are drawn and future research indicated.

2 State of the Art

As previously mentioned, EC has been used in the development and improvement of classifier systems. However, we were unable to find works that match closely the approach proposed in this paper. In this section we make a short overview of approaches that share common features and goals.

Ventura et al.[18] used EC to generate training samples for a Neural Network (NN). The goal was to optimize a computer network routing system. The fitness function was designed to achieve a pre-determined state. The individuals were composed by vectors of control values that represented the state of the network at some point in time. A NN was then trained with the best individuals. This NN was submitted to a series of tests related to the aimed network state. This work represents and attempt to evolve training samples using a GA which is one of the common goals of our work.

The work of Mayer et al. [12] focused in the optimization of NN's training set. It consisted in a Genetic Algorithm Active Selection method. An Active Sampling method generated new data patterns based on the selected training data, in order to enhance the dataset with new information. The GA evolved subsets of training and sampled data, which were used to train NNs. These NNs were assessed by a testing set. The performance in the test phase determined which subset was the best training set to be used. Related to our work, this approach generates new training samples from the existing samples.

Chen et al. [2] proposed a self-adaptive GA to improve face detection systems. It consisted on resampling the face training dataset. The individuals of the GA were encoded as strings containing the pixel intensity values. The individuals were submitted to mutation and recombination operators. Recombination consisted in segmenting two individuals and combining some of the segmented parts. Mutation consisted in the probability of changing illumination, position and angle of the selected segmented parts. The whole process starts with the face training set being employed as an initial population to perform GA operations.

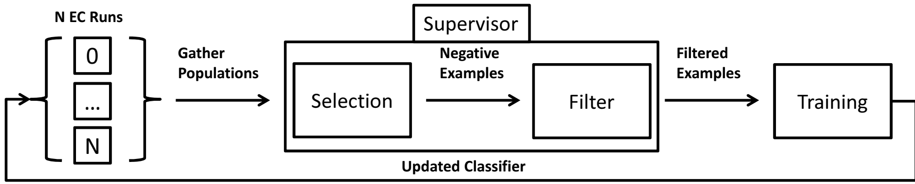


Fig. 1. System overview

The intermediate solutions of each generation were evaluated by a Sparse Network of Winnows (SNoW) classifier [20], trained with the last non face samples. The classifier output was used to assign fitness of the individuals. The fittest individuals continued to the next generation and the weaker were discarded. The last population was added to the face training set which was used to train a new SNoW classifier. In this case it relates to this paper due to the usage of the classifier output to fitness assignment.

More recently, D. Dubey [3] explored the face detection problem by using NNs, resampling methods and a GA. The images pixel intensity values were considered as individuals. The initial face training set examples were *resampled* by using rotation and scale operations, generating and adding new samples to the original ones. The non-face training set started with white noise images, created by assigning random intensities to each pixel. A NN was trained to discriminate between face and non-face image with the initial sets. The GA was used to evolve the non-face initial set. The output of the NN was used to assign fitness. The non-face image set was updated by randomly selecting non-face individuals that were misclassified during fitness. In each generation a new NN was trained. After ending the GA process, a final NN was trained with the last generation of non-faces and existing face images. The relation to our work lays on the usage of misclassified examples and their inclusion in the training set.

3 The Framework

The proposed framework comprises three main modules: EC engine, Classifier and Supervisor. Figures 1 and 2 present an overview of the framework and the interaction between the EC engine and Classifier, respectively.

The application of this approach involves the following steps:

1. Selection of a positive and negative image set;
2. A Classifier System (CS) is trained based on the positive and negative instances;
3. N independent EC runs are started; The CS is used to classify the generated individuals; Their fitness depends on the results, including intermediate ones, of the classification task;
4. The EC runs stop when a termination criterion is met (e.g., a pre-established number of generations, attaining a fitness value);

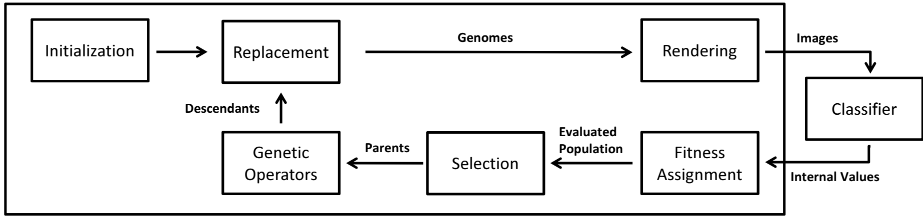


Fig. 2. Evolutionary model and its interaction with the classifier

5. The set of negative images is updated by adding the evolved images for which the CS and the Supervisor do not agree (e.g. classified as positive by the CS and as negative by the Supervisor)
6. The process is repeated from step 2 until the boosting criterion is met;

By explaining how this framework is instantiated we will also explain the underlying rationale. In the context of this paper the CS system consists in a Haar Cascade classifier (see Viola et al. [19]) built to detect frontal faces. The code and executables are included in the OpenCV API[1]. This CS approach was chosen due to its state of the art relevance and for its fast classification. This algorithm uses a set of small features in combination with a variant of the Ada-boost [4], and is able to attain efficient classifiers. The classifiers assume the form of a cascade of small and simple classifiers that use Haar features [13].

The EC engine used in this experiments is inspired by the works of Sims [15]. It is a general purpose, expression-based, GP image generation engine that allows the evolution of populations of images. The genotypes are trees composed from a lexicon of functions and terminals. The functions include mathematical and logical operations; the terminal set is composed two variables, x and y, and random constant values. The phenotypes are images, rendered by evaluating the expression-trees for different values of x and y, which serve both as terminal values and image coordinates. In other words, to determine the value of the pixel in the (0,0) coordinates one assigns zero to x and y and evaluates the expression-tree. A thorough description of the GP engine can be found in [10].

In the context of this paper, positives are images that contain faces while negatives are images where no face is present. The goal of the EC engine is to evolve images that the CS classifies as faces. To create a fitness function able to guide evolution it is necessary to convert the binary output of the face detector, to one that can provide suitable fitness landscape. This is attained by accessing internal results of the classification task that give an indication of the degree of certainty in the classification. As such, images that are immediately rejected by the classifier will have lower fitness values than those that were close to be classified as possessing a frontal face.

[1] OpenCV — http://opencv.willowgarage.com/wiki/

Considering the structure of the selected classifier and through trial and error we developed the following fitness formula:

$$fitness(x) = \sum_{i}^{countstages_x} beststagedifference_x(i) * i + countstages_x * 10 \quad (1)$$

Variables $countstages_x$ and $beststagedifference_x(i)$ are extracted from the face detection algorithm. Variable $countstages_x$, holds the number of stages that image, x, has successfully passed in the cascade of classifiers. The rationale is the following, an image that passes several stages is likely to be closer to being recognized as having a face than one that passes fewer stages. In other words, passing several stages is a pre-condition to being identified as a face image. Variable $beststagedifference_x(i)$ holds the maximum difference between the threshold necessary to overcome stage i^{th} and the value attained by the image at the ith stage. Images that are clearly above the thresholds necessary to pass each stage are preferred over ones that are only slightly above them. Obviously, this fitness function is only one of the several possible ones. Although room for improvement is likely to exist, such improvements are not necessary for the goals of the current paper.

The proposed framework relies on the ability of EC systems to find and exploit the shortcomings of the classifiers to "artificially" increase fitness. The propensity of EC to find "shortcuts" that exploit weaknesses of the fitness assignment scheme is well-known (see, e.g., [16,17,11]). Thus, the goal is to evolve false-positives: images that are classified as faces, but that should not have been classified as faces. By adding this false-positives to the negative training set and re-training the CS we wish to correct exploitable flaws of the classifier.

The Supervisor for this experiment is an automatic module that is responsible for gathering all distinct images created during the EC runs. Evolved images that are classified as faces are added to the training set for the next boosting iteration.

4 Experimental Setup

To assess the validity of the proposed approach we performed 30 independent runs of the framework described in the previous section. The framework proposes the use o N independent evolutionary runs, however, we are primarily interested in assessing the contributions that each EC run may bring. Thus, for the scope of this paper we set $N = 1$ and perform 30 independent runs of the the proposed framework. In this section we describe the experimental settings employed in these runs.

4.1 Classifier Training

For training purposes we used the "opencv_haartraining" tool of OpenCV. The relevant classifier parameters are presented in table 1 and were chosen based on the works of Viola [19] and Lienhart [9,8]. They reflect a compromise between

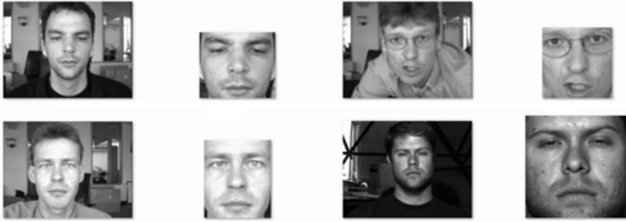


Fig. 3. Examples of cropped positive images

attaining good classifier's performance and manageable training time. In addition to the training parameters, there are other classifier settings that need to be established. We chose to use the default parameters of OpenCV (see table 2).

The quality of the positive and negative datasets used in training significantly influences the performance of a classifier. It is important to have good positive examples of the object that we are training in order to attain good success rates. For this experiment images from two well-known datasets were used: "The Yale Face Database B" ([5]) and "BioID Face Database"([6]). "The Yale Face Database B" is a dataset with a total 5850 grayscale images with the subjects in diverse positions and light variations. The Bio-ID Face Database dataset has 1521 frontal grayscale images. Each image shows the frontal view of a face of one out of 23 different test persons with various expressions.

We wish to test if the proposed framework contributes to improvements of classifier's performance. Adding different poses has no interest in this context and would make development and analysis harder. As such, we decided to focus exclusively on frontal faces. Although it is easier to develop a good initial classifier, it is likely to make improvements harder, since there is less room for improvement.

Considering this constraint, the total number of available positive examples is 2172. In order to build the ground truth file, the images have to be manually selected and cropped. These cropped images, see figure 3, are the objects that the Haar classifier attempts to discriminate from negative samples. After manually filtering out images that were too dark, or where only part of the face was illuminated, a total of 1905 positive examples, and corresponding cropped versions, was attained.

Table 1. Haar Training parameters

Parameter	Setting
features	ALL
Input width	20
Input height	20
Number of stages	14
Number of splits	1
Min Hit rate	0.999
Max False Alarm	0.5
Adaboost Algorithm	GentleAdaboost

Table 2. Classifier parameters

Parameter	Setting
Window width	20
Window height	20
Scale factor	1.2
Min face width	$0.75 \times inputwidth$
Min face height	$0.75 \times inputheight$

Fig. 4. Examples of negative images

Table 3. Parameters of the GP engine

Parameter	Setting
Population Size	100
Number of generations	50
Crossover probability	0.8
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion, node mutation
Initialization method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Function set	+, -, *, /, min, max, abs, sin, cos, if, pow, mdist, warp, sqrt, sign, neg
Terminal set	X, Y, scalar and vector random constants

The negative dataset influences both the training time and test performance. Generally speaking hard and large negative datasets imply longer training times, but also better performance. We employed the "Urtho - Negative face Dataset"[2], which consists of a total of 3019 images of landscapes, objects, drawings, etc. To keep the carnality of the negative and positive datasets balanced we randomly selected 1905 of the Urtho images. A sample is presented in figure 4.

4.2 Genetic Programming Engine

The settings of the GP engine are presented in table 3. The number of generations may appear low, however, preliminary experiments indicated that 50 generations were enough to evolve images classified as faces. Further tests showed that some of the evolutionary runs (23% in the conducted experiments) were unable to evolve such images, but also that increasing the number of generations was inefficient. Therefore, we tackle this problem as follows: if after 50 generations the evolutionary run is unable to find a minimum of 300 images, this run is discarded, and a new evolutionary run with a different random seed is initiated.

[2] Tutorial haartraining — http://tutorial-haartraining.googlecode.com/svn/trunk/data/negatives/

Table 4. Parameters used by the performance tool

Parameter	Setting
Minimum Window width	20
Minimum Window height	20
Scale factor	1.2
Maximum size difference factor	1.5
Maximum position difference factor	0.3

4.3 Assessing Classifier's Performance

In order to test the different classifiers, a performance evaluation tool was implemented. It allows loading an image test set, with a ground truth file associated, and a classifier configuration file. The performance is measured in terms of hits (H), misses (M), false alarms (FA), correct (C) and incorrect (I). In order to do this, it loads the parameters and classifier of the configuration file, and perform the face detection. Then it compares the result with the ground truth file. If the result matches or lays within the tolerance area defined by the performance tool parameters, it is a hit. If it lays outside the tolerance area it is counted as a false alarm. If no face is detected and a face exists, it is counted has a miss. A positive instance is considered correctly classified if, and only if (i) at least one face was detected and (ii) the regions were the faces were detected match the expected region. In other words, there must be at least one hit and no false alarms. An example follows, if the classifier identifies 2 faces on an image, one in the expected position and the other in an incorrect position, then the instance is considered incorrectly classified. A negative instance is classified as correct if the classifier detects no faces.

The parameters are defined in table 4 and are based on the default parameters of OpenCV's "opencv_performance" tool.

5 Experimental Results

As previously mentioned we performed 30 independent runs of the framework presented in section 3 using the experimental settings described in section 4. As previously mentioned, although the framework proposes the use of several parallel evolutionary runs, we in this test $N = 1$.

Figure 5 displays the evolution of the population average fitness and of the best population individual across generations. In essence this chart shows that in successful runs the GP engine finds images that are classified as faces in few generations. Please notice that runs where the GP was unable to find images classified as faces were discarded. These runs are useless for improving the classifier's performance since no images would be added to the dataset.

Figure 6 presents examples of images evolved in different evolutionary runs. All of these images have been considered faces by the classifiers. This highlights

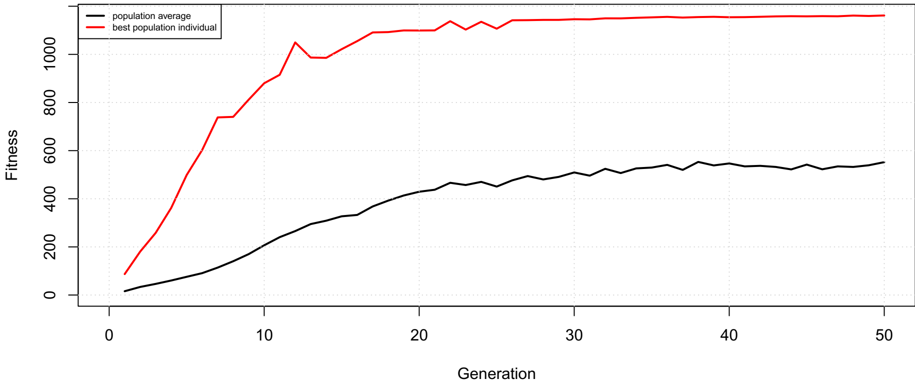


Fig. 5. Evolution of fitness across generations. Results are averages of 30 independent runs.

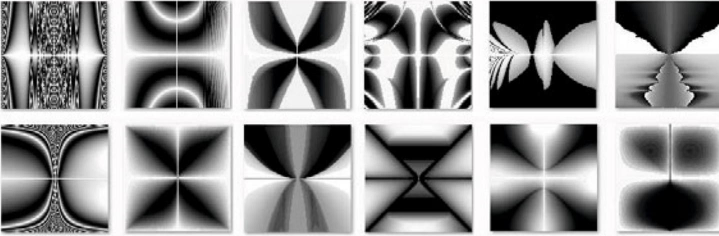


Fig. 6. Examples of evolved images that were classified as faces

the shortcomings of the classification system, based on a state of the art classification approach, and further indicates the ability of the GP engine to exploit these shortcomings finding images that are false positives.

Once each evolutionary run ends, the images classified as faces are added to the negative dataset and the classifier is re-trained. This process yields 30 new classifiers. Considering the goals of our research our primary interest is the comparison of the performance of these classifiers with the initial classifier model. For this purpose we consider two validation sets:

– Flickr – 2166 negative images;
– Feret – 902 positive images from Facial Recognition Technology Database[3];

The Flickr image dataset consists in images retrieved from a search in Flickr using the keyword "image" and excluding from the resulting set, images that contain a frontal human face. This process results in a *negative* dataset composed of landscapes, buildings, animals, computer screenshots, varied objects, etc.

The Feret validation set is a *positive* dataset composed by grayscale frontal faces, one face per image with a simple background. The images were manually

[3] The Feret Database – http://face.nist.gov/colorferet/colorferet.html

Fig. 7. On the top row, samples of images of the Flickr dataset; On the bottom row samples of the Feret dataset

Table 5. Results attained by the initial classifier and by the framework classifiers in three independent validation datasets. The Flickr dataset is a negative dataset the concept of hits and misses does not apply.

	Flickr		Feret			
Classifier	FA	%C	H	M	FA	%C
Initial	861	73.45	852	50	97	85.59
Average	643.00	78.91	844.00	58.00	77.73	86.12
Classifier 14	581	80.97	852	50	63	88.47
Classifier 30	701	77.75	856	46	64	88.91

selected and cropped. The purpose of using this validation dataset is to test the ability of the classifiers in detecting a clear frontal face.

Samples of the validation sets are presented in figure 7.

Table 5 presents a synthesis of the attained results, indicating the performance of the: initial classifier; average performance of the 30 classifiers created using the framework; performance of two of the best framework classifiers found.

Focusing on the comparison of the initial classifier with the average performance of the framework classifiers: the most striking difference in performance is the significant decrease in the number of false alarms which occurs for both validation datasets. On average, for the two datasets, there is a decrease of 25% in the number of false alarms. Adding false positives to the negative training dataset results in classifiers that are more "demanding" than the initial one when it comes to consider the presence of a face in an image. As a consequence, it becomes more robust and precise in the identification, which leads to a decrease in the number of false positives.

The disadvantage is that some face images may go unnoticed. In fact a decrease of the number of hits occurs in the Feret validation dataset (852 vs. 844, which represents a decrease of less than 1%) and, consequently, of the number of misses (50 vs. 58, a 13.8% increase). More importantly, the percentage of correctly identified images (C) increases for both validation datasets. As expected, the improvements of performance are more noticeable in the Flickr dataset, which is composed exclusively of negative images.

It is also important to compare the performance of the best framework classifiers with the performance of the initial models. Table 5 also presents the results

of two of the frameworks classifiers that showed increases in performance in both validation datasets. These results demonstrate that it is possible to increase the percentage of correctly identified faces and decrease false alarms without sacrificing the number of hits and misses. Unfortunately, unless it is possible to identify which classifiers will show this behavior in validation sets before gathering the validation results, the relevance of this results is limited.

Although in this paper we focus on examining the behavior of the proposed framework, it is important to notice that from a practical perspective we do not need 30 classifiers, we just need one. Further testing is necessary to determine if the performance of these classifiers is generalizable to other validation datasets.

6 Conclusion and Future Work

A novel evolutionary framework for the improvement of classifier's performance through the synthesis of training examples is presented and discussed. The experimental results attained in two validation datasets show the potential of the approach, demonstrating significant decreases in the number of false alarms and small losses in the number of hits. Additionally, several of the framework classifiers yield better performance in all parameters and for both validation datasets.

Although the results are promising there are several aspects that require further testing and development. Additional testing is necessary to assess if the results attained by the best framework classifiers are generalizable to other validation datasets. The framework anticipates the use of several parallel evolutionary runs and boosting iterations, but the presented results consider only one. Gathering the evolved false positives of all EC runs, adding them all to the negative dataset, and training a single classifier is likely to yield better overall performance. By increasing the number of iterations one forces the EC to focus on different shortcomings of the classifier, which may result in better overall performance.

A final word goes to the supervisor module. Judiciously selecting which images should be added to the negative dataset is likely to contribute to better performances and lower training times. Experiments concerning these aspects are already taking place.

Acknowledgments. This research is partially funded by: the Portuguese Foundation for Science and Technology, project PTDC/EIA–EIA/115667/2009; the Spanish Ministry for Science and Technology, project TIN2008–06562/TIN; Xunta de Galicia, project XUGA–PGIDIT10TIC105008PR.

References

1. Baro, X., Escalera, S., Vitria, J., Pujol, O., Radeva, P.: Traffic Sign Recognition Using Evolutionary Adaboost Detection and Forest-ECOC Classification. IEEE Transactions on Intelligent Transportation Systems 10(1), 113–126 (2009)

2. Chen, J., Chen, X., Gao, W.: Resampling for face detection by self-adaptive genetic algorithm. In: Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004, vol. 3, pp. 822–825 (August 2004)
3. Dubey, D.: Face detection using genetic algorithm and neural network. International Journal of Science and Advanced Technology 1(6), 104–109 (2011) ISSN 2221-8386
4. Freund, Y., Schapire, R.E.: A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting (1995)
5. Georghiades, A.S., Belhumeur, P.N., Kriegman, D.J.: From few to many: illumination cone models for face recognition under variable lighting and pose. IEEE Transactions on Pattern Analysis and Machine Intelligence 23(6), 643–660 (2001)
6. Jesorsky, O., Kirchberg, K.J., Frischholz, R.W.: Robust Face Detection Using the Hausdorff Distance. In: Bigun, J., Smeraldi, F. (eds.) AVBPA 2001. LNCS, vol. 2091, pp. 90–95. Springer, Heidelberg (2001)
7. Krawiec, K., Howard, D., Zhang, M.: Overview of Object Detection and Image Analysis by Means of Genetic Programming Techniques. In: Frontiers in the Convergence of Bioscience and Information Technologies, FBIT 2007, pp. 779–784 (2007)
8. Lienhart, R., Kuranov, A., Pisarevsky, V.: Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. In: Michaelis, B., Krell, G. (eds.) DAGM 2003. LNCS, vol. 2781, pp. 297–304. Springer, Heidelberg (2003)
9. Lienhart, R., Maydt, J.: An Extended Set of Haar-Like Features for Rapid Object Detection. In: Proceedings of the 2002 International Conference on Image Processing, vol. 1, pp. 900–903 (2002)
10. Machado, P., Cardoso, A.: All the truth about NEvAr. Applied Intelligence, Special Issue on Creative Systems 16(2), 101–119 (2002)
11. Machado, P., Romero, J., Manaris, B.: Experiments in Computational Aesthetics. In: The Art of Artificial Evolution, Springer, Heidelberg (2007)
12. Mayer, H.A., Schwaiger, R.: Towards the evolution of training data sets for artificial neural networks. In: IEEE International Conference on Evolutionary Computation, pp. 663–666 (April 1997)
13. Papageorgiou, C.P., Oren, M., Poggio, T.: A general framework for object detection. In: Sixth International Conference on Computer Vision, pp. 555–562 (January 1998)
14. Sha, S., Jianer, C., Ling, Q., Sanding, L.: Evolutionary mechanism and implemention for recognition of objects in dynamic vision. In: 4th International Conference on Computer Science Education, ICCSE 2009, pp. 178–182 (2009)
15. Sims, K.: Artificial Evolution for Computer Graphics. ACM Computer Graphics 25, 319–328 (1991)
16. Spector, L., Alpern, A.: Criticism, culture, and the automatic generation of artworks. In: Proceedings of Twelfth National Conference on Artificial Intelligence, pp. 3–8. AAAI Press/MIT Press, Seattle, Washington (1994)
17. Teller, A., Veloso, M.: Algorithm evolution for face recognition: what makes a picture difficult. In: IEEE International Conference on Evolutionary Computation 1995 (1995)
18. Ventura, D., Andersen, T., Martinez, T.R.: Using evolutionary computation to generate training set data for neural networks. In: Proceedings of the International Conference on Neural Networks and Genetic Algorithms, pp. 468–471 (1995)
19. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Hawaii, vol. 1, pp. I–511–I–518 (2001)
20. Yang, M.-H., Roth, D., Ahuja, N.: A snow-based face detector. In: Advances in Neural Information Processing Systems 12, pp. 855–861. MIT Press (2000)

Grammar Bias and Initialisation in Grammar Based Genetic Programming

Eoin Murphy, Erik Hemberg, Miguel Nicolau,
Michael O'Neill, and Anthony Brabazon

Natural Computing Research and Applications Group,
University College Dublin, Ireland
{eoin.murphy,erik.hemberg,miguel.nicolau,m.oneill,anthony.brabazon}@ucd.ie

Abstract. Preferential language biases which are introduced when using Tree-Adjoining Grammars in Grammatical Evolution affect the distribution of generated derivation structures, and as such, present difficulties when designing initialisation methods. Similar initial populations allow for a fairer comparison between different GP methods. This work proposes methods for dealing with these biases and examines their effect on performance over four well known benchmark problems. In addition, a comparison is performed with a previous study that did not employ similar phenotype distributions in their initial populations. It is found that the use of this form of initialisation has a positive effect on performance.

Keywords: Grammatical evolution, Grammar bias, Initialization.

1 Introduction

It has been shown that the form of a grammar and indeed the language biases inherent to that grammar can have a large impact on the performance of grammar-based Genetic Programming (GP) systems [14, 2], such as Grammatical Evolution (GE) [13]. Modification of the grammar, an integral part of the GE algorithm, can cause the algorithm to behave very differently [2]. This is due to the ease with which the language biases in the system can change by just modifying the grammar, effecting how genotypes are mapped into phenotypes.

In a previous study by Murphy et al. [12], GE, which traditionally uses a Context-Free Grammar (CFG), was extended to make use of Tree-Adjoining Grammars (TAG) [6], in the form of Tree-Adjunct Grammatical Evolution (TAGE). A preliminary comparison of the two methods was performed, testing each method on a number of different problems, with TAGE showing improvements in performance, such as finding more correct solutions, as well as finding better solutions in fewer generations, than standard GE on those problems [12]. The transformation from CFG to TAGE modifies the existing language bias as well as introducing new language biases into the GE algorithm. These biases affect the algorithm's ability to generate certain derivation structures and phenotypes, and hence altering the search space [11]. This can be detrimental to the

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 85–96, 2012.
© Springer-Verlag Berlin Heidelberg 2012

generation of common initial populations, an important factor in performing a fair comparison between any two GP systems.

The aim of this study, therefore, is to examine and address the different types of language bias which are introduced by using the TAG representation (see Section 4.1 in order to better align the initial populations of both setups. Biases such as the structural biases imposed by the adjunction operation, as well as those introduced as a result of the grammar transformation, i.e., the loss of explicit biases imposed by the CFG. The study proposes a novel algorithm for the elimination of the grammar transformation biases (see Section 4.3) and goes on to perform a comparison between GE and TAGE. The study observes the effect of these biases on performance and comparing with the results observed by Murphy et al. [12].

In the following section a brief description of GE is given. Section 3 gives an overview of TAGE, the transformation from CFG to TAG and concludes with a sample TAGE derivation. Following this, Section 4 examines the problems involved with creating similar initialisation methods for the two different grammar types. The experimental setup is described in Section 5, with Section 6 outlining the results obtained, as well as providing some discussion on these results. Section 7 concludes the study.

2 Grammatical Evolution

GE is a grammar-based approach to GP, combining aspects of Darwinian natural selection, genetics and molecular biology with the representational power of grammar formalisms [13]. The use of a grammar enables GE to define the legal expressions and structures of an arbitrary language, which in turn allows the possible generated structures and syntax of solutions to be easily modified, something that is not trivial for other forms of GP. In addition to this, the separation of the genotype from phenotype in GE allows genetic operations to be applied to both, extending the search capabilities of GP. GE is considered to be one of the most widely applied GP systems today [10].

Representation in GE consists of a grammar and a chromosome (see Fig. 1). The genotype-phenotype mapping in GE uses codon values from the chromosome to select production rules from the grammar. By performing the modulus operation on these values with the number of possible production choices, productions are selected from the grammar to expand each non-terminal (NT) symbol. Starting from the start symbol, this process, which continues in a left-right manner until there are no more NT leaf nodes to expand, or until the end of the chromosome has been reached, constructs a derivation tree. The phenotype can then be extracted from the leaf nodes of this tree. A sample derivation tree is shown in Fig. 1 along with the grammar and chromosome used to construct it.

3 Tree-Adjunct Grammatical Evolution

TAGE, like GE, uses a representation consisting of a grammar and a chromosome. However, the type of grammar used in this case is a TAG rather than a

CFG. A TAG is defined by a quintuple (T, N, S, I, A) where T is a finite set of terminal symbols; N is a finite set of NT symbols: $T \cap N = \emptyset$; S is the start symbol: $S \in N$; I is a finite set of finite trees called *initial trees* (or α trees); and A is a finite set of finite trees called *auxiliary trees* (or β trees).

The root node of an initial tree is labelled with S and the interior nodes are labelled with NT symbols. Initial tree's leaf nodes are labelled with terminal symbols. Similarly, the interior nodes of auxiliary trees are also labelled with NT symbols, with their leaf nodes being labelled with terminal symbols. However, one special leaf node called the foot node is labelled with the same NT symbol as the root. Foot nodes are marked with * [6].

Initial trees represent the minimal non-recursive structures produced by the grammar, i.e., they contain no repeated NT symbols. Inversely, auxiliary trees of type X represent the minimal recursive structures, which allow recursion upon the NT X [8]. The union of initial trees and auxiliary trees forms the set of *elementary trees*, E; where $I \cap A = \emptyset$ and $I \cup A = E$.

During derivation, the adjunction composition operation joins elementary trees together. Adjunction takes an initial or derived tree a, creating a new derived tree d, by combining a with an auxiliary tree, b. A sub-tree, c is selected from a. The type of the sub-tree (the symbol at its root) is used to select an auxiliary tree, b, of the same type. c is removed from a. b is then attached to a as a sub-tree in place of c and c is attached to b at the position of b's foot node. An example of TAG derivation is provided in Section 3.1.

3.1 TAGE Derivation Example

TAGE generates TAGs from the CFGs used by GE. Joshi and Schabes [6] state that for a *"finitely ambiguous CFG which does not generate the empty string, there is a lexicalised tree-adjunct grammar generating the same language and tree set as that CFG"*. An algorithm was provided by Joshi and Schabes [6] for generating such a TAG. The TAG produced from Fig. 1 is shown in Fig. 2.

Derivation in TAGE is different to GE. Unlike GE derivation trees whose nodes are labeled by symbols, the nodes of a TAGE derivation tree are labelled by elementary trees. The edges between those nodes are labelled with the address of a node in the tree labelling the parent node. It is at this address that the auxiliary tree labelling the child is to be adjuncted. A derived tree in TAGE is a tree of symbols, similar to GE's derivation tree, resulting from the application of the adjunction operations defined in the TAGE derivation tree.

Given the TAG G, where $T = \{$`X, Y, +, -`$\}$, $N = \{$`<e>,<o>,<v>`$\}$, $S =$`<e>` and I and A are shown in Fig. 2, derivation using the chromosome from Fig. 1 operates as follows. The first codon value, `12`, is read and is used to choose an initial tree based on the number of trees in I. Using the same mapping function as GE, `12 mod 2 = 0`, the zero-th tree, α_0, is chosen from I. This tree is set as the root node of `t`, the derivation tree (as seen in Fig. 3(a)).

Next, a location to perform adjunction must be chosen. The vector `N` is created of the adjunctable addresses available within all nodes (trees) contained within `t`. An adjunctable address in a tree is the breadth first traversal index of a node

labelled with a NT symbol, of which there is an auxiliary tree of that type and there is currently no auxiliary tree already adjoined at that index. In this case N = $\{\alpha_0[0]\}$ (the zeroth node of α_0), so a codon is read and an address is selected from N, 3 mod 1 = 0 indicating which address to choose, N[0]. Adjunction will be performed at $\alpha_0[0]$, or index 0 of tree α_0, <e>. An auxiliary tree is now chosen from A that is of the type T, i.e., the label of its root node is T, where T is the label of the node where adjunction is being performed. In this case T = <e>. There are 8 such trees in A, Reading the next codon, 7, 7 mod 8 = 7, therefore β_7 is chosen. This is added to t as a child of the tree being adjoining to, labelling the edge with the address 0, see Fig. 3(b). The adjunctable addresses in β_7 will be added to N on the next pass of the algorithm. This process is repeated until all remaining codons have been read. The resulting derivation and derived trees at each stage of this process can be seen in Fig. 3.

4 Difficulties with Comparing GP Systems

Performing a fair comparison between different GP systems is difficult to achieve. As suggested by Hoai et al. [4], it is easy to assume that the benefits observed when testing a new modification to an algorithm are a direct consequence of the modification in question [1], whereas in reality this can be a flawed assumption. Unless the modification is very localised, there can be far reaching indirect effects, and if these effects influence the starting conditions of the algorithm, performing a comparison can be difficult. This problem is even more evident when comparing completely different algorithms.

These difficulties can be seen when comparing standard GP algorithms with grammar-based versions, as was shown in [4] when comparing GP and TAG3P. The change in representation makes it difficult to create common initial conditions for both algorithms in order to achieve a good comparison.

Comparing similar algorithms with different representations raises an interesting question: having a common initial population (or at least initial populations drawn from similar distributions) is good practice and helps ensure a fair comparison, but should these similar populations be in the genotypic or phenotypic spaces? Search is performed in the genotypic space, whereas the fitness landscape lies in phenotypic space, and depending on the mapping between the two there could be a many to one relationship between genotypes and phenotypes. That is to say, creating initial populations of genotypes for two different representations, e.g., GE and TAGE, would likely result in very different populations of phenotypes. The same can be said for similar populations of phenotypes, the populations of genotypes could be very different (see Fig. 4).

In Murphy et al. [12], a set genotype length was used; as a counter point, this study uses an initialisation method which produces similar sets of derivation (TAG derived) trees, and hence similar sets of phenotypes, for both GE and TAGE. The following subsections outline and address some of the problems faced while attempting to achieve this.

```
Grammar:
<e>:= <e><o><e> | <v>
<o>:= + | -
<v>:= X | Y

Chromosome:
12, 3, 7, 15, 9, 36, 14
```

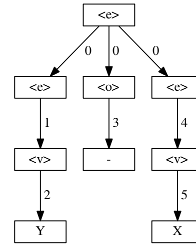
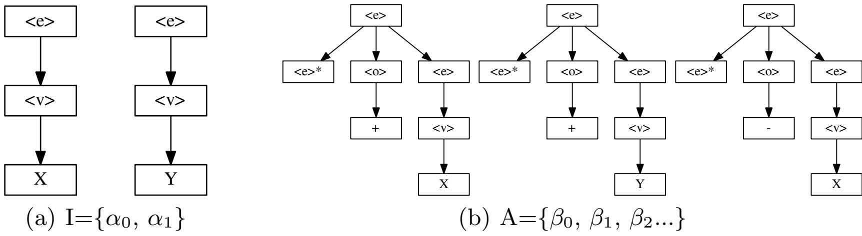
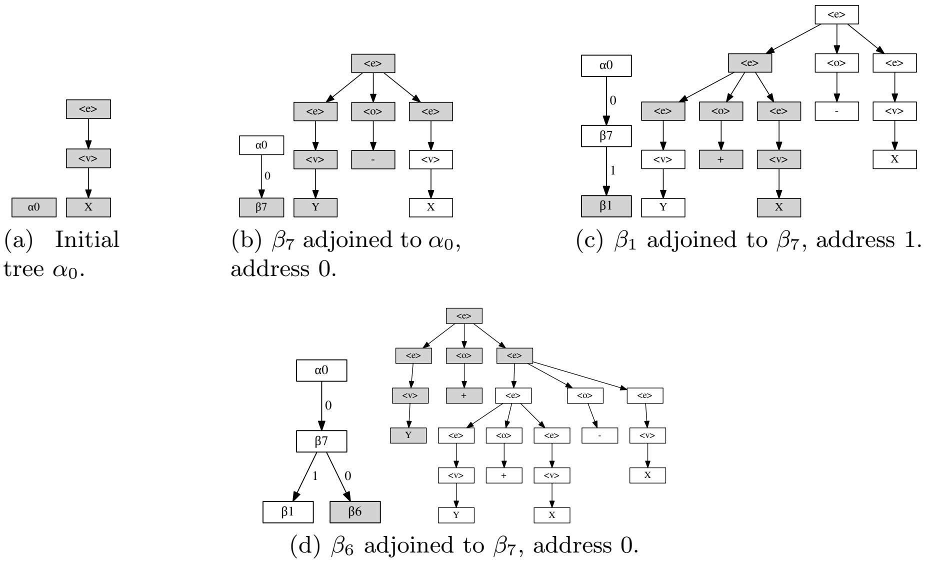


Fig. 1. Sample GE grammar, chromosome and resulting derivation tree (edge labels indicating the order of expansion). <> denotes a non-terminal symbol.



(a) I={α_0, α_1} (b) A={β_0, β_1, β_2...}

Fig. 2. The initial tree set (I) and a subset of the auxiliary tree set (A) of the TAG produced from the CFG in Fig. 1



(a) Initial tree α_0.

(b) β_7 adjoined to α_0, address 0.

(c) β_1 adjoined to β_7, address 1.

(d) β_6 adjoined to β_7, address 0.

Fig. 3. The derivation tree (left) and derived tree (right) throughout TAGE derivation. The shaded areas indicate new content added at each step.

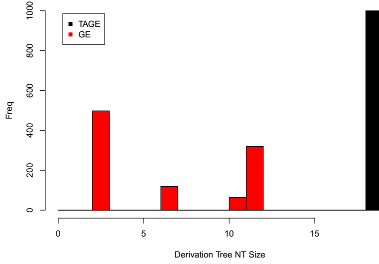


Fig. 4. The distributions of tree size (NT nodes) when initialising to a common genotype length

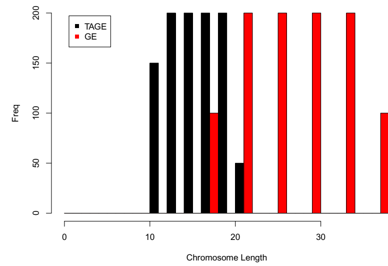


Fig. 5. The distributions of genotype lengths when initialising to common distribution of derivation tree sizes

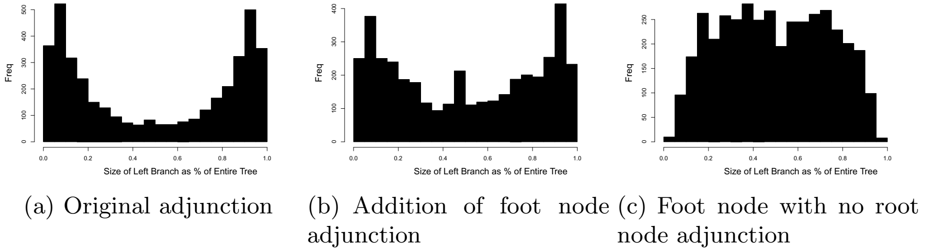
4.1 Initialisation and Transformation Bias

While the typical method of initialisation in GP is the Ramped Half and Half method [7], dividing the population between a minimum and maximum depth interval with half the trees being grown randomly and the other half being grown to be full trees, depth is not as important in GE as in GP. In GE, to ensure that there is a good distribution of phenotypes, the distribution of the number of NT nodes, or tree size is more important. With that in mind, a ramped tree size with a max depth initialisation method is employed by this study (similar to the method used by Harper [2], and PTC2 by Luke [9] without probability tables).

As mentioned in Section 3, it is possible to generate a lexicalised TAG from a finitely ambiguous CFG. However, there are biases inherent to TAGs which affect the probabilities of certain shapes being generated, as well as biases inherent to CFGs that are not preserved by this grammar transformation. Specifically these are an adjunction bias, biases imposed upon the language by the choice of adjunction points, and a grammar transformation bias, introduced when transforming from one grammar type to another. More detail on these is given below.

4.2 Adjunction Bias

While TAGs are said to be both weakly and strongly equivalent to the CFG used to generate them [5], depending on the constraints imposed upon the adjunction operation, biases appear in the shapes of randomly generated derived trees. For example, in the initial implementation of TAGE [12] adjunction is not allowed to be performed on foot nodes of auxiliary trees already in the derivation tree. The result of which is that once an adjunction is performed, the tree can only be expanded at its other adjunctable addresses, preventing the branch contained by the foot node from being expanded. Fig. 6(a) shows the distribution of tree shapes when using the adjunction constraints from Murphy et al. [12]. Tree shape is measured as the percentage of NT nodes used to build the left branch of the tree. This figure shows that the distribution of tree shapes are heavily skewed



(a) Original adjunction

(b) Addition of foot node adjunction

(c) Foot node with no root node adjunction

Fig. 6. Adjunction biases affecting the shape of generated trees. The histograms plot the frequency of the percentage of nodes used by the left side of the tree. The initialisation method described in Section 4.1 was used to generate 4000 trees for each approach, with a maximum depth of 20, a minimum/maximum size interval of 21/220.

towards trees with either very large left or right branches, with very few full trees. Ideally this distribution would be even across the entire spectrum of tree shapes favouring no particular shape. Fig. 6(b) shows that the distribution of tree shapes begins to level out once foot node adjunction is allowed.

In addition, allowing adjunction at the root nodes of auxiliary trees can have a similar but more pronounced effect on the form of the tree shape distribution. If at any point during derivation, an adjunction is performed on the top-most adjunctable address, usually the root of an auxiliary tree, the derived tree below this point of adjunction becomes the a sub-tree of the new tree's foot node, with the remainder of the new auxiliary tree off to one side. This causes the shape of the tree to be heavily skewed. By eliminating the adjunction at the root nodes of auxiliary trees, the tree shape distribution become much more level (see Fig. 6(c)). The probability of the tree reverting to a less skewed state depends on the ratio of adjunctable addresses available in the new auxiliary tree (usually quite small) to the adjunctable addresses in the displaced sub tree.

4.3 Grammar Transformation Bias

The probability of a specific terminal production being selected when generating a word using a CFG not only depends on the probability of that terminal production being selected within its own rule, but also on the probability of selecting each preceding production in order to reach the current rule from the start symbol. When transforming a CFG into a TAG these biases are lost and while it can be argued that this is a feature of the TAG representation, it can have unexpected effects for certain types of grammars.

For example, when generating a word from the balanced CFG presented in Fig. 7, whose derivation does not contain any recursive productions, there is a 0.5 chance of selecting `x` or `<digit>`. If `<digit>` is chosen there is a 0.1 chance of selecting any of the digits. From this it can be seen that even though there are 11 different words which could be generated, there is an equal probability of ending up with either an `x` or any one of the digits. When the CFG is transformed into a

```
<code>  := <value>
<value> := <value> <value> + |
           <value> <value> - |
           <value> <value> * |
           <value> <value> / |
           <digit> | <digit>
           x | x
<digit> := 0 | 1 | 2 | ... | 9
```

Fig. 7. A sample balanced grammar (equal probability of recursion and termination)

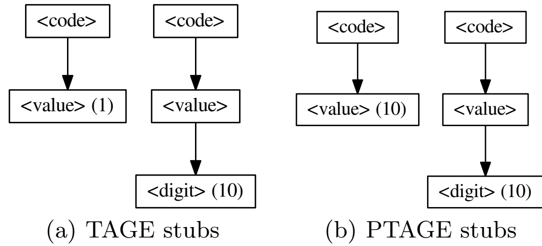


Fig. 8. TAGE tree stubs with a $0.\overline{09}$ chance of selecting `x`. Equivalent PTAGE stubs with a 0.5 chance.

TAG there are 11 trees to chose from, one with an `x` on the frontier and ten with a digit. Consequently, there is now a $0.\overline{09}$ chance of generating an `x`, as opposed to a 0.5 chance when using the CFG. This can make it difficult for certain words to be generated, both during initialisation and throughout a run.

In order to correct for the problems mentioned above, a novel method was designed which examines the probabilities contained within CFGs and applies them to the TAG. This method, named Probabilistic Tree-Adjoining Grammatical Evolution (PTAGE), is similar in theory to the structural and lexical biases imposed using TAGs by Hoai et al. [3] with TAG3P+. Whereas TAG3P+ uses properties of TAGs to impose language bias on the search, PTAGE's main function is to recreate the biases imposed by the CFGs used to generate each TAG. While this aids in generating similarly distributed initial populations, these biases affect the mapping process throughout the entire run.

In TAGE, the sets of initial and auxiliary trees are not generated at the beginning of the algorithm, but rather sets of elementary tree *stubs* are generated. This can greatly reduce the amount of memory needed to store the grammar. An elementary tree stub is an almost fully expanded elementary tree, with the terminal symbol leaf nodes excluded. In their place is a number representing the total number of different terminal nodes (variations) that can be attached at that point to complete the tree. For example, continuing with the sample grammar from Fig. 7 above, there would be a stub with the number `1` rather than an `x` and another stub with the number `10` rather than 10 different trees each with one of the digits, `0` through `9` (see Fig. 8(a)). The process of expanding a stub into a complete elementary tree is explained in the proceeding paragraph.

When choosing a tree in TAGE, the modulus operation is performed on the codon value and the total number of trees to select from, resulting in a number, c, between zero and the total number of trees minus one. If c has been used before, the correct tree is retrieved directly from a map. Alternatively, if c has not been seen before, each stub's variations are summed in order until the sum is greater than c in order to find the correct stub to expand. Then, proceeding in a depth first manner, the stub is completed by visiting each NT leaf node and dividing c by the product of the variations of all the NT leaves visited so far

while expanding that stub, performing the modulus operation on this product and the number of possible variations at the current NT node. This results in a number between zero and the total variations possible at that node, allowing the selection of the correct terminal production to expand the node by. This process continues until there are no more NT nodes to expand, storing the complete tree in a map for later use before being returned.

PTAGE examines the CFG and updates the number of variations at each stub's leaf nodes to reflect the probability of reaching those terminal symbols when expanding using the original CFG. In this example, since there should be an equal chance of generating an `x` as a digit, the variations on that stub are updated from `1` to `10` (as shown in Fig. 8(b)). The effect of this is that when selecting a tree there are now twenty trees to chose from, ten `x` trees and a single tree for each digit.

5 Experiments

The focus of this study is to improve the similarity of the initial setup of both GE and TAGE by examining the language biases which affect this, enabling a better comparison of performance and behaviour. As such, the experiments run in the initial study [12] are repeated twice here. First, using only the new method of initialisation and a second time incorporating the modified adjunction constraints (adjunction at foot nodes and no adjunction at root nodes).

Four benchmarks are used for this study, *Even Five Parity*, *Santa Fe Ant Trail*, *Symbolic Regression* and *Six Multiplexer*.The grammars used are identical to those used by Murphy et al. [12] apart from those of Symbolic Regression and Six Multiplexer, which were each given a new extra start symbol (see Fig. 9). This grammar change does not affect the behaviour of either algorithm but enables the disabling of root node adjunction. 100 independent runs were performed for each of GE, TAGE and PTAGE on both setups, the first using the new initialisation method, outlined in Sec. 4.1, with the original adjunction constraints (**NI**), and a second time using the new adjunction constraints (**NA**), outlined in Sec. 4.2. See Table 1 for the GE parameters.

6 Results and Discussion

6.1 Initialisation

It is clear from Table 2 that the improved initialisation of the population (using the original adjunction addresses) has a dramatic effect on the performance of GE. Improvements range from an increase of ∼10% (Even Parity) to almost an increase of an order of magnitude (Santa Fe) in the success rate. The new initialisation method did not have an effect of the same magnitude on TAGE, with improvements in success rate between ∼10% and ∼80%. As a result of this, GE's performance has surpassed that of TAGE on the Santa Fe Ant Trail problem, with TAGE showing superior performance on the remaining three problems.

Table 1. GE parameters adopted for each of the benchmark problems

Parameter	Value
Generations	200
Population Size	100
Initialisation	Ramped NT Size with Max Depth
Min NT Size	21
Max NT Size	70
Max Depth	10
Max Chromosome Wraps	0
Replacement Strategy	Generational
Elitism	10 Individuals
Selection Operation	Tournament
Tournament Size	3
One Point Crossover Prob	0.9
Integer Mutation Prob	0.02

```
<prog> ::= <expr>
<expr> ::= ( <op> <expr> <expr> ) | <var>
<op>   ::= + | - | *
<var>  ::= x0 | 1.0
```

(a) Symbolic Regression Grammar

```
<prog> ::=  <B>
<B>    ::= (<B>) && (<B>) | (<B>) "||" (<B>)
         | !(<B>) | (<B>) ? (<B>) : (<B>)
         | a0 | a1 | d0| d1 | d2 | d3
```

(b) Six Multiplexer Grammar

Fig. 9. Updated grammars for Symbolic Regression and Six Multiplexer problems

Table 2. The number of successful runs out of the 100 runs performed for all setups. **GE** and **TAGE** are the results from the original comparison (**NI** indicates the use of the new initialisation method and **NA** indicates the use of the new adjunction addresses).

	Even 5	**Santa Fe**	**Sym. Reg.**	**Six Multi.**
GE	79	3	44	6
TAGE	88	12	76	63
GE-NI	88	28	75	18
TAGE-NI	100	22	99	72
PTAGE-NI	100	13	99	20
TAGE-NI-NA	98	23	98	78
PTAGE-NI-NA	98	14	98	34

6.2 The Effect of PTAGE

From Table. 2 it can be seen that the application of PTAGE on two of the problems examined, Santa Fe and Six Multiplexer, has had a negative effect on performance. By examining the grammars of these two problems it can be noted that as a result of the transformation from CFG to TAG a bias is introduced, causing the selection of certain structures to be favoured over others. In the case of the Six Multiplexer problem, there is a probability of ~ 0.81 of selecting a tree containing `<B>?<B>:<B>` compared to a ~ 0.18 chance of selecting a tree containing either `<B> || <B>` or `<B> && <B>` or a < 0.01 chance of selecting a tree containing `!(<B>)`. This bias causes the TAGE tree to grow much wider than when using the PTAGE approach, as PTAGE balances these probabilities to be equal since they have an equal chance of being selected when deriving using CFGs. Similar effects are observed in the Santa Fe grammar. PTAGE has no effect on the other problems as the grammar transformation does not introduce any new biases.

6.3 New Adjunction Addresses

The inclusion of adjunction at root nodes and the exclusion of adjunction at the foot nodes appear to have only a marginal difference on performance, whereas they had a significant difference on the distribution of tree shapes as was seen in Fig. 6. This might suggest that TAGE benefits less from having a more diverse initial population than GE. This could be a result of the greater connectivity observed in TAGE landscapes in [11].

7 Conclusions

In the process of creating an initialisation method which generates similar sets of trees for both GE and TAGE, biases in the shape of the trees being generated by TAGE were detected. These biases were introduced due to the constraints placed upon the adjunction operation by TAGE as well as by the transformation algorithm used to generate TAGs from the original CFGs. New adjunction constraints and a system to eliminate the transformation biases, PTAGE, were described. It was noted that the transformation biases can be beneficial to the algorithm but are dependant on the grammar and the problem in question.

Subsequently, by improving the initialisation method used for the comparison of GE and TAGE, ensuring that similar distributions of trees sizes and shapes were created by each setup, it was seen that while there does not appear to be a statistically significant improvement in performance of TAGE over that of GE as was suggested in [12], TAGE still manages to generate more successful solutions in three of the four problems.

Interesting future work prompted as a result of this study includes examining the trends of the distributions of derivation tree shapes and sizes over the course of a run. Investigating these trends with both commonly distributed initial genotypic populations, as well as phenotypic populations, might give better insight into why a more diverse initial population appeared to be more beneficial to GE than TAGE.

Acknowledgements. This research is based upon work supported by the Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References

[1] Daida, J.M., Ampy, D.S., Ratanasavetavadhana, M., Li, H., Chaudhri, O.A.: Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1851–1858. Morgan Kaufmann, Orlando (1999)

[2] Harper, R.: GE, explosive grammars and the lasting legacy of bad initialisation. In: IEEE Congress on Evolutionary Computation (CEC 2010). IEEE Press, Barcelona (2010)

- [3] Nguyen, X.H., McKay, R.I., Abbass, H.A.: Tree adjoining grammars, language bias, and genetic programming. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 335–344. Springer, Heidelberg (2003)
- [4] Nguyen, X.H., McKay, R., I(B.), E.D.L., Abbass, H.A.: Toward an Alternative Comparison between Different Genetic Programming Systems. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) EuroGP 2004. LNCS, vol. 3003, pp. 67–77. Springer, Heidelberg (2004)
- [5] Joshi, A.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions, ch. 6, pp. 205–250. Cambridge University Press, New York (1985)
- [6] Joshi, A., Schabes, Y.: Tree-Adjoining Grammars. In: Handbook of Formal Languages, Beyond Words, vol. 3, pp. 69–123 (1997)
- [7] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
- [8] Kroch, A., Joshi, A.: The Linguistic Relevance of Tree Adjoining Grammar, Technical Report, University Of Pennsylvania (1985)
- [9] Luke, S.: Two fast tree-creation algorithms for genetic programming. IEEE Transactions on Evolutionary Computation 4(3), 274–283 (2000)
- [10] McKay, R., Hoai, N., Whigham, P., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. Genetic Programming and Evolvable Machines 11, 365–396 (2010)
- [11] Murphy, E., O'Neill, M., Brabazon, A.: Examining Mutation Landscapes In Grammar Based Genetic Programming. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 130–141. Springer, Heidelberg (2011)
- [12] Murphy, E., O'Neill, M., Galvan-Lopez, E., Brabazon, A.: Tree-adjunct grammatical evolution. In: 2010 IEEE World Congress on Computational Intelligence, pp. 4449–4456. IEEE Computational Intelligence Society, IEEE Press, Barcelona, Spain (2010)
- [13] O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language. Genetic programming, vol. 4. Kluwer Academic Publishers (2003)
- [14] Whigham, P.A.: Grammatical Bias for Evolutionary Learning. Ph.D. thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia (1996)

Improving Relevance Measures Using Genetic Programming

Kourosh Neshatian and Mengjie Zhang

School of Engineering and Computer Science
Victoria University of Wellington, P.O. Box 600, Wellington, New Zealand
{kourosh.neshatian,mengjie.zhang}@ecs.vuw.ac.nz

Abstract. Relevance is a central concept in many feature selection algorithms. Given a relevance measure, a feature selection algorithm searches for a subset of features that maximise the relevance between the subset and target concepts. This paper first shows how relevance measures that rely on the posterior estimation such as information theory measures may fail to quantify the actual utility of subsets of features in certain situations. The paper then proposes a solution based on Genetic Programming which can improve the usability of these measures. The paper is focused on classification problems with numeric features.

Keywords: Genetic programming, Relevance measure, Binary classification, Multivariate dependency analysis.

1 Introduction

Feature selection algorithms have—in abstract terms—two main components: a search mechanism and an evaluation mechanism. The search mechanism searches the search space (the set of all subsets of features in a problem) and generates candidate solutions (subsets of features). Most search mechanisms (perhaps all except random and exhaustive search) use the information obtained by evaluating the subsets of features to move in the search space and generate new candidates. The evaluation mechanism is based on a measure of utility of a subset of features.

The most common measure of utility in the filter approach is *relevance*. Relevance quantifies the degree of relatedness between a subset of features and another feature (that does not exist in the subset). Features with a significant degree of relevance to target concepts (such as class labels) are desired while features with a considerable degree of relevance to each other are considered *redundant* and thus unwanted.

Examples of commonly-used relevance measures are those based on information theory such as *mutual information* and *information gain ratio* [6], and statistical measures such as Pearson product-moment correlation coefficients.

Information theory measures are usually used to measure the relevance between an input feature and the decision variable (the class label variable in a classification task). The calculation of information-theory measures depends on

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 97–108, 2012.
© Springer-Verlag Berlin Heidelberg 2012

estimating probability densities. We will show how the scarcity of observations (i.e. number of examples in a training data set) and lack of prior knowledge in the problem domain can lead to poor density estimation which in turn causes the relevance measure to give a wrong account of utility of the subset of features at hand.

Genetic programming (GP) is a hyper-heuristic search algorithm that evolves computer programs [5,9]. GP is capable of building fairly complex logical and mathematical expressions using a set of primitive building blocks (functions and terminals) in order to optimise an objective function. GP has been used to detect complex redundant relationships between features [7]. This paper proposes a method to use GP to evolve functions that can be combined with traditional relevance measures in order to deal with circumstances that traditional relevance measures are unable to.

2 Relevance Measures

A good subset of features must be 'relevant' to the response variable. The concept of relevance has been formally investigated separately by Keynes, Carnap and Gardenfors [1,3,4]. According to their work, relevance can be expressed using the concepts of probability and independence: a feature X is relevant to a response variable Y on the basis of a prior evidence (here a subset of features $\mathcal{S}$ that is already available) if and only if

$$P\{C = c|X = x, \mathcal{S}\} \quad \neq \quad P\{C = c|\mathcal{S}\} \tag{1}$$

that is, the feature X is relevant if knowing the value of the feature can change the probability distribution of the concept variable; that is, C depends on X. While this definition gives a good logical foundation for relevance, it cannot help with quantifying relevance. This is mainly because in real-world problems, the conditional probability distribution of Y is unknown and depending on how one estimates the distribution, the meaning and the quotient of relevance changes.

From an optimisation perspective, feature selection depends on relevance measures. In a supervised learning scenario where there are a number of input variable (features) and one or more decision (output) variables, the goal of feature selection is to find a subset of input features that has maximum relevance to the target variables and the features in the subset have minimum relevance among each other.

Information theory has a measure of purity called *information entropy* which is used to measure the information content (aka *uncertainty*) of a communication channel. The most common way of measuring entropy is perhaps *Shannon's entropy*. Let $\mathbb{C}$ be the set of all possible class labels in a classification problem and C be a categorical response variable that takes its values from $\mathbb{C}$. The Shannon entropy of C is defined as

$$H(C) = -\sum_{c \in \mathbb{C}} p(c) \log_2 p(c) \ . \tag{2}$$

When binary encoding is used, the base of the logarithm must be 2 but in the context of measuring relevance it could essentially be any positive value. The entropy of a variable measures its information content (number of bits required to encode the variable). The entropy of C will be the highest if all class labels are equally likely and it will be the lowest if the probability of one of the class labels is 1 and the rest are 0.

The conditional entropy of C measures its information content given the value of an input feature and is defined as

$$H(C|X) = -\sum_{x \in \mathcal{X}} p(x) \sum_{c \in \mathbb{C}} p(c|x) \log_2 p(c|x) \tag{3}$$

where x is a categorical (nominal) input feature which takes its values from $\mathcal{X}$. The feature is considered relevant if the above conditional entropy is lower than $H(C)$—that is, knowing the value of the feature, some classes become more likely than others and thus the conditional entropy would drop. This notion of relevance can be quantified by *mutual information* which is defined as

$$I(C;X) = H(C) - H(C|X) \ . \tag{4}$$

Equations 3 is for a single categorical variable (input feature). Thus equation 4 is also for a single categorical variable. Let $\mathbb{F}$ be the set of all input features in a classification problem and $\mathcal{S} \subseteq \mathbb{F}$ where the cardinality $|\mathcal{S}| = m$. If all the features in subset $\mathcal{S}$ are categorical then the multi-variate conditional entropy can be expressed as

$$H(C|\mathcal{S}) = H(C|X_1, X_2, \ldots, X_m) = -\sum_{x_1 \in \mathcal{X}_1, x_2 \in \mathcal{X}_2, \ldots, x_m \in \mathcal{X}_m} p(x_1, x_2, \ldots, x_m) \times \sum_{c \in C} p(c|x_1, x_2, \ldots, x_m) \log_2 p(c|x_1, x_2, \ldots, x_m) \tag{5}$$

where the joint probability density (mass) function can be estimated by computing the contingency table. Using equation 5, the mutual information of a subset of features and the class variable is defined by

$$I(C;\mathcal{S}) = H(C) - H(C|\mathcal{S})$$

When the input features are numeric the sums in equation 5 change to integrals:

$$H(C|\mathcal{S}) = H(C|X_1, X_2, \ldots, X_m) = \int \ldots \int p(x_1, x_2, \ldots, x_m) \tag{6}$$

$$\times \sum_{c \in C} p(c|x_1, x_2, \ldots, x_m) \log_2 p(c|x_1, x_2, \ldots, x_m) dx_1 \ldots dx_m \ . \tag{7}$$

Estimating the joint density however, when the input features are numeric, is not trivial and often not feasible. As the number of input features increases more

and more examples (training data) are required in order to have a reasonable estimate for the density function (the curse of dimensionality).

To avoid non-parametric estimation, two strategies are used at the same time: i) numeric features are discretised, ii) the relevance of features are measured individually and then added together as heuristic approximation for the relevance of a subset of features. For example a widely-accepted heuristic measure is defined as [8]:

$$D(C, \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{X \in \mathcal{S}} I(C; X) \ . \tag{8}$$

In the next of this section we discuss how these widely-used strategies can lead to a poor judgement on relevance.

2.1 Deficiency in Handling Multi-modal Distributions

Figure 1 shows a binary classification problem with a single feature x, whose conditional distribution—conditioned on the positive class—is bimodal. The conditional distribution of x given the negative class is Gaussian and thus unimodal. 100 examples have been sampled from each class. The examples are plotted under the distribution curves. There is very little overlap between the two class. The feature x in this example is observably good because a near-perfect classification model can be obtained by computing over this feature. For example a simple model could be learnt by finding an interval for the negative class (approximately from -2.5 to 2.5), and then unseen examples would be assigned to the negative class if their x value falls in this interval, and to the positive class otherwise.

When mutual information is used to measure the relevance of X to C in this problem, the algorithm tries to find the best split point using which, instances from different classes can be separated. The original entropy of examples without the presence of any features, $H(C)$, is 1 (one bit). Using one-split-point discretisation, the numeric feature, X, is converted to a discrete binary event X'.

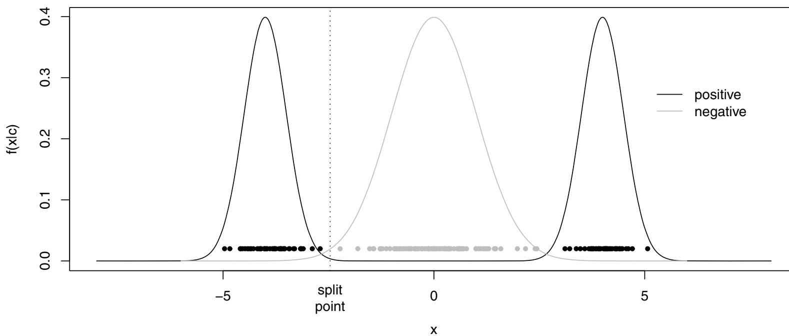


Fig. 1. A binary classification problem where the conditional distribution of x given the positive class is bimodal (the black curve). The vertical line shows a split point on the x axis around which partitioning yields the highest information gain.

The best split point (using a one-pass search) is at -2.46. Now $p(C|X')$ is easy to estimate and $H(C|X')$ can be computed. After computation, the value of $I(C;X')$ turns out to be 0.327 (instead of 1.0) which incorrectly implies that the feature is not very useful. This is because only a small proportion of instances can be correctly described by a single-point discretisation.

For a single-feature bimodal problem like the one presented in Figure 1, the issue can be resolved by performing multiple-point discretisation. However, multiple-point discretisation is not always feasible because: i) determining the optimal number of split points is not trivial; ii) the search time considerably increases as the number of split points increases; iii) too many split points may result in overfitting; iv) the three previous issues become exponentially more problematic as the dimensionality of the problem increases. In general, if the distribution of one of the classes is multi-modal, the features can easily be dismissed as irrelevant due to poor discretisation.

2.2 Deficiency in Handling Non-orthogonal Multi-variate Relationships

Another deficiency of single-feature evaluation and discretisation is illustrated in Figure 2. The figure shows a sample from a binary classification problem where the two classes are easily separable; for example, a straight line can separate the two classes.

The two features, X and Y, are certainly relevant as a linear model based on the two features can completely describe (and predict) the conditional distribution of classes. However, since the class boundary (the imaginary line between the instances of the two classes) is not orthogonal to either of the axes (features), discretisation along X or Y individually, would not yield any information gain. In other words, if the cloud of data is projected to either of the axes, the resulting conditional marginal densities will be overlapping. For this reason heuristic measures that rely on single-feature discretisation cannot measure the relevance of the features (together) correctly.

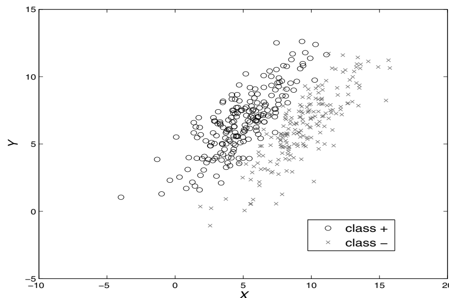


Fig. 2. A sample from a binary classification task plotted against its two input features, x and y

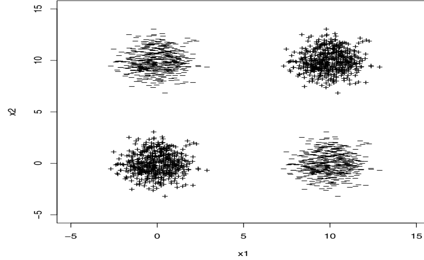


Fig. 3. A binary classification problem ('+' vs '-') with non-linear class boundaries

For the data set in Figure 2, the original entropy (without any input features) is 1 (one bit). The best split points along X and Y are at 7.4 and 4.6 respectively. The mutual information of the two features are $I(C;X) = 0.42$ and $I(C;Y) = 0.00$ and thus the relevance of the subset according to equation 8 is $D(C, \{X, Y\}) = 0.21$. This is of course far from the actual utility of the subset, $I(C; \{X, Y\} = 1$, which can be obtained by discovering the linear relationship between C and the two features[1].

The reason for underestimating the relevance of this subset is that very little information (separation) can be obtained by using only one of the features. The situation might improve to some extent if one used equation 4 and estimated the joint conditional probability $p(c|x, y)$. However, this would not be a good/feasible solution for two reasons: i) as the dimensionality of the problem increases, the need for training examples increases exponentially. ii) a grid-like partitioning does not give a smooth and realistic picture of the density necessarily. The latter too becomes more severe as the dimensionality increases.

2.3 Deficiency in Handling Epistatic Relationships

Very often the relationship between a subset of features and the class variable is epistatic; that is, the contribution of a feature in predicting the class label will depend on the value of some other features. An example of such a situation is presented in Figure 3 which shows a sample from a binary classification problem with two input features. The dispersion of the instances of the two classes form an XOR-like problem in a two-dimensional numeric space.

The two features in Figure 3 are both important. Partitioning the input space into four subdivision using two straight lines can produce a complete classification model in this problem; that is, a decision tree can completely describe the data.

[1] One might think that principle component analysis (PCA) can resolve the issue in these types of problems. However, note that since PCA is an unsupervised transformation—the input space is (linearly) transformed regardless of class labels—the class boundary will not necessarily be orthogonal to any of the resulting input axes (components).

Non-linear classification learning algorithms such as ANNs and SVMs can also describe the data using the two features. On the other hand, according to equation 4, $I(C; X_1) = 0$ and $I(C; X_2) = 0$; that is, neither X_1 nor X_2 are individually important. Thus heuristic relevance measures that reduce the concept of relevance (utility) of a subset, to the relevance (utility) of its elements, fail to recognise the importance of the two features in this problem[2]. For example according to equation 8, $D(C, \{X_1, X_2\}) = 0$.

3 Using GP for Partitioning the Input Space

Estimating the conditional joint probability distribution as expressed in equations 5 and 7 is not feasible because the number of required training examples grows exponentially with respect to the number of dimensions. On the other hand, estimating distribution by partitioning single input features (discretisation) does not yield a good picture of relevance either. To improve the situation, we propose a GP-based approach to partitioning the input space that does not have the drawbacks of single feature discretisation.

We extend the traditional concept of partitioning—in which a one-dimensional input space is divided into several segments—to a new model in which every point in the multi-dimensional input space belongs to two complementary fuzzy sets (partitions) A and A^c with membership functions μ_A and μ_{A^c}. Since A^c is the complement of A, $\mu_{A^c}(.) = 1 - \mu_A(.)$. μ_A is function of the form $\mu_A : \mathbb{R}^n \mapsto [0, 1]$ where n is the number of input dimensions. Given the two complementary sets A and A^c, the conditional probabilities $p(C|A)$ and $p(C|A^c)$ are simply estimated by measuring the frequency of positive and negative examples in A and A^c partitions. After estimation, $I(C; A)$ can be calculated. If a particular partitioning (i.e a particular definition of μ_A) yields high mutual information $I(C; A)$, it can be concluded that the set of original features $\mathcal{S}$ participating in the definition of μ_A are relevant to C. This is because the value of μ_A is completely determined by the values of features in $\mathcal{S}$.

We use untyped (uni-type) GP to evolve appropriate membership functions. The terminals (constants) and variable terminals (input features) are all numeric (pseudo-real numbers). The function set contains binary and unary primitive functions that take and return numeric values. Using this configuration, the codomain of a GP program φ is $\mathbb{R}$. We use the following equation to map the codomain into $[0, 1]$ so it would be a valid membership function.

$$\mu_A(x_1, x_2, \ldots, x_n) = \frac{\tanh(\varphi(x_1, x_2, \ldots, x_n)) + 1}{2} \tag{9}$$

[2] Even learning algorithms that examine features individually cannot find a complete solution for this problem. For example typical decision tree learners (such as C4.5) that use a greedy search to find a proper tree cannot find the right decision tree even though it is in their search space. The average performance (accuracy) of J48-induced decision trees over the data in Figure 3 is 50% which is equivalent to a random classifier.

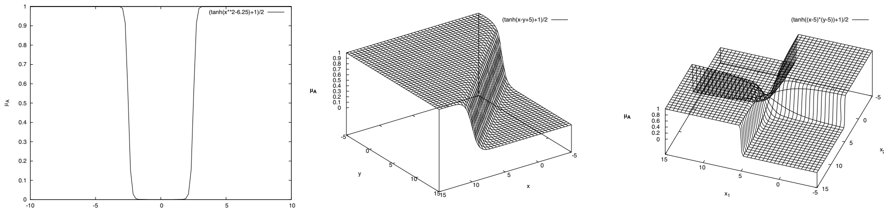


Fig. 4. Partition membership according to the membership functions evolved by GP for the problems presented in Figures 1, 2 and 3. The corresponding evolved programs from left to right are `(- (* X X) 6.25)`, `(+ (- X Y) 5)` and `(* (- X1 5) (- X2 5))`.

Now this can be turned to an optimisation problem in which the goal is to maximise $I(C; A)$ (the fitness function). Note that unlike traditional partitioning, in the proposed method, the resulting partitions are not necessarily connected to each other—that is, partition A may be scattered across different regions in the input space. The relevance of a subset $\mathcal{S}$ in this model is calculated by

$$IGP(C; \mathcal{S}) = H(C) - p(A)H(C|A) - p(A^c)H(C|A^c)$$

Figure 4 shows the plot of membership functions μ_A evolved by GP for the problems presented in Figures 1, 2 and 3. μ_{A^c} has not been plotted as it is simply $1 - \mu_A(.)$. Notice the transition from high membership values to low values happens in areas of the input space where there is a significant change in the conditional density of the positive and the negative classes[3]. Area with high membership values (e.g. greater than 0.5) are considered belonging to one partition. This new way of partitioning would give a better picture of the density function as the density of the positive and negative class in A and A^c are quite different. In all three cases in Figure 4, $I(C; \mathcal{S})$ is close to 1.

In abstract terms, we are using GP to see if there is a way of partitioning the input space such that one of the classes (positive or negative) is more likely in A than the other. If this happens then the conditional entropy of C decreases which implies the set of features used in the definition of μ_A are relevant to C. This is in essence similar to using GP for classification where GP evolves discriminant functions.

4 Empirical Evaluation

Most relevance measures do intrinsically well at judging *irrelevant* subsets of features; they return a zero (or close to zero) degree of relevance for such subsets. This is because relevance is estimated based on the usefulness of a subset of features in predicting the behaviour of a target variable. When a subset of features is irrelevant to learning a concept, the relevance measure cannot find any predictive relationship between the subset and the target concept and thus

[3] This is, in a way, parallel with the concept of edge detection in signal processing

judges the subset as irrelevant. The only two situations in which an irrelevant subset may seem relevant to a subset are: **i**) when there is a *severe* scarcity of observations (training data). For example having only two observations (H, H) and (T, T) from tossing two coins may wrongfully imply that the two coins will always produce the same results. **ii**) when there is overfitting. Both situations can be dealt with by providing more training data. The second situation can also be dealt with by controlling the model complexity (for example by putting a limit on the depth of GP programs or penalising lengthy descriptions of data [10]).

Discovering complex relationships between a subset of features and target concepts is a challenging task for most relevance measures. In the previous section we saw some example situations in which relevance measures had difficulty in discovering the relationship between input features and the target concept and therefore judged relevant features as not being very relevant. In this section, we evaluate the performance of the proposed GP-based measure and see how it compares to the conventional way of measuring relevance (density estimation by partitioning).

As it is not known into what extent features that are measurements on natural phenomena (such as the blood pressure of a patient) are actually relevant to target concepts in a problem domain (e.g. the concept of illness), we focus our studies on synthesised classification problems in which a subset of features is completely relevant—that is, it is possible to find a function (e.g. a classifier) of features in the subset that can correctly and completely explain the data (predict the class labels).

4.1 Synthesising Data

To evaluate the performance of the proposed system a number of different binary classification problems are synthesised. All the synthesised classification problems have ten numeric input features and one categorical feature for the class label. The problems are generated by a mechanism similar to the population initialisation in GP. For each problem a random mathematical expression φ is generated using the *growth* method of GP. The variable terminal set is $\{X_1, X_2, \ldots, X_{10}\}$, the function set is $\{+, -, \times, \div\}$, and set of randomly-generated constant numeric values is also used for terminal nodes. Each expression φ is a mapping (function) of the form $\mathbb{R}^{10} \mapsto \mathbb{R}$ (even though it may not contain all ten input variable terminals). We set the log of odds for having a positive instance given that the instance has been observed at $(x_1, x_2, \ldots, x_{10})$ as following:

$$\ln \frac{p(C = positive|X_1 = x_1, X_2 = x_2, \ldots, X_{10} = x_{10})}{p(C = negative|X_1 = x_1, X_2 = x_2, \ldots, X_{10} = x_{10})} = \varphi(x_1, x_2, \ldots, x_{10})$$

and thus the probability of having a positive instance at that point is

$$p(C = positive|X_1 = x_1, X_2 = x_2, \ldots, X_{10} = x_{10}) = \frac{1}{1 + e^{-\varphi(x_1, x_2, \ldots, x_{10})}}$$

and $p(C = negative|\dots) = 1 - p(C = positive|\dots)$. In other words, the output of a genetic program—which could be anywhere in the interval $(-\infty, +\infty)$—is mapped into the interval $[0, 1]$ and is used as the probability of the positive class for the corresponding point in the input domain.

Not all variable terminals are used in a randomly generated expression. Those that have not been used are considered irrelevant. Some of the variable terminals that have been used in an expression may also be irrelevant due to the *intron* effect in GP (e.g. a clause like `(- X X)`). To detect such variable terminals, a set of randomly generated number are fed to the variable while other variables are kept fixed to detect if there is any change in the output of the program. Problems (conditional densities) in which C is totally independent of the input (i.e. randomly-generated expressions that do not have any variable terminals in them) are removed. For each problem (for each generated expression φ) a data set is sampled and is used for the evaluation of the system. All the data sets are sampled in such a way that they are balanced.

4.2 GP Settings and Implementation Details

Conventional tree-based GP is used in all experiments. In this model, each program produces a single floating-point number at its root as the result of its evaluation (output). There is one variable terminal per input feature in the problem. A number of randomly generated constants are also used as terminals. The four standard arithmetic operators, $\{+, -, \times, \%\}$, were used to form the function set. The division operator is *protected*—that is, it returns zero for division by zero. All the members of the function set are binary—they take two parameters. The *ramped half-and-half* method is used for generating programs in the initial population and for the mutation operator. The probability of the crossover and mutation operators are adapted automatically at runtime [2]. An elitist approach has been taken to ensure that the performance of the fittest individual in the population never deteriorates. The population size is 1024 and the evolution is terminated, at the latest, after the 50th generation or when $I(C; A) = H(C)$ —that is the maximum information gain has been reached. The platform is implemented in Java and *grid computing* has been used in order to have parallel GP runs.

4.3 Results

Table 1 shows the result of our experiments. For each data set, we use $\mathcal{S}$ to refer to the relevant subset of features in that problem. The first row of the table is for problems where the cardinality of $\mathcal{S}$ is one, the second row is for problems where the cardinality is two and so forth. The number of problems in each row is represented by n. For example, according to the third row, there are 400 different data sets (densities) with ten input features in which the cardinality of the relevant subset of features is three.

Two average errors are presented in the table, one for the heuristic measure D and the other one for the proposed IGP measure. The errors show the difference

Table 1. Average Error in Relevance Measures

$\lvert\mathcal{S}\rvert$	n	$\overline{err}_D$	$\overline{err}_{IGP}$	$\hat{\mu}_d$	$\hat{\sigma}_d$	p-value
1	400	0.379	0.080	0.299	0.028	0.000 ⋆⋆⋆
2	400	0.534	0.134	0.400	0.047	0.000 ⋆⋆⋆
3	400	0.583	0.198	0.385	0.054	0.000 ⋆⋆⋆
4	200	0.561	0.288	0.273	0.087	0.000 ⋆⋆⋆
5	200	0.612	0.369	0.243	0.104	0.000 ⋆⋆⋆
6	200	0.603	0.448	0.155	0.110	0.000 ⋆⋆⋆
7	100	0.627	0.464	0.163	0.140	0.000 ⋆⋆⋆
8	50	0.656	0.523	0.133	0.156	0.000 ⋆⋆⋆
9	50	0.669	0.547	0.122	0.176	0.000 ⋆⋆⋆
10	50	0.674	0.588	0.086	0.198	0.001 ⋆⋆

between the reported relevance by the corresponding method and the actual relevance of $\mathcal{S}$. The actual relevance of $\mathcal{S}$ denoted by $I(C|\mathcal{S})$ is 1 in all the data sets for the following two reasons: i) since all the data sets are for binary classification and balanced (i.e. $p(C = positive) = p(C = negative)$), $H(C) = 1$; ii) since there exists a functional form based on $\mathcal{S}$ that can completely and correctly describe C, $H(C|\mathcal{S}) = 0$. The average error of the heuristic measure D is calculated by

$$\overline{err}_D = \frac{1}{n}\sum_{i=1}^{n}(I(c;\mathcal{S}_i) - D(c;\mathcal{S}_i)) = 1 - \frac{1}{n|\mathcal{S}_i|}\sum_{i=1}^{n}\sum_{X\in\mathcal{S}_i} I(C;X)$$

and the average error of the proposed GP method is calculated by

$$\overline{err}_{IGP} = \frac{1}{n}\sum_{i=1}^{n}(I(c;S_i) - IGP(c;S_i)) = 1 - \frac{1}{n}\sum_{i=1}^{n} I(C;\varphi(\mathcal{S}_i))$$

and in order to perform a paired statistical significance testing, the difference between the errors on each data set is defined as

$$d = \sum_{X\in\mathcal{S}} I(C;X) - I(C;\varphi(\mathcal{S})) \ .$$

The estimated mean and standard deviation of d are represented by $\hat{\mu}_d$ and $\hat{\sigma}_d$ in the table. The p-value is $p\{X \geq d\}$ where $X \sim N(0, \frac{\hat{\sigma}_d}{\sqrt{n}})$ and is used for statistical significance test. The stars in front of the p-values show the statistical significance at the given levels: $\star \equiv$ sig. at 95% confidence level, $\star\star \equiv$ sig. at 99% confidence level and $\star\star\star \equiv$ sig. at 99.9% confidence level. It can be seen that in all cases, the error of the proposed system in estimating relevance is significantly lower than the D heuristic.

5 Conclusion

Relevance measures that depend on estimating a probability density function in order to find out how a group of variables (features) help predict another variable,

can misjudge the relevance in a variety of situations. Many of the traditional methods deficiencies are related to the fact that estimating probabilities based on partitioning features individually does not usually give a good picture of relevance. The proposed GP-based approach can produce more realistic relevance degrees by finding a proper way of partitioning the input space (estimating the posterior). The new approach gives a better account of relevance by actually finding the relationship between class variable and input features.

References

1. Carnap, R.: Logical foundations of probability. University of Chicago Press (1967)
2. Davis, L.: Adapting operator probabilities in genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, pp. 61–69. Morgan Kaufmann Publishers Inc., George Mason University, United States (1989)
3. Gärdenfors, P.: On the logic of relevance. Synthese 37(3), 351–367 (1978)
4. Keynes, J.: A treatise on probability. Macmillan & Co., Ltd. (1921)
5. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
6. Last, M., K, A., Maimon, O.: Information-theoretic algorithm for feature selection. Pattern Recognition Letters 22, 799–811 (2001)
7. Neshatian, K., Zhang, M.: Unsupervised Elimination of Redundant Features Using Genetic Programming. In: Nicholson, A., Li, X. (eds.) AI 2009. LNCS, vol. 5866, pp. 432–442. Springer, Heidelberg (2009)
8. Peng, H., Long, F., Ding, C.: Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1226–1238 (2005)
9. Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programming. Lulu Enterprises, UK Ltd. (March 2008)
10. Quinlan, J.R.: Improved use of continuous attributes in C4. 5. Journal of Artificial Intelligence Research 4, 77–90 (1996)

An Investigation of Fitness Sharing with Semantic and Syntactic Distance Metrics

Quang Uy Nguyen[1], Xuan Hoai Nguyen[2], Michael O'Neill[3], and Alexandros Agapitos[3]

[1] Faculty of Information Technology, Military Technical Academy, Vietnam
[2] IT Research and Development Center, Hanoi University, Vietnam
[3] Natural Computing Research & Applications Group, University College Dublin, Ireland
{m.oneill,alexandros.agapitos}@ucd.ie, {quanguyhn,nxhoai}@gmail.com

Abstract. This paper investigates the efficiency of using semantic and syntactic distance metrics in fitness sharing with Genetic Programming (GP). We modify the implementation of fitness sharing to speed up its execution, and used two distance metrics in calculating the distance between individuals in fitness sharing: semantic distance and syntactic distance. We applied fitness sharing with these two distance metrics to a class of real-valued symbolic regression. Experimental results show that using semantic distance in fitness sharing helps to significantly improve the performance of GP more frequently, and results in faster execution times than with the syntactic distance. Moreover, we also analyse the impact of the fitness sharing parameters on GP performance helping to indicate appropriate values for fitness sharing using a semantic distance metric.

Keywords: Genetic programming, Fitness sharing, Semantic, Syntactic.

1 Introduction

Genetic Programming (GP) [1,2] is an evolutionary paradigm for automatically finding solutions for a problem. Since its introduction, GP has been applied to a wide range of fields [1], and routinely exhibits human-competitive performance [3]. In GP, one of the crucial properties that strongly affects its performance is the diversity and dispersion of the population [4,5,6,7]. The diversity and dispersion of a population represents its ability to cover different parts of the search space. Therefore, promoting dispersion and diversity is important for the efficiency of search. There have been a number of methods for enhancing diversity and dispersion [8,5,6,9,10,11], of these fitness sharing has been widely used in Genetic Algorithms (GA) and Genetic Programming.

In Genetic Algorithms, fitness sharing was introduced as a technique for maintaining population diversity [12,13]. The basic idea is to cluster the population

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 109–120, 2012.
© Springer-Verlag Berlin Heidelberg 2012

into a number of groups, based on their similarity with respect to a distance metric. Members of the same group are penalized by having to share fitness, while isolated individuals retain the full reward. In GP, Langdon is perhaps the first person who used fitness sharing to preserve population diversity [14]. In Langdon's work, the distance metric is based on the fitness of individuals. Then, Ekart and Nemeth [15] proposed a metric for fitness sharing based on syntactic (structural) distance between two tree-based individuals. The method was applied to a symbolic repression problem with some success. Following this McKay [16] used implicit fitness sharing, in which the reward for each fitness case is shared by all individuals that give the same output. However, this method is only applied to Boolean problems and is not available to directly apply to continuous real-valued problems.

In this paper, we propose an approach to fitness sharing based on a semantic distance metric. We compared the performance of GP using fitness sharing with semantic and syntactic metrics. We also analyse the impact of the parameters of fitness sharing with semantic distance on the performance of GP. The remainder of the paper is organised as follows. In the next section, we briefly describe fitness sharing, the way we modify fitness sharing to speed up its execution and two distance metrics used for implementing fitness sharing. The experimental settings are detailed in Section 3. The results of the experiments are presented and discussed in section 4. Section 5 concludes the paper and highlights some potential future work.

2 Methods

This section briefly presents fitness sharing. The manner in which we modify fitness sharing is discussed, and following this two distance metrics for implementing fitness sharing are detailed.

2.1 Fitness Sharing

Fitness sharing treats fitness as a shared resource of the population, and thus requires that similar individuals share their fitness. It lowers each population element's fitness by an amount equal to the number of similar individuals in the population. Typically, the shared fitness f'_i of an individual with the raw fitness f_i is simply calculated as follows [17].

$$f'_i = \frac{f_i}{m_i} \tag{1}$$

where m_i is the niche count which measures the approximate number of similar individuals with whom the fitness f_i is shared. The niche count of individual i is calculated by summing a sharing function over all members of the population.

$$m_i = \sum_{j=1}^{N} sh(d_{ij}) \tag{2}$$

where N denotes the population size and d_{ij} represents the distance between the individual i and the individual j. Hence, the sharing function sh measures the similarity level between two population elements, it returns one if the elements are identical, zero if their distance d_{ij} is higher than a threshold of dissimilarity, and an intermediate value at intermediate level of dissimilarity. The most widely used sharing function is given as follows:

$$sh(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma)^{\alpha} & \text{if } d_{ij}<\sigma \\ 0 & otherwise; \end{cases} \quad (3)$$

where σ denotes the threshold of dissimilarity (also the niche radius) and α is a constant parameter which regulates the shape of the sharing function. While α is commonly set to one with the resulting sharing function referred to as the triangular sharing function [13], in this paper, several values of σ will be tested to find a range of suitable values for symbolic regression problems.

The distance d_{ij} between two individuals i and j is characterized by a similarity metric based on either semantic or syntactic similarity. In this paper, we will compare the efficiency of fitness sharing with two similarity metrics: semantic versus syntactic similarity.

2.2 Modifying Fitness Sharing

In our experiments, we modify fitness sharing to speed up its execution. The first modification is the way to calculate the shared fitness. It can be seen that the shared fitness calculated in Equation 1 can only be used if the raw fitness favors bigger values, meaning that the bigger value is better. Since for symbolic regression, we use the raw fitness as the mean of the absolute error with the condition that smaller values are better, it can not directly use Equation 1 to calculate the shared fitness. Instead we use the following equation for quantifying the shared fitness.

$$f'_i = (f_i) * (m_i + 1) \quad (4)$$

The main drawback of fitness sharing is that the computation of the shared fitness for the entire population in each generation can be very time-consuming [17]. We alleviate this by calculating the niches for only a small subset of the population that is randomly sampled from the whole population. Let P be the number of individuals that are randomly sampled from the population for each individual niche that is being calculated. In our experiments, we will investigate different values of P to find the appropriate values for GP.

2.3 Syntactic Distance

To implement fitness sharing using a syntactic metric, a syntactic distance between any two trees is required. In this paper, we use an extended version of tree distance that has been use by Ekart and Nemeth [15]. In other words, the syntactic distance between two trees is calculated as follows:

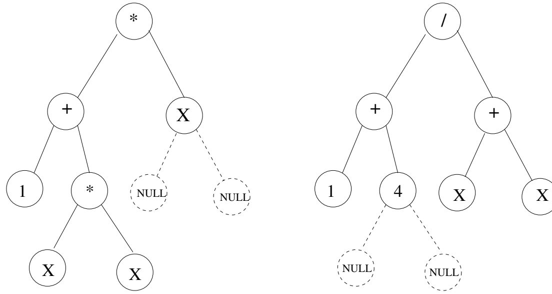


Fig. 1. Two trees are added the NULL nodes to have the same layout

1. Make the two trees to be compared to have the same tree-structure (adding NULL nodes if necessary). Figure 1 gives an example of two trees which are completed by adding NULL nodes so that they have the same structure.
2. Count the distance between any two nodes located at the same position in the two trees. If two nodes are labeled with the same symbol, the distance between them is 0, otherwise the distance is 1.
3. Sum the distances computed in the previous step to form the distance of the two trees.

2.4 Semantic Distance

To calculate the semantic distance between two individuals, a way to quantify their semantics must first be defined. In this paper, we use Sampling Semantics that has been used in previous work on semantic based crossovers [18,19]. Formally, sampling semantics between two trees (subtrees) is defined as follows:

Let F be a function expressed by a (sub)tree T on a domain D. Let P be a set of points sampled from domain D, $P = \{p_1, p_2, ..., p_N\}$. Then the *Sampling Semantics* of T on P on domain D is the set $S = \{s_1, s_2, ..., s_N\}$ where $s_i = F(p_i), i = 1, 2, ..., N$.

The value of N depends on the problem. If it is too small, the approximate semantics might be too coarse-grained and not sufficiently accurate. If N is too big, the approximate semantics might be more accurate, but more time consuming to measure. The choice of P is also important. If the members of P are too closely related to the GP function set (for example, π for trigonometric functions, or e for logarithmic functions), then the semantics might be misleading. For this reason, in this paper, the number of points for evaluating sampling semantics is set as the number of fitness cases of the problem (20 points), and we choose the set of fitness cases as the sample points for evaluating sampling semantics.

Based on *Sampling Semantics* (SS), we define a *Sampling Semantics Distance* between two trees. In the previous work [19], *Sampling Semantics Distance* (SSD) was defined as the sum of absolute difference of all values of SS. While the experiments show that this kind of SSD is acceptable, it has undoubted weakness that the value of SSD strongly depends of the number of SS points

(N) [19]. To soften this drawback, in this paper we use the mean of absolute distance as the SSD between trees. In other words, let $U = \{u_1, u_2, ..., u_N\}$ and $V = \{v_1, v_2, ..., v_N\}$ be the SS of $Tree_1(Tr_1)$ and $Tree_2(Tr_2)$ on the same set of evaluating values, then the SSD between Tr_1 and Tr_2 is defined as follows:

$$SSD(Tr_1, Tr_2) = \frac{|u_1 - v_1| + |u_2 - v_2| + ... + |u_N - v_N|}{N} \quad (5)$$

Since it could be expensive to compute SS, we reduce the cost by caching. The SS of each subtree is stored in the root node using attributes; the resulting GP system is known as *Attributes Genetic Programming* (AGP). In more detail, assume that the problem has N fitness cases; then N attributes are added to each node in the individual's tree. In figure 2 N is set to 3, so three attributes A_1, A_2, A_3 are added to every node, to cache the SS of the corresponding subtree.

Figure 2 also describes the process of evaluating attribute values in AGP. Initially (Figure 2a), the attributes are set to zero. Assume that the fitness cases include three values 0, 0.5, and 1, then, in the second step, the attributes of the leaves of the individual are assigned with these values (Figure 2b, attributes at the nodes labeled with a constant are assigned with the value of that constant). Next, the attributes at the level above the leaves are assigned with values. At this point, the semantics of the leaves is passed upward to their parents, and the operator at those nodes are applied to calculate the values for the attributes (Figure 2c) at these nodes. This process is then continued until the attributes at the root node are assigned with values (Figure 2d). It is noted that when this process of value propagation completes, the fitness of the individual can be ob-

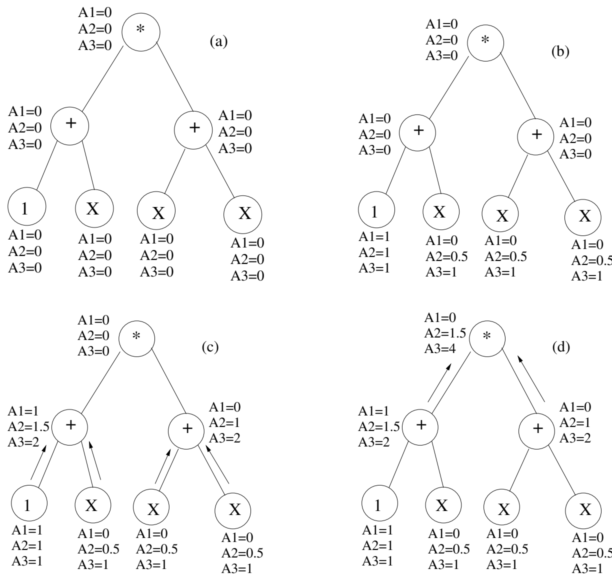


Fig. 2. An individual in AGP and the Process of Evaluating its Attributes

Table 1. Symbolic Regression Functions

Functions	Training Data
$F1 = x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F2 = x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F3 = x^5 + x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F5 = (x + 1)^3$	20 random points $\subseteq$ [-1,1]
$F6 = x^3 - x^2 - x - 1$	20 random points $\subseteq$ [-1,1]
$F7 = 0.3sin(2\pi x)$	20 random points $\subseteq$ [-1,1]
$F8 = cos(3x)$	20 random points $\subseteq$ [-1,1]

Table 2. Run and Evolutionary Parameter Values

Parameter	Value
Population size	500
Generations	50
Selection	Tournament
Tournament size	3
Crossover probability	0.9
Mutation probability	0.05
Initial Max depth	6
Max depth	15
Max depth of mutation tree	5
Non-terminals	+, -, *, / (protected version), sin, cos, exp, log (protected version)
Terminals	X, 1
Raw fitness	mean absolute error on all fitness cases
Trials per treatment	100 independent runs for each value

tained by comparing the semantics of the root node with the values of the target function on the corresponding fitness cases and the semantic distance between two trees can be calculated by summing the attributes of their root nodes. This helps to speed up the calculation of semantic distance between individuals in fitness sharing.

3 Experimental Settings

To investigate the impact of using these distance metrics in fitness sharing on GP performance, we used eight real-valued symbolic regression problems. The problems and training data are shown in Table 1. These functions were taken from previous work on using semantics based operators in GP [20,21].

The GP parameters used for our experiments are shown in Table 2. It should be noted that the raw fitness is the mean of absolute error on all fitness cases.

Therefore, the smaller values are better. For each problem and each parameter setting, 100 runs were performed.

We divided our experiments into two sets. The first is to compare the performance of fitness sharing using a semantic metric with a syntactic metric and with standard GP, and the second set aims to investigate the impact of some parameters (the number of the sampled individuals and the niche radius, σ) on the performance of GP using fitness sharing with semantic metric. Hereafter, the fitness sharing using the semantic metric is called *Semantic Sharing* and the fitness sharing using the syntactic metric is called *Syntactic Sharing*.

4 Results and Discussion

This section first presents the comparison on the performance of GP using semantic sharing with syntactic sharing and standard GP. After that the impact of some parameters on the performance of GP using semantic sharing is discussed.

4.1 On the Performance

We tested fitness sharing using semantic and syntactic distance on the above eight problems. For semantic sharing, we selected the niche radius, σ at 0.1. This is the value determined to achieve the best performance with semantic sharing (the following subsection will investigate the impact of σ on the performance of GP using semantic sharing). Three values for the number of the individuals that are sampled to calculate niche were tested. They are 5, 10 and 15. Semantic sharing with these values will be shorthanded as SS5, SS10, and SS15.

For syntactic sharing, we fixed the niche radius, σ at 10. This was indicated from experiments as the best value for GP performance. Similarly, three values for the number of sampled individuals are 5, 10 and 15 were tested. Syntactic sharing with these three values are referred to as SyS5, SyS10, and SyS15.

To measure the performance of GP with these approaches we use a classical performance metric: mean of the best fitness. Table 3 shows the best fitness found, averaged over all 100 runs of each GP system. We tested the statistical significance of the results in Table 3 using a Wilcoxon signed-rank test with a confidence level of 95%. In Table 3, if a run of semantic sharing or syntactic sharing is significantly better than Standard GP (GP), its result is printed bold face.

It can be seen from Table 3 that syntactic sharing barely improves the performance of GP. Sometimes, syntactic sharing is even worse than standard GP. This can be observed in some cases on Function F2, F5, F7 and F8. In fact, syntactic sharing only significantly improves GP performance on three occasions, namely on F1 (SyS10 and SyS15) and F4 (SyS15). These results are not entirely surprising as Ekart and Nemeth [15] also showed that fitness sharing based on structural distance provides very little advantage for GP performance.

On the contrary, semantic sharing always helps to improve the performance of GP. It can be seen from Table 3 that the mean best fitness found by semantic sharing is consistently smaller than the value found by standard GP. For the

Table 3. Mean best fitness of three methods. Note that the values are scaled by 10^2

Methods	F1	F2	F3	F4	F5	F6	F7	F8
GP	1.05	1.52	2.14	2.78	2.65	3.17	4.40	1.50
SyS5	0.85	1.39	1.86	2.56	3.00	2.90	4.44	1.73
SyS10	**0.78**	1.73	1.94	2.57	2.52	3.01	4.58	1.37
SyS15	**0.69**	1.37	1.78	**2.06**	2.58	2.97	4.33	1.56
SS5	**0.69**	**1.13**	1.66	**2.09**	**2.22**	**2.56**	**3.87**	**1.12**
SS10	**0.55**	**1.12**	1.70	**2.03**	2.32	**2.48**	**3.62**	**1.11**
SS15	**0.75**	**1.19**	1.71	**2.14**	**2.15**	**2.59**	**3.91**	**0.97**

three values of the numbers of the individuals that are sampled for calculating the niche, it can be seen that the performance of semantic sharing is consistent. The table also shows that the majority of the improvement achieved by semantic sharing over standard GP is statistically significant.

Table 4. Average time of a run (in seconds) for the two fitness sharing strategies

Methods	F1	F2	F3	F4	F5	F6	F7	F8
GP	4.36	5.09	4.85	5.32	5.30	6.44	7.47	5.85
SyS5	14.2	15.5	15.8	16.8	16.8	18.7	22.7	17.9
SyS10	23.1	26.4	27.4	28.8	27.6	32,8	36.6	28.8
SyS15	33.2	36.2	38.6	39.7	36.4	41.2	45.9	39.7
SS5	5.21	5.42	5.26	5.37	5.31	6.34	7.11	5.52
SS10	5.12	5.69	5.30	5.61	5.52	6.59	7.34	5.94
SS15	4.94	5.62	5.38	5.54	5.59	6.48	7.38	5.90

As has been previously mentioned, one of the weaknesses of fitness sharing is that it takes time to calculate the semantic or syntactic distance between individuals in the population. To estimate the extra time of these methods, we measured their running time. The average time of a run of these methods compared to standard GP is shown in Table 4.

It can be seen from Table 4 that it is very time-consuming to implement syntactic sharing. The average time of a run of syntactic sharing is much greater than the value of standard GP. When the number of individuals that are sampled increases, the average running time also increases. Conversely, semantic sharing runs almost at the same speed as standard GP. The average run time of semantic sharing is mostly equal to that of standard GP, and in some cases these values are even smaller. This represents the effectiveness of using attributes to store semantics in the calculation of semantic distance.

Table 5. Mean best fitness of semantic sharing with different values of the niche radius. Note that the values are scaled by 10^2

Methods	F1	F2	F3	F4	F5	F6	F7	F8	*Mean*
GP	1.05	1.52	2.14	2.78	2.65	3.17	4.40	1.50	*2.40*
SSR005	0.61	1.25	1.75	2.19	2.43	2.70	3.71	1.20	*1.98*
SSR0075	0.52	1.11	1.66	2.23	2.31	2.56	3.61	1.14	*1.89*
SSR01	0.55	1.12	1.70	2.03	2.32	2.48	3.62	1.11	*1.87*
SSR0125	0.53	1.34	1.79	2.16	2.25	2.70	3.92	1.14	*1.98*
SSR015	0.61	1.35	1.74	2.07	2.45	2.82	3.92	1.13	*2.01*

4.2 Parameters Analysis

This section analyses the impact of some parameters on the performance of GP using semantic sharing [1]. There are two parameters that potentially impact the performance of GP with semantic sharing: the threshold of dissimilarity (also the niche radius), σ and the size of sampled individuals P. To investigate the sensitivity of the niche radius on GP performance, we fixed the size of sampled individuals at 10 and tested 5 values of σ. The five values tested are: 0.05, 0.075, 0.1, 0.125, 0.15. Five configurations of semantic sharing with these values are referred to as SSRX with X=0.05, 0.075, 0.1, 0.125, 0.15.

To estimate the effect of changing σ, we recorded the best fitness of a run. These values were averaged over 100 runs and are shown in Table 5. For the purpose of comparison, the mean best fitness of standard GP is also shown in the top row of this table.

It can be seen from Table 5 that the values of the niche radius around 0.1 are good values overall. The performance of semantic sharing with values 0.075 and 0.1 are the most consistent. When this value is too small (0.05) or too great (0.125 and 0.15), the performance is worse.

We now examine the impact of the second parameter, the size of sampled individuals, on the performance of GP using semantic sharing. To do this, we fixed σ at 0.1 and 6 values of the size of sampled individuals were tested. The six values are 5, 10, 15, 20, 40, and 80. The corresponding configurations of semantic sharing with these six values are shorthanded as SSSX with X=5, 10, 15, 20, 40, and 80, respectively.

To discover the effect of changing this parameter, we again recorded the best fitness of a run. These values were averaged over 100 runs and are shown in Table 6. For the purpose of comparison, the mean best fitness of standard GP is also shown in the top row of this table.

Table 6 shows that the size of sampled individuals needs only relatively small values. It can be seen that the performance of semantic sharing is best with the values from 5 to 20 (1 to 4% of the total population size). If this value is

[1] Since the performance of syntactic sharing is not as good as the performance of semantic sharing, it is not investigated further in this paper.

Table 6. Mean best fitness of semantic sharing with different values of the size of sampled individuals. Note that the values are scaled by 10^2.

Methods	F1	F2	F3	F4	F5	F6	F7	F8	*Mean*
GP	1.05	1.52	2.14	2.78	2.65	3.17	4.40	1.50	*2.40*
SSS5	0.69	1.13	1.66	2.09	2.22	2.56	3.87	1.12	*1.92*
SSS01	0.55	1.12	1.70	2.03	2.32	2.48	3.62	1.11	*1.87*
SSS15	0.75	1.19	1.71	2.14	2.15	2.59	3.91	0.97	*1.93*
SSS20	0.58	1.23	1.65	1.99	2.41	2.69	3.72	1.17	*1.93*
SSS40	0.85	1.61	2.17	2.48	2.32	3.04	4.06	1.24	*2.22*
SSS80	1.14	1.52	2.33	2.56	2.53	2.96	4.72	1.35	*2.39*

too great (40 and 80) the performance is worse. This can be explained by the fact that promoting too much diversity (when increasing the value of the size of sampled individuals) can hinder the convergence of GP to the global optimal.

5 Conclusions and Future Work

In this paper, we investigate the efficiency of fitness sharing using semantic and syntactic distance metrics. We propose a novel way to implement fitness sharing using a semantic distance metric based on sampling semantics. We also modify fitness sharing to speed up its execution. We compare the performance of Genetic Programming using fitness sharing with semantic and syntactic distance on a class of real-value symbolic regression problems. The experimental results show on the tested problems that while fitness sharing with the syntactic metric hardly improves the performance of GP, fitness sharing with the semantic metric often significantly improves GP performance. At the same time, fitness sharing that implements the semantic distance metric runs much faster than with the syntactic metric. Further analysis shows the impact of the two main parameters on the performance of fitness sharing with the semantic distance metric.

There are a number of areas for future work which arise from this paper. First, we want to measure the change of semantic diversity and syntactic diversity of fitness sharing implemented in this paper during the course of evolution to understand its impact on GP performance. Second, we would like to combine promoting semantic diversity with controlling semantic locality [20] to see if it provides additional improvement in performance. Last but not least, we aim to investigate the impact of this method on dynamic problems where maintenance of population dispersion/diversity is critical for adaptation to a changing environment.

Acknowledgements. This work was funded by The Vietnam National Foundation for Science and Technology Development (NAFOSTED), under grant number 102.01-2011.08. The third author acknowledges the support of Science Foundation Ireland under grant number 08/IN.1/I1868. A. Agapitos is supported by Science Foundation Ireland Grant No. 08/SRC/FM1389.

References

1. Poli, R., Langdon, W., McPhee, N.: A Field Guide to Genetic Programming (2008), http://lulu.com
2. Koza, J.: Genetic Programming: On the Programming of Computers by Natural Selection. MIT Press, MA (1992)
3. Koza, J.: Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines 11(3-4), 251–284 (2010)
4. Gustafson, S., Burke, E.K., Kendall, G.: Sampling of Unique Structures and Behaviours in Genetic Programming. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) EuroGP 2004. LNCS, vol. 3003, pp. 279–288. Springer, Heidelberg (2004)
5. Burke, E.K., Gustafson, S., Kendall, G.: Diversity in genetic programming: An analysis of measures and correlation with fitness. IEEE Transactions on Evolutionary Computation 8(1), 47–62 (2004)
6. Looks, M.: On the behavioral diversity of random programs. In: GECCO 2007: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, July 7-11, vol. 2, pp. 1636–1642. ACM Press (2007)
7. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genetic Programming and Evolvable Machines 11(3-4), 339–363 (2010)
8. Gustafson, S.: An Analysis of Diversity in Genetic Programming. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, Nottingham, England (February 2004)
9. Beadle, L., Johnson, C.G.: Semantic analysis of program initialisation in genetic programming. Genetic Programming and Evolvable Machines 10(3), 307–337 (2009)
10. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers (2011)
11. Morrison, R.: Designing Evolutionary Algorithms for Dynamic Environments. Springer, Heidelberg (2004)
12. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor (1975)
13. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
14. Langdon, W.B.: Genetic Programming and Data Structures: Genetic Programming + Data Structure = Automatic Programming! Kluwer Academic, Boston (1998)
15. Ekárt, A., Németh, S.Z.: A Metric for Genetic Programs and Fitness Sharing. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 259–270. Springer, Heidelberg (2000)
16. McKay, B.: An investigation of fitness sharing in genetic programming. The Australian Journal of Intelligent Information Processing Systems 7(1/2), 43–51 (2001)
17. Sareni, B., Kraehenbuehl, L.: Fitness sharing and niching methods revisited. IEEE-EC 2(3), 97 (1998)
18. Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Semantic Aware Crossover for Genetic Programming: The Case for Real-Valued Function Regression. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 292–302. Springer, Heidelberg (2009)

19. Nguyen, Q.U., O'Neill, M., Nguyen, X.H., Mckay, B., Galván-López, E.: Semantic Similarity Based Crossover in GP: The Case for Real-Valued Function Regression. In: Collet, P., Monmarché, N., Legrand, P., Schoenauer, M., Lutton, E. (eds.) EA 2009. LNCS, vol. 5975, pp. 170–181. Springer, Heidelberg (2010)
20. Nguyen, Q.U., Nguyen, X.H., O'Neill, M., McKay, R.I., Galvan-Lopez, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines, 91–119 (2011)
21. Nguyen, Q.U., Nguyen, T.H., Nguyen, X.H., O'Neill, M.: Improving the Generalisation Ability of Genetic Programming with Semantic Similarity based Crossover. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 184–195. Springer, Heidelberg (2010)

Evolving Reusable Operation-Based Due-Date Assignment Models for Job Shop Scheduling with Genetic Programming

Su Nguyen[1], Mengjie Zhang[1], Mark Johnston[1], and Kay Chen Tan[2]

[1] Victoria University of Wellington, Wellington, New Zealand
[2] National University of Singapore, Singapore
{su.nguyen,mengjie.zhang}@ecs.vuw.ac.nz,
mark.johnston@msor.vuw.ac.nz,eletankc@nus.edu.sg

Abstract. Due-date assignment plays an important role in scheduling systems and strongly influences the delivery performance of job shops. Because of the stochastic and dynamic features of job shops, the development of general due-date assignment models (DDAMs) is complicated. In this study, two genetic programming (GP) methods are proposed to evolve DDAMs for job shop environments. The experimental results show that the evolved DDAMs can make more accurate estimates than other existing dynamic DDAMs with promising reusability. In addition, the evolved operation-based DDAMs show better performance than the evolved DDAMs employing aggregate information of jobs and machines.

Keywords: Genetic programming, Job shop, Due-date assignment.

1 Introduction

Job Shop Scheduling (JSS) has been one of the most popular topics in the scheduling literature due to its complexity and applicability in real world situations. A large number of studies on JSS have focused on sequencing decisions, which determine the order in which waiting jobs are processed on a set of machines (or resources) in a manufacturing system (shop). However, sequencing is only one of several steps in the scheduling process [1]. One of the other important activities in JSS is due-date assignment (DDA), sometimes referred to as estimation of job flowtimes (EJF). The objective of this activity is to determine the due-dates for arriving jobs by estimating the job flowtimes (the time from the arrival until the completion of the job), and therefore DDA strongly influences the delivery performance, i.e., the ability to meet promised delivery dates, of a job shop [4]. In addition, accurate flowtime estimates [18] are needed for better management of the shop floor activities, evaluation of the shop performance and leadtime comparison, etc.

Due-date assignment decisions are made whenever jobs (customer orders) are received from customers. Due-dates can be set exogenously or endogenously [4,17]. In the former case, due-dates are decided by some independent agency

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 121–133, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Table 1. Performance measures of DDAMs

Mean Absolute Percentage Error	$\mathtt{MAPE} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}\frac{\lvert e_j\rvert}{f_j}$
Mean Percentage Error	$\mathtt{MPE} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}\frac{e_j}{f_j}$
Mean Absolute Error	$\mathtt{MAE} = \frac{\sum_{j\in\mathbb{T}}\lvert e_j\rvert}{\lvert\mathbb{C}\rvert}$
Standard Deviation of Lateness	$\mathtt{STDL} = \sqrt{\frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}(e_j-\bar{e})^2}$
Percent Tardiness	$\%\mathtt{T} = 100 \times \frac{\lvert\mathbb{T}\rvert}{\lvert\mathbb{C}\rvert}$
Mean Flowtime	$\mathtt{MF} = \frac{\sum_{j\in\mathbb{C}} f_j}{\lvert\mathbb{C}\rvert}$

e.g., sellers or buyers. In this study, we only focus on the second case, in which the due-dates are internally set based on the characteristics of the jobs and the shops [17], to improve the delivery performance of job shops. Basically, the due-date of a new job is calculated as:

$$d_j = r_j + \hat{f}_j \quad (1)$$

where d_j is the due-date, r_j is the release time of job j (in our study, the release time is the arrival time of the job since the job is released to the shop immediately), and $\hat{f}_j$ is the estimated (predicted) flowtime of job j. The task of a due-date assignment model (DDAM) is to assign a value to $\hat{f}_j$. In the ideal case, we want the calculated due-date d_j to be equal to the completion time of the job C_j. The miss due-date performance is normally measured by the error (lateness) between the completion time and due-date $e_j = C_j - d_j = f_j - \hat{f}_j$, where f_j is the actual flowtime.

Some criteria to evaluate the performance of DDAMs [5,2] in the JSS literature are shown in Table 1. In this table, $\mathbb{C}$ is the set of jobs collected from the simulation runs to calculate the performance measures, e_j is the lateness of job j, $\bar{e}$ is the mean lateness and $\mathbb{T}$ is the set of tardy jobs ($C_j - d_j > 0$). MAPE and MAE measure the accuracy of the flowtime estimation. Smaller MAPEs or MAEs indicate that the DDAM can make better predictions. MPE measures the bias of the DDAM. If the DDAM results in a negative (positive) MPE, it means that the DDAM tends to overestimate (underestimate) the due-date. STDL measures the delivery reliability of the DDAM. Smaller STDL indicates that the estimated due-dates are more reliable. Another delivery performance measure is %T, which shows the percentage of jobs that fail to meet the due-date. Finally, MF measures the delivery speed of the scheduling system.

Many DDAMs have been proposed in the job shop literature. Traditional DDAMs focus on exploiting the shop and job information to make good flowtime estimates. Most of the early DDAMs are based on linear combinations of different terms (variables) and the coefficients of the models are then determined based on simulation results. Regression (linear and non-linear) has been used very often in order to help find the best coefficients for the models employed [16,7,23,22,18,19,9]. Since the early 1990s, artificial intelligence methods have also been applied to deal with due-date assignment problems such as

neural networks [15,20,14], decision trees [13], regression trees [21], and a regression based method with case-based tuning [19].

Even though experimental results with these DDAMs are promising, some limitations are still present. First, since a job can include several operations which represent the processing steps of that job at particular machines, the operation-based flowtime estimation (OFE) method [18] that utilises the detailed job, shop and route information for operations of jobs can help improve the quality of the prediction. However, this OFE method depends strongly on the determination of a large number of coefficients, which is not an easy task. Thus, there is a need to create a dynamic OFE method similar to Dynamic Total Work Content (DTWK), Dynamic Processing Plus Waiting (DPPW) [5], and ADRES [2] to overcome this problem by replacing the coefficients with more general aggregate terms (job characteristics and states of the system). Second, there are no studies on the reusability of the proposed DDAMs in the JSS literature, so it is questionable whether the proposed models can be applied when there are changes in the shop without major revisions. Finally, various relevant factors need to be considered in order to make a good estimation of flowtime, which makes the design of a new DDAM a time-consuming and complicated task.

Genetic Programming (GP) [10] is an evolutionary computation method which has been applied to evolve (train) programs that are able to solve difficult computational problems. We see that GP is also a good candidate approach to helping overcome the three limitations discussed above because (1) the DDAMs can be easily represented by GP, (2) DDAMs can be automatically evolved/trained on different shop environments to provide the generality for the evolved DDAMs, and (3) the DDAMs evolved by GP can be partially interpreted.

This paper aims to develop a new approach to the use of GP for evolving due-date assignment models (DDAMs) for job shop environments. We expect the evolved DDAMs to outperform the existing models in terms of mean absolute percentage error and to be reusable for new (unseen) simulation scenarios. Following are the objectives for this study:

1. Develop two GP methods to automatically evolve reusable Aggregate Due-date Assignment Models (ADDAMs) and Operation-based Due-date Assignment Models (ODDAMs) for the job shop environment.
2. Compare the evolved DDAMs with existing dynamic DDAMs.
3. Compare the performance of the two proposed GP methods.

In the next section, we provide details about the two proposed GP methods. The experimental setting is presented in Section 3. The experimental results, the comparisons, and analysis of evolved DDMAs are presented in Section 4. Section 5 gives some conclusions from this research and directions for future studies.

2 GP for Evolving DDAMs

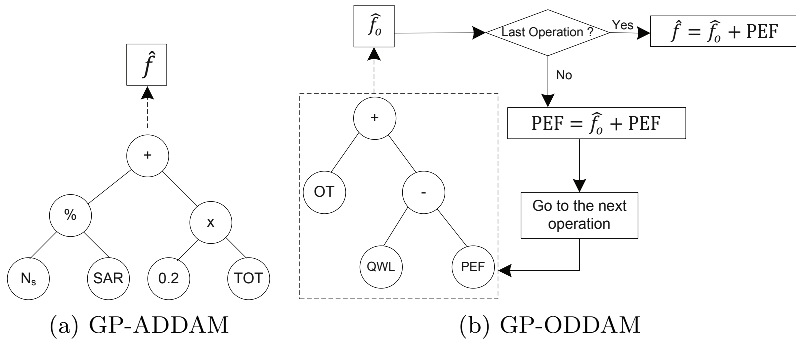
2.1 Representation

The purpose of the proposed GP-ADDAM and GP-ODDAM is to evolve dynamic ADDAMs and ODDAMs that estimate job flowtimes (i.e. due-dates by

Table 2. Terminal sets for GP-ADDAM and GP-ODDAM (ψ is the new job, ϕ is the considered operation in GP-ODDAM, and δ is the machine that process will ϕ)

GP-ADDAM		GP-ODDAM	
N		Number of jobs in the shop	
SAR		Sampled arrival rate	
#		Random number from 0 to 1	
TAPR	total average processing time of job in queues of machines that ψ will visit	APR	average processing times of jobs in the queue of the machine that processes ϕ
TOT	total processing time of ψ	OT	processing time of ϕ
TLOT	average LOT for all machines that ψ will visit	LOT	time for δ to finish the leftover job
AOTR	average OTR for all queues of machines that ψ will visit	OTR	percentage of jobs in queues of δ that require less processing time less than OT
ASOTR	average SOTR for all queues of machines that ψ will visit	SOTR	percentage of sampled jobs processed at δ that require less processing time less than OT
TQWL	total QWL for all machines that ψ will visit	QWL	total processing time of jobs in the queue of δ
TSAPR	total SAPR for all machines that ψ will visit	SAPR	sampled average processing time of jobs processed at δ
TRWL	total RWL for all machines that ψ will visit	RWL	total processing time of jobs that need to be processed at δ
SL	sampled average error e_j from previous jobs	PEF	partial estimated flowtime

using equation (1)) by employing information from jobs and the shop similar to DTWK and DPPW. In this case, we use tree-based GP [10] to create mathematical combinations of these pieces of information in each GP individual. For this reason, the function set will include standard mathematical operators $+,-,\times$, and protected division %, along with a conditional function If to allow GP to evolve sophisticated DDAMs. Function If includes three arguments and if the value from the first argument is greater than or equal to zero, If will return the value from the second argument; otherwise If will return the value from the third argument. Since ADDAMs and ODDAMs need different types of information, GP-ADDAM and GP-ODDAM will use different terminal sets as shown in Table 2. In this table, the first three terminals are the same for the two proposed GP methods. The next eight terminals are variables that characterise the state of operations/machines for GP-ODDAM and their aggregate counterparts for GP-ADDAM. The last terminal of each method provides some extra information to estimate the flowtime. SOTR and SAPR are calculated based on the sample of the last 20 jobs processed at machine δ. SAR, on the other hand, is calculated based on the arrivals of the last 20 jobs.



(a) GP-ADDAM (b) GP-ODDAM

Fig. 1. DDAM evaluation scheme

2.2 Evaluation

An example of how an individual in GP-ADDAM is evaluated is shown in Fig. 1(a). In this method, a GP individual represents a mathematical function and the output of this function is the estimated flowtime $\hat{f}$ of the new job. The information used in this function is extracted from the new job and machines in the shop.

The GP individual in GP-ODDAM aims at estimating the flowtime of each operation of the new job. Therefore, instead of using the function obtained from the GP individual to estimate job flowtime $\hat{f}$, the output of this function is used to estimate the operation flowtime $\hat{f}_o$ of each operation of the new job, starting from the first operation. When $\hat{f}_o$ is obtained, a condition is checked to see whether the operation being considered is the last operation. If it is not the last operation of the new job, $\hat{f}_o$ will be used to update the partial estimated flowtime (PEF), which will also be used as a terminal in the GP individual. Then, the GP individual is applied to estimate the flowtime for the next operation. In the case that the flowtime of the last operation has been estimated, $\hat{f}_o$ will be added to the current PEF to obtain the estimated flowtime $\hat{f}$. The evaluation scheme for GP-ODDAM is shown in Fig. 1(b). The use of PEF (initially zero for the first operation) in the terminal set of GP-ODDAM also provides DDAMs a chance to predict the changes of the system, given that the partial estimated flowtime is well predicted.

2.3 Genetic Operators

Traditional genetic operators are employed by the proposed GP methods. For crossover, the GP system uses the subtree crossover [10], which creates new individuals for the next generation by randomly recombining subtrees from two selected parents. Meanwhile, mutation is performed by subtree mutation [10], which randomly selects a node of a chosen individual and replaces the subtree rooted at that node by a newly randomly-generated subtree. For reproduction, an individual is selected from the population by the selection mechanism (e.g. tournament selection) and copied to the population of the next generation.

2.4 Fitness Function

As discussed in Section 1, the performance of a DDAM can be measured in many different ways, which indicate the delivery accuracy and delivery reliability. In this study, we will use `MAPE` to measure the quality of evolved DDAMs because it is a good indicator for both delivery accuracy and delivery reliability. A discrete-event simulation model of a job shop is built for evaluation of evolved DDAMs. In this model, the arrivals of jobs, the processing times and route information of jobs will follow some particular probability distributions. Upon the arrival of a job j, the DDAM will be applied to estimate the flowtime $\hat{f}_j$ of that job. The error e_j of this estimation is recorded when job j leaves the system and the errors of all recorded jobs will be used to calculate `MAPE` as shown in Table 1. Since our objective is to evolve reusable DDAMs, the quality of the evolved DDAMs will be measured based on their performance on a number of simulation scenarios $\mathbb{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K\}$ which represent different shop characteristics. For a simulation scenario $\mathcal{S}_k$, the quality of a DDAM p_i is indicated by $\texttt{MAPE}^{\mathcal{S}_k}_{p_i}$. The fitness value of p_i is calculated as followed:

$$fitness(p_i) = \frac{1}{K}\sum_{k=1}^{K} \texttt{MAPE}^{\mathcal{S}_k}_{p_i} \qquad (2)$$

With this design, smaller $fitness(p_i)$ indicates that the evolved DDAM p_i can make more accurate estimations of jobs across different scenarios.

2.5 Evolution of DDAMs

Algorithm 1 shows how GP can be used to evolve DDAMs in both GP-ADDAM and GP-ODDAM. A variety of simulation scenarios will be employed in this algorithm to provide the evolved (trained) DDAMs better generality but it should be noted that a large number of scenarios also increase the computation time of the GP systems. The evolution process will be terminated when the maximum generation is reached and the algorithm will return the best found DDAM p^*.

3 Experimental Setting

3.1 Job Shop Simulation Environment

In this study, we use a symmetrical (balanced) job shop simulation model in which each operation of a job has equal probability to be processed at any machine in the shop (a job visits each machine at most once). Therefore, machines in the shop have the same level of congestion in long simulation runs. This model has also been used very often in the JSS literature [3,5,18,11,8]. Based on the factors discussed above, the scenarios for training and testing of DDAMs are shown in Table 3.

In these scenarios, the mean processing time of operations is fixed to 1 and the arrival of jobs will follow a Poisson process with the arrival rate adjusted based on the utilisation level. For the distribution of number of operations, the

Algorithm 1. General GP algorithm for GP-ADDAM and GP-ODDAM

```
load simulation scenarios 𝕊 ← {S_1, S_2, ..., S_K};
randomly initialise the population ℙ ← {p_1, p_2, ..., p_popsize} ;
p* ← null and fitness(p*) = +∞;
generation ← 0;
while generation ≤ maxGeneration do
    foreach p_i ∈ ℙ do
        foreach S_k ∈ 𝕊 do
            calculate MAPE_{p_i}^{S_k}
        end
        evaluate fitness(p_i) by using equation (2) ;
        if fitness(p_i) < fitness(p*) then
            p* ← p_i;
            fitness(p*) ← fitness(p_i);
        end
    end
    ℙ ← apply reproduction, crossover, mutation to ℙ ;
    generation ← generation + 1
end
return p*;
```

Table 3. Training and testing scenarios

Factor	Training	Testing
Number of machines	4,6	4,5,6,10,20
Utilisation	70%,80%,90%	60%,70%,80%,90%,95%
Distribution of processing time	Exponential	Exponential, Erlang-2, Uniform
Distribution of number of operations	missing	missing/full

missing setting is used to indicate that the number of operations will follow a discrete uniform distribution from 1 to the number of machines. Meanwhile, the *full* setting indicates the case that each job will have its number of operations equal to the number of machines in the shop. In each replication of a simulation scenario, we start with an empty shop and the interval from the beginning of the simulation until the arrival of the 1000th job is considered as the warm-up time and the information of the next completed 5000 jobs (set $\mathbb{C}$ used in Table 1) is collected to evaluate the performance of DDAMs.

In the training stage, since the simulation is very time-consuming, we only perform one replication for each scenario. There are $(2\times3\times1\times1) = 6$ simulation scenarios used to evaluate the performance of the evolved DDAMs. For testing, the best DDAM p^* obtained from a run of GP is applied to $(5 \times 5 \times 3 \times 2) = 150$ simulation scenarios and 30 simulation replications are performed for each scenario; therefore, we need $150 \times 30 = 4500$ simulation replications to test the performance of p^*. The use of a large number of scenarios and replications in the testing stage will help us confirm the quality and reusability of the evolved

Table 4. Parameters of the proposed GP systems

Population Size	2000	Crossover rate	80%	Mutation rate	15%
Reproduction rate	5%	Generations	50	Max-depth	17

DDAMs. For the shop floor level, First-In-First-Out (FIFO) is used as the dispatching rule to sequence jobs in queues of machines. By using FIFO, the earliest job that joins the queue of the machine will be processed first. We examine FIFO in this study because it is one of the the most popular dispatching rules in the scheduling literature.

3.2 GP Parameters

The GP system for learning DDAMs is developed based on the ECJ20 library [12]. The parameter settings of the GP system used in the rest of this study are shown in Table 4. The initial GP population is created using the ramped-half-and-half method [10]. Tournament selection of size 7 is used to select an individual for the genetic operators.

4 Results

4.1 Comparison of DDAMs

For each GP method, 30 independent runs are performed and the best ADDAM $p^{\mathtt{ADDAM}}$ and ODDAM $p^{\mathtt{ODDAM}}$ are recorded and compared with existing dynamic DDAMs (DTWK, DPPW, and ADRES). Table 5 and Table 6 show $\mathtt{MAPE}^{\mathcal{S}_k}_{p^{\mathtt{ADDAM}}}$, $\mathtt{MAPE}^{\mathcal{S}_k}_{p^{\mathtt{ODDAM}}}$ and the *t-test* results between the evolved DDAMs and other DDAMs in 150 testing scenarios. In these tables, the a, b, and c are the indices to represent DTWK, DPPW, and ADRES, respectively. The superscript of a result in these tables shows the DDAMs that are **not** significantly different (by using *t-tests*) from the evolved DDAMs. Meanwhile, the subscript shows the DDAMs that are significantly better than the evolved DDAM, respectively. If an index is neither shown in the superscript nor subscript, it means that the evolved DDAM is significantly better than all other DDAMs. The tests are considered significant when the obtained *p-value* is less than 0.05. It is easy to see that the evolved DDAMs dominate other DDAMs in most scenarios. In a few specific cases, DPPW and ADRES are competitive with the evolved DDAMs and DPPW can also beat the evolved DDAMs in some scenarios with high utilisation level (95%), *full* setting and large numbers of machines. This may be because DPPW is based on the steady state performance of the system and can perform better in shops with less diverse jobs when the *full* setting is used and with high levels of utilisation. It is also noted that $\mathtt{MAPE}^{\mathcal{S}_k}_{p^*}$ is better when the utilisation increases, which is similar to that observed in [18]. When the number of machines increases, it is also interesting to see that the performance of the evolved DDAMs deteriorates if the *missing* setting is used, but the performance of these DDAMs improves if the *full* setting is used.

Table 5. Comparing the evolved ADDAM with existing DDAMs

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Expon.	4	0.1645	0.1669^{bc}	0.1550	0.1292	0.1055	0.2313	0.2279	0.2115	0.1700	0.1305
	5	0.1788	0.1782	0.1638	0.1336^{bc}	0.1067	0.2358	0.2296	0.2099	0.1675	0.1304^{b}
	6	0.1877	0.1877	0.1754	0.1444	0.1183	0.2358	0.2295	0.2088	0.1660	0.1303^{b}
	10	0.2057	0.2030	0.1881	0.1528^{c}	0.1269	0.2170	0.2095	0.1905	0.1516^{b}	0.1218^{b}
	20	0.2058	0.2032	0.1884	0.1549	0.1348	0.1758	0.1719	0.1585$_{b}$	0.1303$_{b}$	0.1110$_{b}$
Erlang-2	4	0.1527	0.1567	0.1500	0.1265	0.0996	0.2115	0.2152	0.2057	0.1732	0.1362
	5	0.1660	0.1681	0.1619	0.1368	0.1095	0.2118	0.2140	0.2042	0.1700	0.1371^{b}
	6	0.1724	0.1765	0.1688	0.1429	0.1169	0.2094	0.2114	0.2017	0.1701	0.1384^{b}
	10	0.1883	0.1904	0.1805	0.1499	0.1208	0.1902	0.1925	0.1833	0.1537^{b}	0.1251^{b}
	20	0.1843	0.1872	0.1799	0.1541	0.1335	0.1524	0.1563	0.1518	0.1317$_{b}$	0.1125$_{b}$
Uniform	4	0.1148	0.1320	0.1421	0.1404	0.1235	0.1525	0.1747	0.1875	0.1875	0.1694
	5	0.1228	0.1416	0.1515	0.1480	0.1300	0.1498	0.1700	0.1831	0.1857	0.1691
	6	0.1292	0.1476	0.1576	0.1545	0.1341	0.1427	0.1626	0.1777	0.1824	0.1656
	10	0.1333	0.1517	0.1653	0.1646	0.1451	0.1221	0.1402	0.1593	0.1662	0.1515
	20	0.1236	0.1428	0.1594	0.1623	0.1443	0.0928	0.1109	0.1287	0.1360	0.1269^{b}

Table 6. Comparing the evolved ODDAM with existing DDAMs

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Expon.	4	0.1418	0.1473^{bc}	0.1404	0.1208	0.1010	0.2105	0.2114	0.1995	0.1640	0.1278
	5	0.1570	0.1604	0.1514	0.1271^{bc}	0.1034	0.2175	0.2145	0.1993	0.1620	0.1278
	6	0.1672	0.1705	0.1627	0.1378	0.1149	0.2192	0.2161	0.1992	0.1607	0.1277^{b}
	10	0.1880	0.1886	0.1781	0.1481	0.1247	0.2053	0.1995	0.1832	0.1481^{b}	0.1203^{b}
	20	0.1935	0.1941	0.1827	0.1528	0.1340	0.1695	0.1670	0.1558^{b}	0.1294$_{b}$	0.1108$_{b}$
Erlang-2	4	0.1315	0.1383	0.1361	0.1178	0.0946	0.1914	0.1980	0.1929	0.1657	0.1323
	5	0.1458	0.1508	0.1484	0.1293	0.1054	0.1947	0.1992	0.1925	0.1634	0.1331
	6	0.1537	0.1607	0.1564	0.1361	0.1130	0.1943	0.1982	0.1906	0.1630	0.1343^{b}
	10	0.1730	0.1773	0.1705	0.1447	0.1186	0.1809	0.1831	0.1755	0.1493	0.1227^{b}
	20	0.1745	0.1793	0.1744	0.1518	0.1325	0.1483	0.1532	0.1498	0.1310^{b}	0.1123$_{b}$
Uniform	4	0.0905	0.1109	0.1261	0.1306	0.1166	0.1278	0.1527	0.1715	0.1771	0.1620
	5	0.1002	0.1220	0.1374	0.1395	0.1245	0.1316	0.1550	0.1725	0.1760	0.1606
	6	0.1078	0.1293	0.1448	0.1464	0.1292	0.1306	0.1530	0.1703	0.1734	0.1573
	10	0.1177	0.1392	0.1559	0.1569	0.1400	0.1199	0.1402	0.1571	0.1613	0.1474
	20	0.1159	0.1375	0.1554	0.1596	0.1430	0.0957	0.1133	0.1295	0.1370	0.1280^{b}

Table 7 shows the detailed results obtained by evolved and existing DDAMs for a particular scenario. Mean and standard deviation of each performance measure are shown in this table to provide a general evaluation of each DDAM. The MAPE, MAE and STDL of the evolved DDAMs are better (smaller) than those obtained by the existing DDAM. This indicates that the evolved DDAMs provide better delivery accuracy and delivery reliability than existing DDAMs. It is also interesting that the MPEs of evolved DDAMs are positive while those of the existing DDAMs are negative. This means that the existing DDAMs tend to overestimate the job flowtimes while the evolved DDAMs tend to underestimate the job flowtimes but the estimations made by evolved DDAMs are better because their MPEs are closer to zero compared to those of existing DDAMs. Because the existing DDAMs overestimate flowtimes, %T of those DDAMs are smaller than those of evolved DDAMs. However, with the current emphasis on the just-in-time (JIT) [6] production concept where both earliness and tardiness

Table 7. Performance of DDAMs (utilisation = 80%, full jobs, 4 machines, processing times follow Exponential distribution)

DDAM	MAPE	MAE	MPE	STDL	%T	MF
$p^{\mathtt{ADDAM}}$	0.211 ± 0.006	4.031 ± 0.313	0.005 ± 0.003	5.377 ± 0.452	53.296 ± 0.735	20.254 ± 2.137
$p^{\mathtt{ODDAM}}$	0.199 ± 0.005	3.924 ± 0.312	0.048 ± 0.004	5.154 ± 0.448	59.045 ± 0.725	20.254 ± 2.137
DTWK	0.411 ± 0.008	8.417 ± 1.024	−0.006 ± 0.018	12.232 ± 1.826	57.093 ± 1.392	20.254 ± 2.137
DPPW	0.259 ± 0.007	4.750 ± 0.380	−0.033 ± 0.013	6.251 ± 0.526	50.074 ± 1.987	20.254 ± 2.137
ADRES	0.455 ± 0.026	6.710 ± 0.360	−0.363 ± 0.031	7.343 ± 0.580	26.175 ± 1.456	20.254 ± 2.137

are undesireable and meeting the target job due date would be of significance for the practice of JIT philosophy, smaller MAPE and STDL would be more attractive than smaller %T.

4.2 GP-ADDAM vs. GP-ODDAM

Table 8 shows the *p-value*s of *t-tests* of the average $\mathtt{MAPE}_{p^*}^{\mathcal{S}_k}$ (from 30 simulation replications) in the testing scenarios between GP-ADDAM and GP-ODDAM. In this table, the highlighted values indicate that the GP-ADDAM is not significantly different from GP-ODDAM and other values show that GP-ODDAM is significantly better than GP-ADDAM (with *p-value* greater than 0.05). The results show that GP-ODDAM is significantly better than GP-ADDAM in most simulation scenarios, especially in the case where the *missing* setting for the distribution of number of operations is used. In the case that the *full* setting for the distribution of number of operations is used, GP-ODDAM is also significantly better than GP-ADDAM in most cases except for the scenarios with a high level of utilisation (95%) and large numbers of machines. These results suggest that the aggregate information of jobs used in ADDAMs is quite sufficient for estimating the flowtime of jobs when the shop is at a high congestion level and has a large number of machines. Also, ODDAMs may have difficulty in estimating the operation flowtimes of later operations of jobs with a large number of operations. This observation, and that observed in section 4.1 for DPPW, indicates the importance of aggregate information for flowtime estimation in the cases of high utilisation levels, less diverse jobs and large numbers of machines. It suggests that systematic incorporation of information between ADDAM and ODDAM could enhance the accuracy of the flowtime estimation.

4.3 Typical Examples of Evolved DDAMs

In this section, we further examine the evolved DDAMs to explore useful patterns for the development of more effective DDAMs. The best evolved ADDAM and ODDAM in a set of evolved DDAMs obtained from independent GP runs are shown in Fig. 2. Both evolved DDAMs include the total processing times of jobs in queues and the processing time of the new job (QWL + OT for ODDAM and TQWL + TOT for ADDAM). This term is actually a good estimate of flowtime for a job with a small number of operations (for ADDAM) or for the first operation of a new job (for ODDAM). Other common terms used in these two rules are

Table 8. GP-ADDAM vs. GP-ODDAM (*p-value*s from *t-tests*)

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0019	0.0293
	5	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0002	0.0054	0.0649
	6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0020	0.0133	0.1193
	10	0.0000	0.0000	0.0000	0.0000	0.0103	0.0082	0.0372	0.0483	0.0938	0.6603
	20	0.0015	0.0020	0.0027	0.0774	0.3551	0.5121	0.5499	0.3576	0.4581	0.6168
Erlang-2	4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0006
	5	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0003	0.0027
	6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0007	0.0035
	10	0.0000	0.0000	0.0000	0.0000	0.0016	0.0003	0.0049	0.0167	0.0150	0.0203
	20	0.0046	0.0077	0.0062	0.0137	0.5195	0.3497	0.4543	0.4203	0.1516	0.4618
Uniform	4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	5	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0003	0.0000	0.0000	0.0000	0.0000
	10	0.0000	0.0000	0.0000	0.0000	0.0003	0.9032	0.0120	0.0000	0.0004	0.0025
	20	0.1356	0.0114	0.0169	0.0471	0.1304	0.1054	0.6348	0.4269	0.5364	0.2944

```
(If(−(−N(+TQWL(−TQWL(/(+(+TQWL(+TOT TLOT))(+(/TQWL SAR)(−TRWL(∗TAPR SAR))))(/TQWL SAR)))))(+TLOT(∗TQWL TSAPR)))
   (+TQWL(+TOT TLOT))
   (If(−(−TQWL(/(+(+TOT TQWL)(+(/0.84095263(+(−TRWL(∗TAPR SAR))(+TOT TLOT)))(−TRWL(∗TAPR SAR))))TAPR))
   (−N(+TQWL(−TQWL(/(+(+TOT TQWL)(+(/0.84095263(+(−TRWL(∗TAPR SAR))(+TOT TLOT)))(−TRWL(∗TAPR SAR))))TAPR)))))
    (+TQWL TOT)   (+(−(−(−TQWL ASOTR)(/TLOT(+TLOT TAPR)))ASOTR)(+TOT TLOT))))
```

(a) $p^{\texttt{ADDAM}}$

```
(If(+(−(+(−(+(−(/LOT LOT)(+QWL LOT))SAPR)PEF)SAPR)PEF)(+OT(+QWL LOT)))
   (If(+(−(−LOT PEF)PEF)LOT) (+OT(+QWL LOT)) (+QWL OT))
   (−  (If(+(−(/(−(+(−(−RWL QWL)(+QWL LOT))SAPR)(+QWL LOT))LOT)(+QWL LOT))SAPR)
       (+OT(+(+OTR QWL)LOT))  (−(+(+LOT QWL)OT) (/LOT LOT)))  OTR))
```

(b) $p^{\texttt{ODDAM}}$

Fig. 2. Best evolved DDAMs

the leftover time of jobs in process (`LOT` and `TLOT`) and percentage of jobs in queues that require less processing time than the processing time of the new job (`OTR` and `TOTR`). The main difference between these two evolved DDAMs is the use of conditional terms to decide which extra terms should be included in the estimation. For ODDAM, `PEF`, `QWL` and `LOT` are used in the conditional term of function `If`. This DDAM shows that `PEF` is an important term to provide better flowtime estimations. It is noted that the conditional terms in ADDAM are more complex than that in ODDAM. The detailed analysis of these conditions is beyond the scope of this study but would be very useful for future research.

5 Conclusions

In this paper, two proposed GP methods are developed for evolving due-date assignment models. The experimental results show that the evolved DDAMs can outperform the existing dynamic DDAMs with `MAPE` as the performance measure. Comparisons using other performance measures also confirm the effectiveness of the evolved DDAMs. From the performance of the evolved DDAMs on a number of simulation scenarios, it can also be concluded that the evolved DDAMs

have good reusability since they are able to make good job flowtime estimates for unseen scenarios with different processing time distributions, utilisation, job settings and numbers of machines. When comparing the two proposed GP methods, it has been shown that GP-ODDAM is significantly better than GP-ADDAM in most testing scenarios. Typical examples of the evolved DDAMs show that these DDAMs are partially understandable. In future work, a detailed analysis of these evolved rules will be performed to show how they solve DDA problems. In addition, we would like to investigate the use of the proposed GP methods for automatic design of DDAMs for job shops employing other dispatching rules.

References

1. Ahmed, I., Fisher, W.W.: Due date assignment, job order release, and sequencing interaction in job shop scheduling. Decision Sciences 23(3), 633–647 (1992)
2. Baykasoglu, A., Gocken, M., Unutmaz, Z.D.: New approaches to due date assignment in job shops. European Journal of Operational Research 187, 31–45 (2008)
3. Chang, F.-C.R.: A study of due-date assignment rules with constrained tightness in a dynamic job shop. Computers & Industrial Engineering 31, 205–208 (1996)
4. Cheng, T.C.E., Gupta, M.C.: Survey of scheduling research involving due date determination decisions. European Journal of Operational Research 38(2), 156–166 (1989)
5. Cheng, T.C.E., Jiang, J.: Job shop scheduling for missed due-date performance. Computers & Industrial Engineering 34, 297–307 (1998)
6. Cheng, T.C.E., Podolsky, S.: Just-in-Time Manufacturing: an Introduction. Chapman and Hall, London (1993)
7. Fry, T.D., Philipoom, P.R., Markland, R.E.: Due date assignment in a multistage job shop. IIE Transactions 21(2), 153–161 (1989)
8. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In: GECCO 2010: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 257–264. ACM, New York (2010)
9. Joseph, O.A., Sridharan, R.: Analysis of dynamic due-date assignment models in a flexible manufacturing system. Journal of Manufacturing Systems 30(1), 28–40 (2011)
10. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
11. Land, M.J.: Workload Control in Job Shops, Grasping the Tap. Ph.D. thesis, University of Groningen, The Netherlands (2004)
12. Luke, S.: Essentials of Metaheuristics. Lulu (2009)
13. Ozturk, A., Kayaligil, S., Ozdemirel, N.E.: Manufacturing lead time estimation using data mining. European Journal of Operational Research 173(2), 683–700 (2006)
14. Patil, R.J.: Using ensemble and metaheuristics learning principles with artificial neural networks to improve due date prediction performance. International Journal of Production Research 46(21), 6009–6027 (2008)
15. Philipoom, P.R., Rees, L.P., Wiegmann, L.: Using neural networks to determine internally-set due-date assignments for shop scheduling. Decision Sciences 25(5-6), 825–851 (1994)

16. Ragatz, G.L., Mabert, V.A.: A simulation analysis of due date assignment rules. Journal of Operations Management 5(1), 27–39 (1984)
17. Ramasesh, R.: Dynamic job shop scheduling: A survey of simulation research. Omega 18(1), 43–57 (1990)
18. Sabuncuoglu, I., Comlekci, A.: Operation-based flowtime estimation in a dynamic job shop. Omega 30(6), 423–442 (2002)
19. Sha, D.Y., Storch, R.L., Liu, C.H.: Development of a regression-based method with case-based tuning to solve the due date assignment problem. International Journal of Production Research 45(1), 65–82 (2007)
20. Sha, D.Y., Hsu, S.Y.: Due-date assignment in wafer fabrication using artificial neural networks. The International Journal of Advanced Manufacturing Technology 23, 768–775 (2004)
21. Sha, D.Y., Liu, C.-H.: Using data mining for due date assignment in a dynamic job shop environment. The International Journal of Advanced Manufacturing Technology 25, 1164–1174 (2005)
22. Veral, E.A.: Computer simulation of due-date setting in multi-machine job shops. Computers & Industrial Engineering 41, 77–94 (2001)
23. Vig, M.M., Dooley, K.J.: Mixing static and dynamic flowtime estimates for due-date assignment. Journal of Operations Management 11(1), 67–79 (1993)

Evolving Interpolating Models of Net Ecosystem CO_2 Exchange Using Grammatical Evolution

Miguel Nicolau[1], Matthew Saunders[2], Michael O'Neill[1], Bruce Osborne[2], and Anthony Brabazon[1]

[1] Natural Computing Research & Applications Group
University College Dublin, Dublin, Ireland
{Miguel.Nicolau,M.ONeill,Anthony.Brabazon}@ucd.ie
[2] UCD School of Biology and Evironmental Science
University College Dublin, Dublin, Ireland
{Matthew.Saunders,Bruce.Osborne}@ucd.ie

Abstract. Accurate measurements of Net Ecosystem Exchange of CO_2 between atmosphere and biosphere are required in order to estimate annual carbon budgets. These are typically obtained with Eddy Covariance techniques. Unfortunately, these techniques are often both noisy and incomplete, due to data loss through equipment failure and routine maintenance, and require gap-filling techniques in order to provide accurate annual budgets. In this study, a grammar-based version of Genetic Programming is employed to generate interpolating models for flux data. The evolved models are robust, and their symbolic nature provides further understanding of the environmental variables involved.

Keywords: Grammatical evolution, Real-world applications, Symbolic regression.

1 Introduction

Eddy Covariance (EC) techniques are utilised globally to measure Net Ecosystem Exchange (NEE), defined as the net flux of Carbon Dioxide (CO_2) between the atmosphere and the biosphere [9]. NEE represents the balance between photosynthetic carbon uptake and respiratory carbon losses, and is typically measured over 30 minute intervals, which are then summed to give an annual carbon budget. Both short-term information and annual sums are of particular interest to scientists, land managers and policy makers. They allow for a comparison of ecosystem carbon budgets across various land use classes, provide a better understanding of the physiological driving processes, and facilitate an assessment of both inter and intra-annual climatic variability [4].

In order to derive the most accurate annual carbon budget, a complete data set is required; however, average data capture using the Eddy Covariance technique is often as low as 65% [4], due to data loss through equipment failure and routine maintenance. Furthermore, a diurnal bias exists in EC data rejection, due to the limitations of the EC technique at night, when low turbulence conditions occur [1,6,10].

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 134–145, 2012.
© Springer-Verlag Berlin Heidelberg 2012

To augment such fragmented data sets, gap-filling procedures are required to provide a more robust annual dataset [4]. Several gap-filling methodologies are currently employed by the global EC flux community, including linear interpolation, look-up tables, non-linear (semi-empirical) models, artificial neural networks, and multiple imputation techniques [18,9,4,1]. The utilisation of a particular gap-filling methodology is influenced by the experimental site-specific conditions, data availability and the particular end use of the EC data [4], however a particular effort has recently been made within the EC flux community to standardise gap-filling methodologies in order to allow the inter-comparison of different ecosystems, bio-climatic zones and long-term data sets [9]. There is however, a real need to continuously evaluate the accuracy of gap-filling models, which can be difficult to constrain, due to the multiple factors that influence EC measurements. For example, the presence of hysteresis loops in measured daytime NEE data can reduce the ability of semi-empirical light response functions to accurately model daytime NEE [20].

In the work presented here, a grammar-based Genetic Programming system was used to generate interpolating models for NEE data. The results obtained are comparable to the best in the literature [18], and the evolved symbolic models are fine-tunable, and also provide an insight into the effect of different environmental variables. These results highlight once again the real world applicability of evolutionary computation, and genetic programming in general.

The next section introduces the evolutionary algorithm used. Section 3 details the experimental setup, and the results obtained are analysed in Section 4. Finally, Section 5 draws some conclusions and future work directions.

2 Evolutionary Approach

Symbolic Regression is arguably one of the most successful applications of Genetic Programming [12] (GP). The tree structure of GP individuals lends itself to good functional representation and manipulation of sub-expressions, providing solutions that are often very precise, analysable, hand-tunable, and potentially provable.

For the purpose of evolving an EC flux gap-filling model, Grammatical Evolution (GE) [17,22] was used. GE is a grammar-based form of GP [14], which specifies the syntax of solutions in a grammar; this grammar is used to map genotypically evolved strings to syntactically correct phenotypic solutions.

GE performs on par with GP for symbolic regression purposes [17], while its grammar allows for extra control of the syntax of evolved programs, both in terms of biases [16,7] and data-structures used. This allows GE to be applied to a variety of domains, such as Financial Modelling [3], horse gait optimisation [15], wall shear stress analysis in grafted arteries [2], and optimisation of controllers for video-games [19], to name a few.

```
<expr>      ::= + <expr> <expr>
              | * <expr> <expr>
              | x
              | <digit>.<digit>
<digit>     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fig. 1. Example grammar for generation of prefix mathematical expressions

2.1 Mapping Process

To illustrate the mapping process employed in GE, consider the grammar shown in Fig. 1, composed of two *non-terminal* symbols (`<expr>` and `<digit>`) and 14 *terminal* symbols (`+`, `*`, `x`, `.` and `0...9`). Given an integer (genotype) string, such as `(1, 7, 4, 8, 6, 5, 9)`, a program (phenotype) can be constructed, which respects the syntax specified in the grammar.

This works by using each integer to choose productions from the grammar, mapping a given start symbol (typically, the first non-terminal symbol appearing in the grammar) to a sequence of terminal symbols. In this example, the first integer chooses one of the four productions of the start symbol `<expr>`, through the formula $1\%4 = 1$, i.e. the second production is chosen (as the count starts from 0), so the mapping string becomes `* <expr> <expr>`.

The following integer is then used with the leftmost unmapped symbol in the mapping string, so through the formula $7\%4 = 3$ the symbol `<expr>` is replaced by `<digit>.<digit>`, so the string becomes `* <digit>.<digit> <expr>`.

The mapping process continues in this fashion, so in the next step the mapping string becomes `* 4.<digit> <expr>` through the formula $4\%10 = 4$, and through $8\%10 = 8$ it becomes `* 4.8 <expr>`. Finally, the remaining non-terminal symbol is mapped with $6\%4 = 2$, and the final expression becomes `* 4.8 x`, which can then be evaluated.

The evolved strings may not have enough values to fully map syntactic valid programs; several options are available to address this issue, such as reusing the same integer string (in a process called wrapping [17]), assigning the individual the worst possible fitness, or replacing it with a legal individual. In this study, an unmapped individual is replaced by its originating parent.

3 Experimental Setup

3.1 Quality of Data and Input Variables

The calculation of NEE represents the balance between photosynthetic carbon assimilation or gross primary productivity (GPP) and net carbon release through ecosystem respiration (R_{eco}), which can be further sub-divided into autotrophic (R_a) and heterotrophic (R_{het}) components. Daytime NEE data represent the balance between GPP and soil derived R_{het}, while night time NEE data (R_{eco})

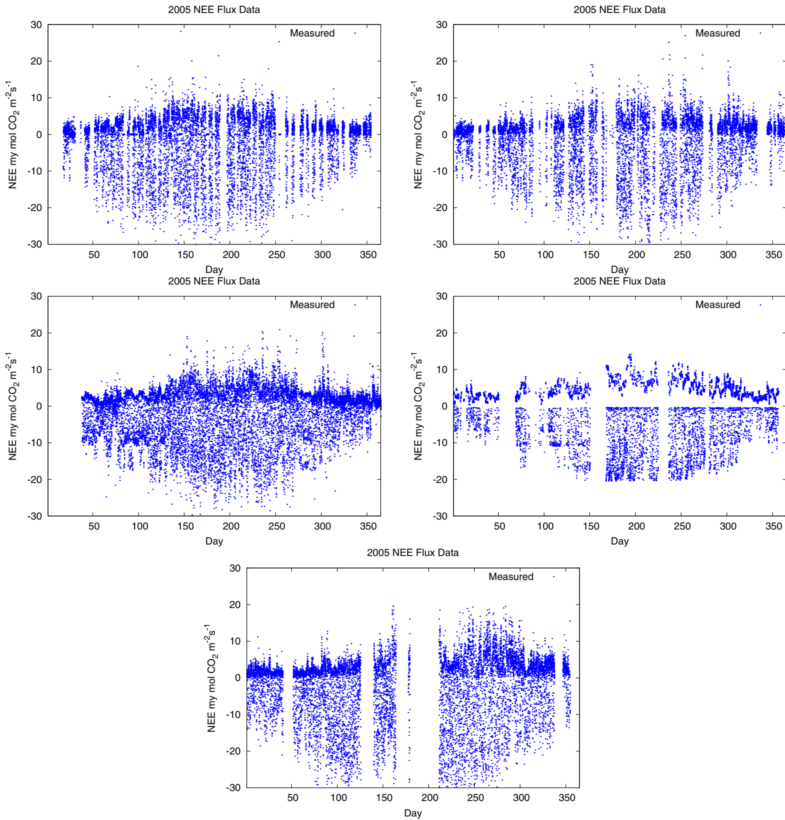


Fig. 2. Observed NEE flux data for the period 2002-2006. Negative NEE values indicate diurnal flux exchanges, whereas positive NEE flux typically occurs at night time.

represent the combined R_a and R_{het} CO_2 efflux from the plant and soil systems combined. The measurement of NEE in this study was made using the closed path EC technique, where fluxes of CO_2 were calculated over 30 minute intervals and the data post-processed and assigned a quality control standard according to the CarboEurope-IP criteria.

Daytime NEE tends to be controlled by both photosynthetic active radiation and air temperature, while R_{eco} is largely a temperature-dependant process. However, even the high quality diurnal flux data show considerable “noise” due to the multiple factors that influence NEE. Figure 2 shows the recorded NEE data for the period 2002-2006. As the annual carbon budget is the typically used unit, and due to annual variations (forest growth and management), each year is treated independently.

As the data is seasonal by nature, the time of day and day of year are used as input variables for model evolution. In order to reduce the linear cumulative numerical weight of these variables ($0 \ldots 23.5$ for time, and $0 \ldots 365$ for day),

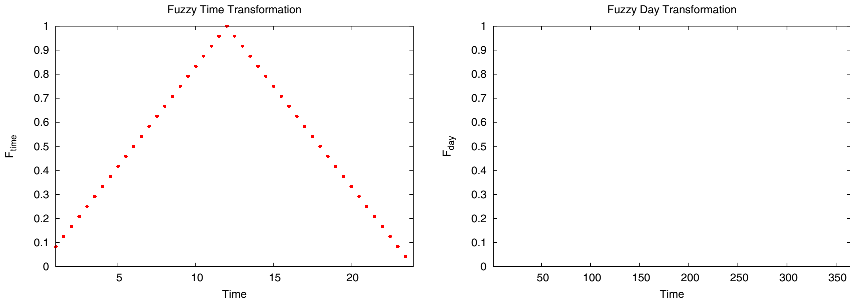


Fig. 3. Fuzzy time and day data transformations used for model evolution

Table 1. Experimental configurations

	Day & Night				Day only				Night only			
	B1	B2	B3	B4	D1	D2	D3	D4	N1	N2	N3	N4
F_{day}, F_{time}, TEMP	x	x	x	x	x	x	x	x	x	x	x	x
PAR	x	x	x	x	x	x	x	x				
sin, cos			x	x			x	x			x	x
TEMP$_{min}$, TEMP$_{max}$, TEMP$_{avg}$		x		x		x		x		x		x
PAR$_{min}$, PAR$_{max}$, PAR$_{avg}$		x		x		x		x				

they were transformed into two fuzzy sets, F_{time} and F_{day}, as seen in previous studies [18]; Fig. 3 shows the fuzzy transformations employed.

Additional meteorological measurements, traditionally used to describe ecosystem carbon flux and model NEE, included air temperature (TEMP), Photosynthetic Active Radiation (PAR), Relative Humidity (RH) and Precipitation (P). Some of these can also exhibit noise in their measurement, and full year-round data is sometimes not available; given the quality of the available data, TEMP and PAR were chosen as meteorological input variables.

Table 1 shows the configurations tested (B1...N4). As diurnal and nocturnal NEE flux dynamics are quite different, models were evolved for either a full dataset, or obtained by combination of separately evolved diurnal and nocturnal models; daytime and night time NEE data were sub-divided based on incident PAR, with data assigned to the daytime data class when $PAR > 10\ \mu$ mol $m^{-2}s^{-1}$ [13]. Also, given the somewhat regular nature of the data, trigonometric functions were tested in half of the configurations. Finally, some configurations were tested where historical data was used in the function set (PAR$_{min}$, PAR$_{max}$ and PAR$_{avg}$ as the minimum, maximum and average PAR data of the last 24 hours, and likewise for TEMP).

3.2 Evolutionary Setup

Grammar design. The grammars used correspond to the function sets detailed in Tab. 1. They are balanced grammars [7], which helps to control the size of

```
<e> ::= + <e> <e> |  - <e> <e> |  * <e> <e> |  / <e> <e>
      |  + <e> <e> |  - <e> <e> |  * <e> <e> |  / <e> <e>
      |  + <e> <e> |  - <e> <e> |  * <e> <e> |  / <e> <e>
      |  + <e> <e> |  - <e> <e> |  * <e> <e> |  / <e> <e>
      |  + <e> <e> |  - <e> <e> |  * <e> <e> |  / <e> <e>
      |  Fday[i] |  Fhour[i] |  PAR[i] |  TEMP[i] |  <d><d>"."<d>
      |  Fday[i] |  Fhour[i] |  PAR[i] |  TEMP[i] |  <d><d>"."<d>
      |  Fday[i] |  Fhour[i] |  PAR[i] |  TEMP[i] |  <d><d>"."<d>
      |  Fday[i] |  Fhour[i] |  PAR[i] |  TEMP[i] |  <d><d>"."<d>
<d> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fig. 4. Grammar used for setting B1

resulting individuals and thus delaying the onset of bloat; to do so, several equal productions were inserted, to maintain the biases of transformations [16]. Finally, the number of non-terminals was reduced, as this has been shown to help improve the performance of GE [16]. Fig. 4 shows the grammar used for setting B1.

To evolve the integer strings used with GE, a variable-length genetic algorithm was used [8]. The first generation was created using a ramped version of *Sensible Initialisation* [21], resulting in a better spread of initial solutions (albeit not perfect [7]). A “fair” tournament selection was used, where every individual participates at least in one tournament event. Finally, genetic operators were applied only to mapping regions of chromosomes.

Table 2 details the evolutionary parameters used. Note that, since the models evolved for day time and night time are later combined together, the computation effort of their runs doubles that of the runs where a single model is evolved; taking this into account, the population size of the latter (B1. . . B4) is doubled, resulting in a comparable computation effort per generation.

Table 2. Evolutionary Setup

Parameter	Value
Population Size	500/1000
Generations	50
Derivation-tree Max Depth (for initialisation)	5
Tail Ratio (for initialisation)	50%
Selection Tournament Size	1%
Elitism (for generational replacement)	10%
Crossover Ratio	50%
Average Mutation Events per Individual	1

3.3 Measuring Performance

Evolved models were compared to available NEE data, and the mean squared error between predictions and available data was used as a performance measure. The available NEE data was divided into training and test sets, so as to ascertain

Table 3. Mean squared error (and standard deviation) on test data

	B1	B2	D1 + N1	D1 + N2	N2 + D1	D2 + N2
2002	27.58 (1.49)	26.70 (7.62)	25.32 (0.76)	25.44 (0.83)	27.28 (2.95)	27.40 (3.01)
2003	22.68 (1.50)	25.48 (1.86)	21.15 (1.14)	21.24 (1.08)	21.64 (0.99)	21.78 (0.98)
2004	16.51 (4.74)	19.43 (1.30)	17.54 (1.59)	17.56 (1.55)	19.23 (2.19)	19.00 (1.93)
2005	12.72 (5.23)	22.13 (14.39)	7.13 (2.10)	7.25 (2.09)	8.65 (2.66)	8.66 (2.76)
2006	33.76 (49.94)	25.23 (5.99)	20.15 (2.84)	20.19 (2.78)	24.85 (3.80)	24.77 (3.91)

how well the evolved models generalise to unseen data. In this study, for every four available data points, the first three were used for training, with the fourth used for testing. [1]

4 Results and Analysis

Results using trigonometric functions were on par or worse than the equivalent setups without these functions, and generally produced more complex expressions, so in accordance with the Occam's Razor principle, they were discarded (they are not reported here). Table 3 reports the mean squared error on test data, for all other configurations, averaged over 50 runs. Average minimum error at end of evolution and standard deviation are reported.

The results obtained match the relative quality of different annual data, as could be observed in Fig. 2 (steady improvement of data quality over the years, apart from 2006). Evolving separate daytime and night time models generally provides better performing models. The use of historical data seems to make no difference to the results, but the resulting night time models are more compact on average and were thus preferred. Also note that combined models can be further enhanced, as models obtained in different runs can be matched.

Figure 5 plots the measured NEE flux data for 2005, and the best single (B1) and combined (D1 + N2) models. The difference in performance, particularly for positive NEE values (night time data) is substantial. Also note the occurrence of asymptotes when evolving a single model (Fig. 5 top), suggesting that the use of interval arithmetic [11] might be required to remove these. Figure 6 shows the combined model prediction for April 2005, highlighting both the matching of measured EC and the interpolation of regions with no data recorded.

Figure 7 plots the average training and testing performance over time, for the (D1 + N2) configuration. It can be seen that the model does not overfit the data. Comparison with runs using all the available data for training achieved similar results, suggesting that the use of a 2-set methodology neither hinders nor improves the performance of the obtained models, confirming previous results reported in GP [5] and GE [23].

[1] A more typical division, such as an initial large proportion of data for training and the remaining for testing, is not feasible, given the seasonality of the data, and the uniqueness nature of each year (different models are evolved for different years).

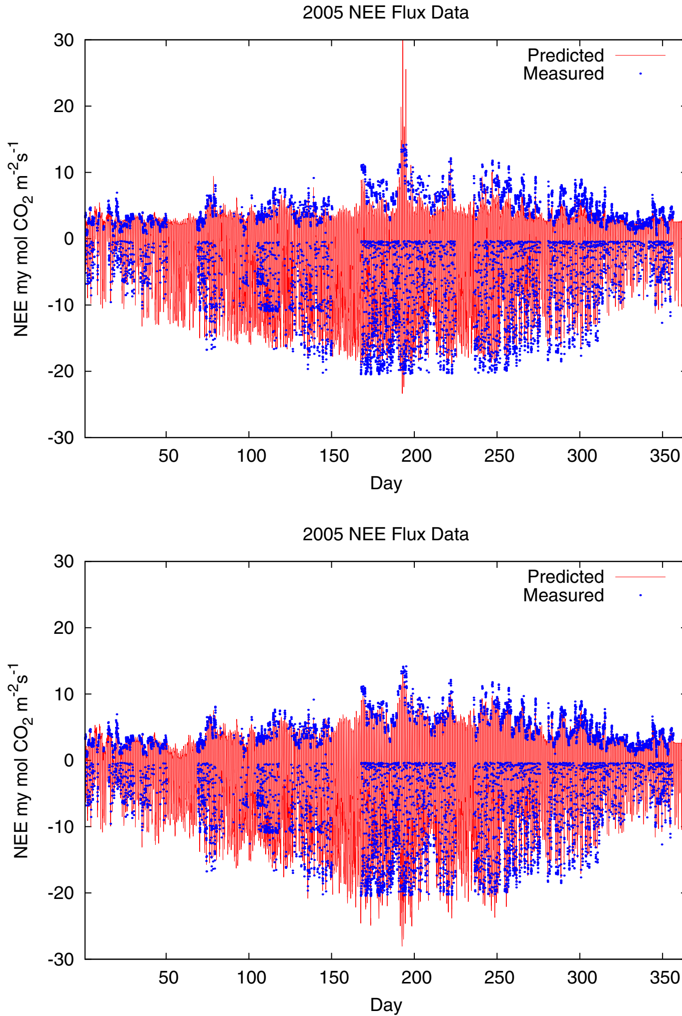


Fig. 5. Eddy value predictions of best full data predictor (top) and combined predictions of the best day time and night time models

The best (D1 + N2) model is shown in Eq. 1. The model has not been simplified; it shows a remarkably compact solution, resulting from the bloat delaying techniques described above, and the relatively short runs (50 generations). Note that night time data only makes use of temperature, showing that the seasonality of this variable is sufficient to match the seasonality of the Eddy values. Similarly, the daytime equation makes no use of F_{time}, showing that daily regularity can be modelled by PAR and $TEMP$. Finally, Fig. 8 shows the correlation between test data and model prediction for this model, including a 1:1 line. The model exhibits a good correlation with measured data, apart from some instances around values close to zero (a mixture of both noisy data and incorrect predictions).

$$\text{eddy} = \begin{cases} \dfrac{\dfrac{93.7}{25.7 + TEMP_{min}} + TEMP}{\dfrac{99.9}{23.6 + TEMP_{avg}}} & \text{if } PAR < 10.0 \\[2ex] \dfrac{13.1 + PAR}{\dfrac{TEMP + PAR}{F_{day} - (TEMP + 11.7)} - 21.0} & \text{otherwise} \end{cases} \tag{1}$$

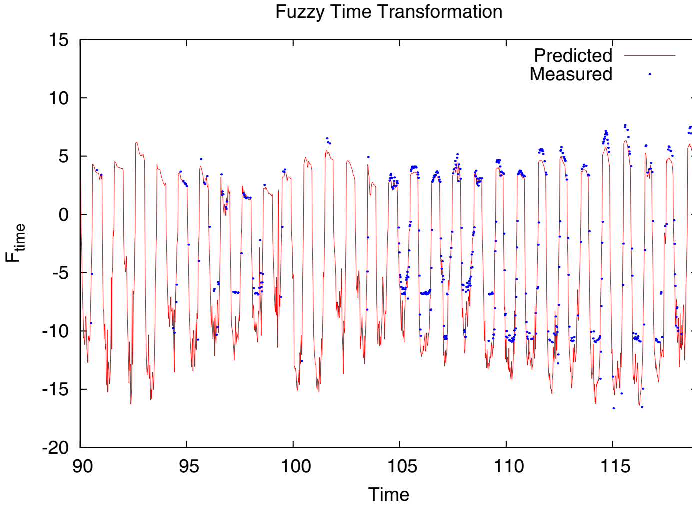


Fig. 6. Measured Eddy value vs. best model prediction, for the month of April 2005

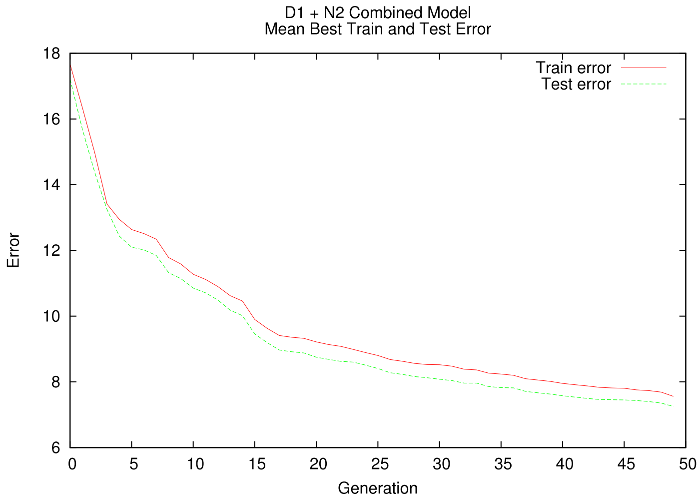


Fig. 7. Training and testing performance for the mean best individual per generation, for (D1 + N2) configuration (averaged across 50 runs)

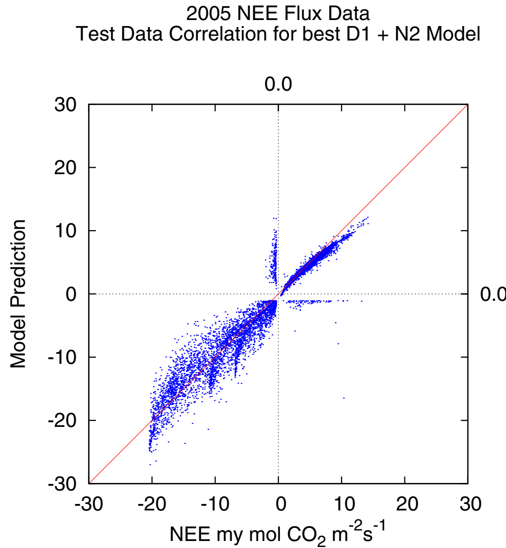


Fig. 8. Correlation between (unseen) test data and model prediction

5 Conclusions

GP in its many flavours has been applied to a multitude of symbolic regression problems over the years, with outstanding results. Yet, in most research fields, standard gap-filling methods such as look-up tables and linear interpolation are still applied as standard. This work presents a collaboration between evolutionary computation practitioners and environmental biologists, in an effort to further highlight the applicability of GP to generate gap-filling models for measured environmental data.

Due to the unique nature of data from different forest sites, proper comparison with other methods is hard to achieve[2]; however, the results obtained seem to be on par with the best in the literature [18]. Not only that, but the use of GP has certain advantages. By providing symbolic models, stating the required input variables, decisions can be made about the required annual measurements, affecting both budget and work force management.

There are plenty of future work directions. The evolved models can have a direct impact on forest management and even policy making, and thus continued efforts to improve their accuracy are ongoing. Another exciting future work direction involves identifying a maximum size of measured data gaps; this allows expensive equipment to be used and rotated across different sites, thus bringing the overall data-gathering costs down. Efforts are ongoing to achieve this.

[2] The data presented here has only been analysed with the current method so far.

Acknowledgements. The authors would like to acknowledge Alexandros Agapitos, for sharing his insightful knowledge of GP and associated techniques. This research is based upon works supported by Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References

1. Moffat, A., et al.: Comprehensive comparison of gap-filling techniques for eddy covariance net carbon fluxes. Agricultural and Forest Meteorology 147, 209–232 (2007)
2. Azad, R.M.A., Ansari, A.R., Ryan, C., Walsh, M., McGloughlin, T.: An evolutionary approach to wall shear stress prediction in a grafted artery. Applied Soft Computing 4(2), 139–148 (2004)
3. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Springer, Heidelberg (2006)
4. Falge, E., et al.: Gap filling strategies for defensible annual sums of net ecosystem exchange. Agricultural and Forest Meteorology 107, 43–69 (2001)
5. Gagné, C., Schoenauer, M., Parizeau, M., Tomassini, M.: Genetic Programming, Validation Sets, and Parsimony Pressure. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 109–120. Springer, Heidelberg (2006)
6. Goulden, M., Munger, W., Fan, S.M., Daube, B., Wofsy, S.: Measurements of carbon sequestration by long-term eddy covariance: methods and critical evaluation of accuracy. Global Change Biology 2, 169–182 (1996)
7. Harper, R.: GE, explosive grammars and the lasting legacy of bad initialisation. In: Proceedings of IEEE Congress on Evolutionary Computation, CEC 2010, July 18-23, Barcelona, Spain, pp. 2602–2609. IEEE Press (2010)
8. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
9. Hui, D., Wan, S., Su, B., Katul, G., Monson, R., Luo, Y.: Gap-filling missing data in eddy covariance measurements using multiple imputation (mi) for annual estimates. Agricultural and Forest Meteorology 121, 93–111 (2004)
10. Humphreys, E., Black, T.A., Morgenstern, K., Cai, T., Drewitt, G., Nesic, Z., Trofymow, J.: Carbon dioxide fluxes in coastal douglas-fir stands at different stages of development after clearcut harvesting. Agricultural and Forest Meteorology 140, 6–22 (2006)
11. Keijzer, M.: Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 70–82. Springer, Heidelberg (2003)
12. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
13. Reichstein, M., et al.: On the separation of net ecosystem exchange into assimilation and ecosystem respiration: review and improved algorithm. Global Change Biology 11, 1424–1439 (2005)
14. McKay, R.I., Nguyen, X.H., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming - a survey. Genetic Programming and Evolvable Machines 11(3-4), 365–396 (2010)

15. Murphy, J.E., O'Neill, M., Carr, H.: Exploring Grammatical Evolution for Horse Gait Optimisation. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 183–194. Springer, Heidelberg (2009)
16. Nicolau, M.: Automatic grammar complexity reduction in grammatical evolution. In: Poli, R., et al. (eds.) Genetic and Evolutionary Computation Conference (GECCO) Workshops. AAAI (2004)
17. O'Neill, M., Ryan, C.: Grammatical Evolution - Evolutionary Automatic Programming in an Arbitrary Language. Genetic Programming, vol. 4. Kluwer Academic (2003)
18. Papale, D., Valentini, R.: A new assessment of european forests carbon exchanges by eddy fluxes and artificial neural network spatialization. Global Change Biology 9, 525–535 (2003)
19. Perez, D., Nicolau, M., O'Neill, M., Brabazon, A., Yannakakis, G.N.: Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcázar, A.I., Merelo, J.J., Neri, F., Pruess, M., Richter, H., Togelius, J., Yannakakis, G.N. (eds.) EvoApplications 2011, Part I. LNCS, vol. 6624, pp. 123–132. Springer, Heidelberg (2011)
20. Pingintha, N., Leclerc, M., Beasley, J., Durden, D., Zhang, G., Senthong, C., Rowland, D.: Hysteresis response of daytime net ecosystem exchange during drought. Biogeosciences 7, 1159–1170 (2010)
21. Ryan, C., Azad, A.: Sensible initialisation in grammatical evolution. In: Cantú-Paz, E., et al. (eds.) Genetic and Evolutionary Computation Conference (GECCO) Workshops. AAAI (2003)
22. Ryan, C., Collins, J., O'Neill, M.: Grammatical Evolution: Evolving Programs for an Arbitrary Language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 83–95. Springer, Heidelberg (1998)
23. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: A Preliminary Investigation of Overfitting in Evolutionary Driven Model Induction: Implications for Financial Modelling. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Urquhart, N., Uyar, A.Ş. (eds.) EvoApplications 2011, Part II. LNCS, vol. 6625, pp. 120–130. Springer, Heidelberg (2011)

Multi-Objective Ant Programming for Mining Classification Rules

Juan Luis Olmo, José Raúl Romero, and Sebastián Ventura

Dept. of Computer Science and Numerical Analysis,
University of Cordoba, Rabanales Campus, Albert Einstein Building,
14071 Cordoba, Spain
{jlolmo,jrromero,sventura}@uco.es

Abstract. Ant programming (AP) is a kind of automatic programming that generates computer programs by using the ant colony optimization metaheuristic. It has recently demonstrated a good generalization ability when extracting classification rules. We extend the investigation on the application of AP to classification, developing an algorithm that addresses rules' evaluation using a novel multi-objective approach specially devised for the classification task. The algorithm proposed also incorporates an evolutionary computing niching procedure to increment the diversity of the population of programs found so far. Results obtained by this algorithm are compared with other three genetic programming algorithms and other industry standard algorithms from different areas, proving that multi-objective AP is a good technique at tackling classification problems.

Keywords: Data mining, Classification, Ant programming, Genetic programming, Multi-objective optimization.

1 Introduction

Genetic programming (GP) is an evolutionary technique that offers a great potential for inducing classifiers. It is a very flexible metaheuristic that can be adapted to adhere to the particular needs of a specific problem, as it can use different representations, such as decision trees or classification rules.

Several GP proposals have been presented addressing the classification task. For instance, a constrained syntax algorithm was presented by Bojarczuk et al. [1], where one rule in disjunctive form predicting each class is induced, so that individuals represent rule sets. Each individual looks for optimizing a scalar aggregation of three objectives: specificity, sensitivity and complexity.

The system presented by Tan et al. [2] tackles multiclass classification performing a different running for each of the classes to be distinguished. Each individual represents a rule, but a niching mechanism and elitism are employed in such a way that a set of rules, all of them predicting the same class, are obtained at the end of the evolutionary process. Hence, the final classifier can have several rules for each of the classes. An accuracy-based fitness function is employed.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 146–157, 2012.
© Springer-Verlag Berlin Heidelberg 2012

The solution construction in GP is based on the combination or modification of other solutions, by using crossover and mutator operators. Nevertheless, this conduct presents several drawbacks, concerning the lack of information about the distribution of the search space, its difficulty when adapting to a changing environment, and the genotype-phenotype mapping which implies that a small change may provoke a high impact on the phenotype [3].

On the other hand, ant colony optimization (ACO) [4] is a constructive method that builds a solution by following a sequence of transitions guided by certain information (heuristic and pheromones), and its application to the automatic construction of programs, a.k.a. ant programming (AP), does not present the aforementioned problems. A short review of AP and its applications can be found in [5], where an AP algorithm for extracting rule-based classifiers is also presented.

Multi-objective optimization appears as a natural way to assess the quality of rules mined in classification problems. It allows to optimize simultaneously several measures that sometimes are conflicting, which means that the increase in the values for one may be detrimental for the other. For example, accuracy and interpretability are conflicting goals, because simpler classifiers tend to obtain lower results in terms of accuracy, and viceversa.

In this paper, we focus on extending the ideas presented in [5], exploring the use of multi-objective optimization to compute the fitness of individuals generated. The contribution of this work is not only allocated to the AP field, but to the classification in general, proposing a new Pareto-based evolutionary strategy that could be adopted by any evolutionary algorithm where each individual represents a rule. The algorithm developed also employs a niching approach to select the rules that make up the output classifier, minimizing overlapping between them. The performance of the proposed algorithm is compared over several standard classification problems to other nine classification algorithms from several paradigms, including three GP algorithms and the original mono-objective AP algorithm proposed for this task. Regarding predictive accuracy, the developed algorithm outperforms statistically most of the others considered, and it also reaches a good trade-off between accuracy and comprehensibility.

The paper is structured as follows. Section 2 presents a detailed description of the AP algorithm proposed. Section 3 explains the experimental set up, listing the benchmark problems, and describing the configuration used for the algorithms involved and how the experiments are carried out. Section 4 shows and analyzes the results obtained. Finally, in Section 5 we present the conclusions and propose some lines for future work.

2 The Multi-Objective Grammar Based Ant Programming (MOGBAP) Algorithm

2.1 Environment and Individual Encoding

The MOGBAP algorithm has been devised for inducing a classifier from a learning process over a given training set, the classifier acting as a decision list.

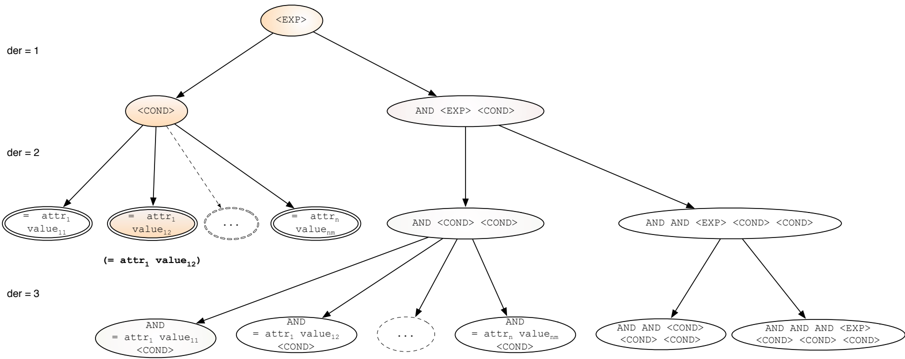


Fig. 1. Space of states generated by the grammar at a depth of three derivations

MOGBAP employs a context-free grammar, which substitutes the terminal and function set, restricting the search space and granting that solutions found are syntactically valid. Hence, the closure property is then guaranteed by the application of the production rules from the start symbol of the grammar.

Any ant inspired algorithm requires an environment where ants cooperate with each other, simulating the stigmergy process that occurs in nature. In MOGBAP, this environment consists of all the possible expressions or programs that can be derived from the grammar in a given number of derivations. The environment adopts the shape of a derivation tree, as depicted in Figure 1. A program or individual in the population will adopt the shape of a path over the derivation tree, as shown with the sample shaded path. The last state of a path is a final state, composed only of terminal symbols. Final states are represented by double-line states. Although the last state of a path encodes the evaluatable expression for the antecedent, ants have an internal memory to store the whole path, fulfilling the properties of an artificial ant. This allow ants to perform an offline pheromone update.

In MOGBAP, each individual or program represents an ant, and it encodes a rule. Thus, concerning the individual encoding, it follows the $individual = rule$ approach [6]. The consequent assigned to a given ant corresponds to the most frequent class among the training instances covered by its antecedent.

2.2 Heuristic Measures, Transition Rule and Pheromone Maintenance

MOGBAP uses two complementary heuristic functions that can not be applied simultaneously, the same used by its precursor, GBAP [5]. The former, named cardinality probability, is used in intermediate transitions, i.e., those not involving the selection of attributes of the problem domain. This measure aims to guide ants to select transition that may lead to a greater number of solutions. The latter is the widely used information gain, which measures the worth of each attribute for separating the training examples with respect to their classification

target. This complementary heuristic function is applied in case of transitions that imply the selection of attributes of the problem domain.

Any ant-based algorithm follows a probabilistic stepwise solution construction method. MOGBAP generates a new individual incrementally by following a sequence of steps or transitions. The information that biases these transitions is given by the aforementioned two-fold heuristic function and the strength of the pheromone trails in each transition, as shown in Equation 1:

$$P_{ij}^{k} = \frac{(\eta_{ij})^{\alpha} \cdot (\tau_{ij})^{\beta}}{\Sigma_{k \in allowed} (\eta_{ik})^{\alpha} \cdot (\tau_{ik})^{\beta}} \tag{1}$$

which represents the probability that a given ant at the state i will take the transition to the state j, k being the number of valid subsequent states, α standing for the heuristic exponent and β for the pheromone's, η being the value of the heuristic function, and τ indicating the strength of the pheromone trail.

Notice that a probability value is assigned to each next state available, but the number of transitions that can be followed is limited by number of derivations allowed for the grammar. Thus, if moving to a given state does not permit any solution to the problem to be found in the number of derivations remaining available at that point, the algorithm will assign a probability of zero to this state, so it will never be selected.

Two categories are considered for multi-objective ant-based algorithms depending on how they store the pheromone information [7], those using just one pheromone matrix, or those that use a pheromone matrix per objective. MOGBAP belongs to the first group, which entails a benefit regarding memory and computational time requirements.

In contrast to GP, where individuals evolve along generations combining genetic material by means of the crossover, mutation and reproduction operators, individuals in AP interact in an indirect manner by means of the reinforcement and evaporation operations over the environment. In MOGBAP, evaporation takes place over the complete space of states:

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot (1 - \rho) \tag{2}$$

where τ_{ij} represents the amount of pheromone in the transition from state i to state j, and ρ represents the evaporation rate.

In turn, concerning pheromone reinforcement, only non-dominated ants are able to retrace their path to update the amount of pheromone in the transitions followed. For a given individual, all transitions in its path are reinforced equally, and the value of this reinforcement is based upon the length and the quality of the solution encoded, represented by the Laplace accuracy [5]:

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot Q \cdot LaplaceAccuracy \tag{3}$$

where Q is a computed measure that favors comprehensible solutions, computed as the ratio between the maximum number of derivations in the current generation and the length of the path followed by the ant (thus shorter solutions will receive more pheromone).

At the end of each generation, a normalization process takes place, in order to bind the pheromone level in each transition to the range $[\tau_{min}, \tau_{max}]$. In addition, at the first generation of the algorithm, all transitions in the space of states are initialized with the maximum pheromone amount allowed.

2.3 Muti-Objective Strategy and Niching Procedure

The quality of the individuals generated in MOGBAP is assessed on the basis of three conflicting objectives: sensitivity, specificity and comprehensibility. A given rule ant_i is said to dominate another rule ant_j, written $ant_i \succ ant_j$, if ant_i is not worse than ant_j in two objectives and is better in at least one objective.

Sensitivity and specificity are two measures widely employed in classification problems, even as a scalar function of them. Sensitivity indicates how well a rule identifies positive cases. On the contrary, specificity reports the effectiveness of a rule's identifying negative cases or those cases that do not belong to the class studied. If the sensitivity value of a rule is increased, it will predict a greater number of positive examples, but sometimes at the expense of classifying as positives some cases that actually belong to the negative class. Both objectives are to be maximized.

$$Sensitivity = \frac{T_P}{T_P + F_N} \qquad Specificity = \frac{T_N}{T_N + F_P} \tag{4}$$

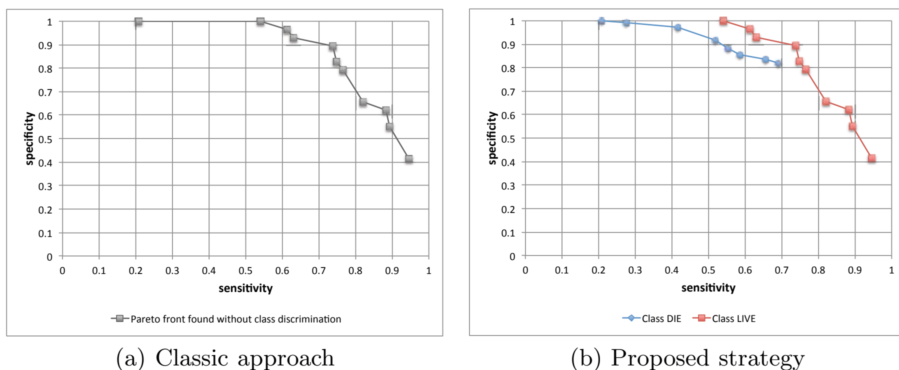
Since MOGBAP is a rule-based classification algorithm, it is intended to mine accurate but also comprehensible rules. Assuming that a rule can have up to a fixed number of conditions, comprehensibility can be measured as:

$$Comprehensibility = 1 - \frac{numConditions}{maxConditions} \tag{5}$$

where $numConditions$ refers to the number of conditions appearing in the rule encoded by the individual, whereas $maxConditions$ is the maximum number of conditions that a rule can have [8]. The advantage of using this comprehensibility metric lies in the fact that its values will be contained in the interval [0,1], and the closer its value to 1, the more comprehensible the rule will be. Hence, just as with the objectives of sensitivity and specificity, this objective, too, should be maximized.

MOGBAP follows a multi-objective strategy that has been specially designed for the classification task. The idea behind this scheme is to distinguish solutions in terms of the class they predict, because certain classes are more difficult to predict than others. Actually, if individuals from different classes are ranked according to Pareto dominance, overlapping may occur, as illustrated in Figures 2 and 3, which show the Pareto fronts found after running MOGBAP for the hepatitis and lymphography data sets, considering the objectives of sensitivity and specificity.

For instance, for the hepatitis problem, if a classic Pareto approach were employed, a single front of non-dominated solutions would be found, as shown in Figure 2(a). Hence, among the individuals represented in this figure, such a



(a) Classic approach (b) Proposed strategy

Fig. 2. Comparison between the proposed strategy and a classic approach for the the two-class data set hepatitis

Pareto front would consist of all the individuals that predict the class 'LIVE' and just one individual of the class 'DIE' (the individual which has a specificity of 1.0). In order for the remaining individuals of the class 'DIE' to be considered, it would be necessary to find additional fronts, and they would have less likelihood of becoming part of the classifier's decision list. On the other hand, the multi-objective approach of MOGBAP (see Figure 2(b)) guarantees that all non-dominated solutions for each available class will be found, so it ensures the inclusion of rules predicting each class in the final classifier.

Moreover, considering now the lymphography data set as depicted in Figure 3, the Pareto front found with a classic approach would consist in just one point with a sensitivity and a specificity of 1.0. In order to contemplate rules predicting the class 'MALIGN_LYMPH' or 'METASTASES', it would be necessary to find more intermediate fronts.

Roughly speaking, the multi-objective approach devised for MOGBAP consists in discovering a separate set of non-dominated solutions for each class in the data set. To this end, once individuals of the current generation have been created and evaluated for each objective considered, they are divided into k groups, k being the number of classes in the training set, according to their consequent. Then, each group of individuals is combined with the solutions kept in the corresponding Pareto front found in the previous iteration of the algorithm, to rank them all according to dominance, finding a new Pareto front for each class. Hence, there will be k Pareto fronts, and only the non-dominated solutions contained will participate in the pheromone reinforcement.

The final classifier is built from the non-dominated individuals that exist in the k Pareto fronts once the last generation has finished. A niching procedure executed over each one of the k fronts is in charge of making up the decision list from these rules: the individuals of the front are sorted by the Laplace accuracy [9] and then they try to capture as many instances of the training set as they can. Each ant can capture an instance just in case it covers it and the instance has not been seized previously by another ant. Finally, only those ants

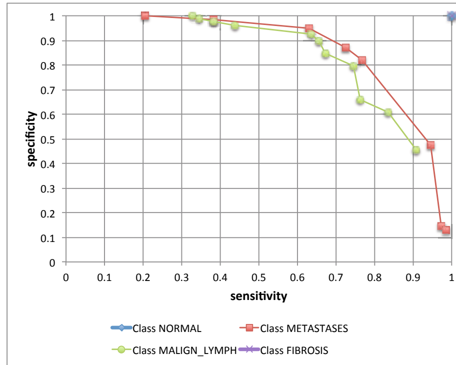


Fig. 3. Pareto fronts found by MOGBAP for the four-class data set lymphography

whose number of captured instances exceeds the percentage of coverage established by the user are added to the list of returned ants, having an adjusted Laplace accuracy computed as follows:

$$LaplaceAccuracy_{adj} = LaplaceAccuracy \cdot \frac{capturedInstances}{idealInstances} \tag{6}$$

where *idealInstances* is equal to the number of instances covered by the ant.

The resulting ants of carrying out the niching procedure over each Pareto front are added to the classifier, sorted by their adjusted Laplace accuracy. A default rule predicting the majority class in the training set is added at the bottom of the decision list and the classifier is run over the test set to compute its predictive accuracy.

3 Experimental Set-Up

An empirical study has been directed to conclude whether multi-objective AP is a competitive technique at extracting comprehensible and accurate classifiers, comparing its results with those obtained by other well-known algorithms from several paradigms. To this end, the experimental study was directed as follows:

- Fifteen real data sets from the UCI [10] machine learning repository were employed in the experimentation, presenting a broad range of characteristics regarding dimensionality, type of attributes and number of classes.
- In order to perform a fair comparison, two preprocessing steps were carried out. Firstly, missing values were replaced with the mode or the arithmetic mean, assuming categorical and numeric attributes, respectively. Secondly, since MOGBAP can not cope with continuous variables, a discretization procedure has to be applied to turn all the continuous attributes into categorical. The Fayyad and Irani's [11] algorithm was used for such purpose. Both steps were performed using WEKA[1].

[1] The open source data mining workbench WEKA is available at http://www.cs.waikato.ac.nz/ml/index.html

- A stratified 10-fold cross validation procedure was followed to evaluate the performance of the algorithms. In case of non-deterministic algorithms we have used 10 different seeds for each partition, so that for each data set we consider the average values obtained over 100 runs.
- For comparison purposes, we considered several rule-based algorithms belonging to different paradigms. Two AP algorithms, the proposed algorihtm, MOGBAP, and GBAP [5]. Three ant-based algorithms, Ant-Miner [12], Ant-Miner+ [13] and the hybrid PSO/ACO2 algorithm [14]. Three GP algorithms, a constrained syntax algorithm proposed by Bojarczuk et al. [1]; the algorithm presented by Tan et al. [2], which implements a niching mechanism that bears some resemblance with the niching procedure used by MOGBAP; and the recently proposed ICRM algorithm [15], which generates very interpretable classifiers in terms of number of rules and number of conditions. The reduced error pruning JRIP algorithm [16]. And PART [17], which extracts rules from a decision tree.

  MOGBAP uses the same parameter configuration as GBAP, i.e., a population of 20 ants, 100 iterations, 15 derivations allowed for the grammar, a minimum coverage of 5%, an initial and maximum pheromone amount of 1.0, a minimum pheromone amount of 0.1, an evaporation rate of 0.05, a value of 0.4 for α and 1.0 for β . The other algorithms were executed using the parameters suggested by their authors. The following implementations were employed: for MOGBAP and GBAP, we used our own implementations. For Ant-Miner, the open source code provided in the framework Myra was employed[2]. In case of Ant-Miner+, the code given by the authors was used. To run PSO/ACO2, its open-source implementation was used[3]. The three GP algorithms were run using the implementations available in the framework JCLEC[4]. Finally, PART and JRIP were run by using the implementations available in WEKA.

4 Experimental Results

4.1 Predictive Accuracy Analysis

A first experimental study focuses on determining whether MOGBAP obtains an accuracy performance comparable to or better than the other algorithms mentioned in Section 3.

Each row in the top half of Table 1 shows the average accuracy results in test obtained by each algorithm for a given data set, with the standard deviation. Bold type values indicate the algorithm that attains the best result for a particular data set. We can observe at a glance that MOGBAP reaches the best results in a 40% of the data set considered.

---

[2] Myra framework is available at http://myra.sourceforge.net/

[3] PSO/ACO2 is publicly available at http://sourceforge.net/projects/psoaco2

[4] JCLEC framework is available at http://jclec.sourceforge.net

Table 1. Results of the experimental study

Data set	MOGBAP		GBAP		AntMiner		AntMiner+		PSOACO2		Bojarczuk		Tan		ICRM		JRIP		PART	
	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}	Acc	σ_{Acc}
Hepat.	**85.15**	1.52	82.17	12.04	83.27	10.32	81.79	10.30	84.59	9.33	71.05	14.45	81.64	12.34	74.25	12.36	81.54	12.05	84.64	7.66
Sonar	79.49	9.26	**81.98**	7.44	76.95	6.89	76.05	7.22	78.49	8.05	79.82	9.24	73.45	7.70	69.16	8.66	80.33	6.61	77.84	8.10
Breast-c	72.02	9.62	71.40	7.86	**73.42**	7.29	73.05	6.86	68.63	6.87	68.63	10.94	60.59	10.56	63.61	8.28	72.00	6.41	68.48	7.90
Heart-c	**83.13**	4.24	82.84	5.24	78.01	6.69	82.41	5.10	82.25	5.36	70.02	7.08	77.14	6.20	73.86	4.69	82.20	5.12	80.13	6.39
Ionos.	90.55	5.67	**93.02**	4.07	84.39	6.73	92.89	4.02	89.97	4.99	76.48	8.19	87.17	6.13	88.96	7.43	91.70	5.14	88.93	4.02
Horse-c	**83.78**	4.67	82.97	6.34	82.71	4.73	81.79	6.03	82.06	4.93	82.52	6.06	82.15	6.77	82.89	7.21	83.72	6.35	81.5	3.72
Vote	94.89	2.92	94.37	3.57	94.29	3.27	94.66	3.72	94.80	3.81	**95.67**	2.78	95.60	3.65	**95.67**	3.61	95.44	3.52	94.51	3.08
Austr.	**87.38**	4.27	85.47	4.49	85.30	4.12	83.48	3.38	85.19	4.69	85.52	4.50	86.21	4.31	86.84	4.20	86.70	5.15	84.66	4.48
Breast-w	95.41	2.31	**96.50**	1.68	94.69	2.04	94.28	2.86	95.86	1.91	87.39	2.75	94.11	2.76	88.69	2.66	95.71	1.81	95.71	1.82
Credit-g	70.82	3.33	70.79	4.27	70.55	3.72	70.80	3.87	70.36	3.55	63.02	7.03	66.77	6.11	68.90	4.68	70.70	3.26	**72.70**	3.26
Iris	95.33	6.00	**96.00**	4.10	95.20	5.47	94.00	3.59	95.33	6.70	91.73	10.46	95.00	2.80	94.00	5.57	**96.00**	5.33	95.33	6.70
Wine	**98.24**	2.75	97.01	4.37	91.86	5.08	93.86	4.61	90.20	2.86	83.69	9.44	93.44	6.10	90.43	6.12	95.61	5.37	95.03	3.89
Lymph.	80.55	9.74	**81.00**	10.35	75.51	9.59	77.23	10.91	76.59	12.20	77.78	12.77	78.35	9.95	79.29	9.60	78.84	11.49	78.43	14.30
Glass	71.03	8.45	69.13	8.66	65.52	9.26	62.03	9.80	71.16	10.54	39.23	11.34	65.06	10.29	67.50	6.00	69.00	8.70	**73.91**	8.43
Primary	**42.18**	7.19	37.91	6.55	37.75	5.27	37.26	5.43	37.19	5.88	16.41	4.96	26.20	6.13	22.29	5.30	38.11	3.75	38.36	5.09
Rank.	**2.5333**		3.3		6.4667		6.2333		5.7		7.9333		7.2		6.8		3.7333		5.1	
	R	C/R	R	C/R	R	C/R	R	C/R	R	C/R	R	C/R	R	C/R	R	C/R	R	C/R	R	C/R
Hepat.	9.1	1.99	8.1	1.89	4.8	1.99	3.9	3.25	7.4	2.28	3.1	**1.22**	6.5	4.3	**2.0**	2.00	3.8	2.15	8.4	2.30
Sonar	11.1	2.04	12.3	1.81	5.2	2.07	4.0	3.48	6.1	2.92	3.0	**1.00**	7.6	4.27	**2.0**	2.00	4.6	2.21	13.9	2.98
Breast-c	11.8	1.69	13.2	1.91	6.0	1.28	5.4	2.82	11.8	1.75	3.5	**1.01**	9.0	4.36	**2.0**	2.00	3.3	1.70	17.1	2.12
Heart-c	9.9	2.25	14.5	1.67	5.9	1.20	4.4	2.82	11.9	3.81	**3.0**	3.02	6.5	4.38	5.0	**1.15**	5.3	2.32	17.3	2.35
Ionos.	6.8	1.49	11.1	1.18	5.7	1.61	8.8	1.41	4.5	4.03	3.1	**1.14**	5.3	4.32	**2.0**	2.00	7.7	1.48	8.2	1.83
Horse-c	9.6	2.03	9.0	1.46	6.3	1.49	4.7	3.41	20.1	3.39	3.0	**1.00**	6.9	4.25	**2.0**	1.90	3.5	1.74	13.2	2.38
Vote	6.6	2.12	17.2	2.19	5.6	1.36	5.2	2.34	6.1	1.33	3.0	**1.00**	3.1	3.21	**2.0**	**1.00**	3.1	1.38	7.7	1.84
Austr.	9.1	2.00	10.1	1.08	6.5	1.53	3.3	2.08	25.8	6.96	3.0	**1.00**	4.9	4.1	**2.0**	**1.00**	5.2	1.80	19.4	2.01
Breast-w	6.1	1.77	6.6	1.65	7.2	1.04	6.4	1.92	10.5	1.10	3.0	**1.00**	3.8	3.82	**2.0**	2.00	6.5	1.74	10.9	1.63
Credit-g	11.6	1.82	22.9	1.82	9.1	1.51	3.3	3.31	52.8	4.20	3.3	**1.17**	9.5	4.25	**2.0**	2.00	7.1	2.54	57.8	2.70
Iris	5.8	1.15	3.7	1.06	4.3	1.03	3.9	1.80	**3.0**	1.20	4.3	1.29	4.1	2.69	**3.0**	**1.00**	**3.0**	**1.00**	4.6	**1.00**
Wine	6.1	1.47	7.2	1.50	5.1	1.33	**2.5**	2.19	4.0	1.73	4.1	**1.27**	5.6	3.88	3.0	2.00	4.2	1.56	6.3	1.77
Lymph.	11.9	1.55	10.2	1.60	4.7	1.69	4.6	2.83	15.6	2.11	5.1	**1.02**	9.1	3.79	**4.0**	1.25	6.9	1.53	10.2	2.30
Glass	17.5	2.22	21.6	1.79	8.4	1.76	12.4	4.10	24.5	3.13	8.2	**1.48**	11.9	4.07	**7.0**	1.67	8.0	2.03	13.7	2.32
Primary	34.9	2.53	45.9	2.60	12.1	3.35	9.3	8.50	86.5	6.01	23.7	1.37	46.6	4.16	22.0	1.88	**8.3**	3.13	48.7	3.23
R	7.5666		8.1666		5.2333		3.8333		7.4333		2.9333		5.5666		**1.4666**		3.7666		9.0333	
C/R	5.1333		3.7		3.7333		8.3333		7.1333		**2.1333**		9.4666		4.4		4.2666		6.7	

Table 2. Friedman and Iman&Davenport tests for predictive accuracy results

	Critical value	Statistic	Hypothesis
Friedman	1.954952	7.562807	Rejected
Iman&Davenport	16.918977	47.349075	Rejected

However, to analyze statistically these results, the Friedman test [18] was applied. This test computes the average rankings obtained by k algorithms over N data sets regarding one measure, distributed according to the χ^2-distribution with (k-1) degrees of freedom, stating the null-hypothesis of equivalence among all the algorithms. Iman&Davenport's test considers a less conservative statistic distributed according to the F-distribution with (k-1) and (k-1)(N-1) degrees of freedom. Table 2 indicates that both Friedman's and Iman&Davenport's statistics do not belong to their respective critical intervals. Hence, the null-hypothesis that all algorithms perform equally well is rejected with a likelihood of 95%.

Table 3. Holm table for $\alpha = 0.05$

i	Algorithm	z	p	α/i	Hypothesis
9	Bojarczuk	4.884484	1.0370E-6	0.005556	Rejected
8	Tan	4.221159	2.4304E-5	0.00625	Rejected
7	ICRM	3.859345	1.1369E-4	0.007143	Rejected
6	Ant-Miner	3.557834	3.7393E-4	0.008333	Rejected
5	Ant-Miner+	3.346776	8.1757E-4	0.01	Rejected
4	PSO/ACO2	2.864358	0.004179	0.0125	Rejected
3	PART	2.321637	0.020252	0.016667	Accepted
2	JRIP	1.085441	0.277726	0.025	Accepted
1	GBAP	0.693476	0.488010	0.05	Accepted

The p value is a measure of the credibility of the null hypothesis. If something is very unlikely to have occurred by chance, we say that it is statistically significant. The Holm test [18] is a step-down posthoc procedure that tests the hypotheses ordered by significance, comparing each p_i with $\frac{\alpha}{i}$ from the most significant p value. Table 3 captures all the possible hypotheses of comparison between the best ranked algorithm, MOGBAP, and the others, ordered by their p value and associated with their level of significance α. When comparing MOGBAP with each of the other algorithms, a p value less or equal to 0.05 means that the hypothesis is unlikely to be true, the difference between both algorithms being statistically significant. Thus, at a significance level of $\alpha =$ 0.05, MOGBAP outperforms statistically the following algorithms: PSO/ACO2, Ant-Miner+, Ant-Miner, ICRM, Tan and Bojarczuk.

4.2 Comprehensibility Analysis

This section analyzes statistically the complexity of the rule set and the rules mined by each algorithm, which can be observed in the bottom half of Table 1,

where the column R indicates the average number of rules obtained by an algorithm over each data set, and C/R stands for the average number of conditions per rule. The last but one row of this table represents the average ranking of each algorithm with regards to the rule set length, and the last row specifies the average rankings regarding the number of conditions per rule. However, due to space limitations we just report on the conclusions obtained after carrying out the same statistic tests done in the previous section.

Regarding the number of rules in the classifier, the best result would be to extract one rule predicting each class in the data set. Nevertheless, this may be detrimental for the accuracy of the algorithm, as occurs in Bojarczuk and ICRM algorithms. At a significance level of $\alpha = 0.05$, ICRM obtains significant differences with respect to Ant-Miner, Tan, PSO/ACO2, MOGBAP, PART and JRIP, in this order. Bojarczuk, JRIP and Ant-Miner+ also behaves significantly better than MOGBAP regarding this metric.

Concerning the complexity of the rules mined, at a significance level of $\alpha = 0.05$, Bojarczuk obtains significant differences with MOGBAP, PART, PSO/ACO2, Ant-Miner+ and Tan algorithms. Bojarczuk is the unique algorithm capable of behaving statistically better than our proposal in this sense. In addition, MOGBAP achieves significant differences with respect to Ant-Miner+ and Tan algorithms.

5 Conclusions and Future Work

We proposed a multi-objective grammar based AP algorithm to accomplish the extraction of IF-THEN rules in multi-class data sets. It follows a novel multi-objective strategy which avoids the overlapping problem when ranking individuals that belong to different classes according to Pareto dominance. It uses a niching approach to combine the solutions of each front found by this method.

Results prove that multi-objective evaluation in AP is more suitable for the classification task than single-objective one, which is used in the original algorithm. They also prove statistically that our proposal outperforms most of the other algorithms regarding predictive accuracy, also obtaining a good trade-off between accuracy and comprehensibility.

For future work, we plan to improve the fitness functions and evaluate the behavior of AP methods when dealing with imbalanced classification.

Acknowledgments. This work has been supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN2008-06681-C06-03, TIN-2011-22408, and FEDER funds.

References

1. Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., Michalkiewicz, E.L.: A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. Artificial Intelligence in Medicine 30, 27–48 (2004)

2. Tan, K.C., Tay, A., Lee, T.H., Heng, C.M.: Mining multiple comprehensible classification rules using genetic programming. In: CEC 2002, pp. 1302–1307 (2002)
3. Abbass, H.A., Hoai, X., Mckay, R.I.: AntTAG: A new method to compose computer programs using colonies of ants. In: IEEE CEC 2002, pp. 1654–1659 (2002)
4. Dorigo, M., Stützle, T.: The Ant Colony Optimization metaheuristic: Algorithms, Applications and Advances. Kluwer Academic Publishers (2002)
5. Olmo, J.L., Romero, J.R., Ventura, S.: Using ant programming guided by grammar for building rule-based classifiers. IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics 41(6), 1585–1599 (2011)
6. Espejo, P., Ventura, S., Herrera, F.: A survey on the application of genetic programming to classification. IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews 40(2), 121–144 (2010)
7. Angus, D., Woodward, C.: Multiple objective ant colony optimisation. Swarm Intelligence 3(1), 69–85 (2009)
8. Dehuri, S., Patnaik, S., Ghosh, A., Mall, R.: Application of elitist multi-objective genetic algorithm for classification rule generation. Appl. Soft Comput. 8, 477–487 (2008)
9. Olmo, J.L., Romero, J.R., Ventura, S.: A grammar based ant programming algorithm for mining classification rules. In: 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 225–232 (2010)
10. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
11. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: IJCAI 1993, pp. 1022–1029 (1993)
12. Parpinelli, R., Freitas, A.A., Lopes, H.S.: Data mining with an ant colony optimization algorithm. IEEE Trans. on Evol. Computation 6, 321–332 (2002)
13. Martens, D., De Backer, M., Vanthienen, J., Snoeck, M., Baesens, B.: Classification with ant colony optimization. IEEE Transactions on Evolutionary Computation 11, 651–665 (2007)
14. Holden, N., Freitas, A.A.: A hybrid PSO/ACO algorithm for discovering classification rules in data mining. J. Artif. Evol. App, 2:1–2:11 (2008)
15. Cano, A., Zafra, A., Ventura, S.: An EP algorithm for learning highly interpretable classifiers. In: ISDA 2011, pp. 325–330 (2011)
16. Cohen, W.: Fast Effective Rule Induction. In: ICML 1995, pp. 115–123 (1995)
17. Frank, E., Witten, I.H.: Generating accurate rule sets without global optimization. In: ICML 1998, pp. 144–151 (1998)
18. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. 7, 1–30 (2006)

Matrix Analysis of Genetic Programming Mutation

Andrew J. Parkes, Ender Özcan, and Matthew R. Hyde

School of Computer Science
The University of Nottingham
Nottingham, NG8 1BB
United Kingdom (UK)
{ajp,exo,mvh}@cs.nott.ac.uk
http://cs.nott.ac.uk/~{ajp,exo,mvh}

Abstract. Heuristic policies for combinatorial optimisation problems can be found by using Genetic programming (GP) to evolve a mathematical function over variables given by the current state of the problem, and whose value is used to determine action choices (such as preferred assignments or branches). If all variables have finite discrete domains, then the expressions can be converted to an equivalent lookup table or 'decision matrix'. Spaces of such matrices often have natural distance metrics (after conversion to a standard form). As a case study, and to support the understanding of GP as a meta-heuristic, we extend previous bin-packing work and compare the distances between matrices from before and after a GP-driven mutation. We find that GP mutations often correspond to large moves within the space of decision matrices. This strengthens evidence that the role of mutations within GP might be somewhat different than their role within Genetic Algorithms.

Keywords: Genetic programming, Genotype-phenotype mapping.

1 Introduction

The effects of the genetic programming (GP) mutation operator are not often analysed in detail. When an analysis is performed, it is often to show how successful different levels of mutation are, or to show the effects of its interaction with crossover. This paper presents a methodology to analyse the effect of mutation on the phenotype of an individual.

In GP, the genotype and phenotype are often indistinguishable, but there are many applications of GP where they are clearly different. One example is when GP is used to generate heuristic functions, which give a score to a number of options at any given decision point (see [1,6,4,8,10,11] for examples on many different problems, including job shop scheduling, cutting/packing, and SAT). This is equivalent to an 'index policy' [12], because each potential option is given a score independently of other options, and the option with the largest score is selected. In the example we present in this paper, the evolved mathematical expressions are used as policies for the online one-dimensional bin packing problem.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 158–169, 2012.
© Springer-Verlag Berlin Heidelberg 2012

In this domain, the choices that are made using the expression can stay the same, even though the expression itself has changed, along with the values it produces for any given inputs. For example, an expression whose results are scaled by 2 gives the same relative scores to each option. In this situation, it is not enough to analyse the effects of genetic operators on only the genotype.

In this paper, we combine these issues with previous work in [16] and present a matrix analysis tool for understanding the effects of mutation on the phenotype. This can be used in any situation where the GP trees represent mathematical expressions with integer variables. The tool is based on the idea that a matrix can be generated from the expression by inputting all possible integer combinations, and storing the results. The resulting matrix will represent the expression exactly, as the only possible inputs are integers. In this paper, we study integer variables, but we expect this to work for at least general discrete and finite cases.

A matrix can be generated for an expression before and after the mutation operator is applied, and the matrices can be compared to analyse what effect the mutation had. We show in this paper that many mutation calls return a different expression, but which corresponds to an equivalent matrix, and so its behaviour as a heuristic is the same. The proportion of such ineffective mutations varies during the run, but can be as high as 45%. When an expression's matrix is different after a mutation, we measure how different, and find that a high proportion of the matrix is often modified by the mutation operator. With further analysis, this research could also provide insight into the code bloat phenomenon, as mutations become less effective as the generations increase. It also can be used as a tool for the analysis of GP runs, to check how the mutation is performing, and modifying its severity accordingly.

2 The Bin Packing Problem

The exact nature of the problem is a secondary concern in this paper, we are interested in analysing the mutation operator of the GP system. However, to do this, we need a problem domain for which to evolve heuristics, and the one dimensional bin packing problem is a highly appropriate domain to test on, given the volume of existing literature on evolving policies for this problem.

The one-dimensional bin-packing problem involves a set of integer-size pieces L, which must be packed into bins of a certain capacity C, using the minimum number of bins possible. In other words, the set of integers must be divided into the smallest number of subsets so that the sum of the sizes of the pieces in a subset does not exceed C [14]. We will assume that all of the bins have the same capacity, and that the pieces are drawn from a uniform distribution.

For this work we consider problem instances where 500 integer sized pieces are uniformly distributed in the range [5,10], and the bin capacity is 20. We use the following notation, UBP(20,5,10,500) to represent this domain. In this paper, the 'on-line' bin packing problem is studied. That is, we do not know in advance how many pieces there are or the size of those pieces. Our system must simply pack the pieces into the bins in the order they arrive, and the pieces cannot be moved once they have been placed in a bin.

3 Previous Work

GP was used to evolve heuristics for online one-dimensional bin packing in [5,6]. In that work, the heuristics were expressions, which provided a score to each available bin. The GP system utilised the $+$, $-$, $*$, and % (protected divide, see [1]) operators, and the three terminals available to the GP were the piece size, the bin fullness, and the bin capacity. The current piece is put into the bin which received the highest score. This work was later extended to reduce the number of inputs to two [7], and to utilise a 'memory' component to learn to use the distribution of piece sizes [3].

Parkes and Özcan noted that for a bin packing problem where the pieces and bins have integer size, the possible inputs to the expression are discrete. Therefore, the expression can be represented by a matrix. They showed that matrices themselves can be evolved with a genetic algorithm [16]. However, in this paper, we employ standard GP to evolve trees (mathematical expressions), and use their matrix representation to analyse the effects of the mutation operator. The matrix representation is explained in detail in section 4.

The key idea of this paper is that when evolving expressions with tree-based GP, a mutation could be made on a part of the tree which represents inactive code. While the tree would look different, the actual results returned by the tree would be the same for any given input values. Furthermore, the values returned by the tree are used to rank the bins by the scores that they receive, so the actual values do not matter. It is only the relative order of the scores that makes a difference. For this reason, a mutation could cause a change in the tree, which does cause a change in the results of the tree, but which *does not* cause a change in the behaviour of the tree when applied to the problem. A simple example of this is a mutation which adds 10 to the value returned by the tree. The tree would be modified, and the values returned would be modified, but the policy that the tree represents would *not* be modified. This is because the tree will always give the same bin the highest score (and the second highest, and so on).

There is previous work on distance metrics for trees, such as Levenshtein edit distance and more [9,2,13]. The previous studies measure the difference between the GP trees, or in other words the genotype. However, often two very different trees can have equivalent functionality. Indicating that in fact, the trees should be measured as similar. This paper suggests an approach to measure the distance between the functionality of the trees, rather than their structural differences. This can show the magnitude of the effect of operators, such as mutation.

4 The Matrix Representation

An example GP expression, with two inputs S and E, is $S + (S/E)$. To choose a bin for a given piece, this tree is evaluated once for each available bin, and the bin with the highest score receives the piece. This system is presented in [5,6], and is also employed in this paper.

Figure 1 shows the matrix which represents $S + (S/E)$. The rows represent the remaining space left in a bin. The columns represent the size of the piece that we

are currently choosing a bin for. The policy matrix contains heuristic values. The values are obtained by evaluating the expression (tree) with the remaining space and piece size as inputs. They show the heuristic score that is given to any bin by the tree, for any given piece size.

The dots in the matrix are positions which will never be used for an instance of UBP(20,5,10,500), as there are no piece sizes less than 5 and greater than 10. No bin can have a remaining space of between 16 and 19 (inclusive), because the minimum piece size is 5. These positions in the matrix are referred to as 'inactive' positions. We can calculate a value for them by evaluating the tree with the relevant inputs, but they will never be used. The other positions are referred to as 'active' positions.

If we add 10 to each of the values in figure 1, then it will make no difference to which bin receives the highest score. For a piece size of 5, a bin with 5 units of remaining space will always be chosen. For this reason, the exact values are not important, it is the relative order of the values that is important. Figure 2 shows a normalised version of the matrix in figure 1, where the bins are still ranked in the same order for a given piece size. In this paper, we are interested in whether the mutation operator makes changes in the normalised matrix, as this determines the behaviour of the heuristic.

E \ S	5	6	7	8	9	10
5	6.00	.	.	.	.	.
6	5.83	7.00	.	.	.	.
7	5.71	6.86	8.00	.	.	.
8	5.63	6.75	7.88	9.00	.	.
9	5.56	6.67	7.78	8.89	10.00	.
10	5.50	6.60	7.70	8.80	9.90	11.00
11	5.45	6.55	7.64	8.73	9.82	10.91
12	5.42	6.50	7.58	8.67	9.75	10.83
13	5.38	6.46	7.54	8.62	9.69	10.77
14	5.36	6.43	7.50	8.57	9.64	10.71
15	5.33	6.40	7.47	8.53	9.60	10.67
16	.	.	.	.	.	.
17	.	.	.	.	.	.
18	.	.	.	.	.	.
19	.	.	.	.	.	.
20	5.25	6.30	7.35	8.40	9.45	10.50

Fig. 1. Value matrix generated from the expresssion: $S + (S/E)$

E \ S	5	6	7	8	9	10
5	12	.	.	.	.	.
6	11	11	.	.	.	.
7	10	10	10	.	.	.
8	9	9	9	9	.	.
9	8	8	8	8	8	.
10	7	7	7	7	7	7
11	6	6	6	6	6	6
12	5	5	5	5	5	5
13	4	4	4	4	4	4
14	3	3	3	3	3	3
15	2	2	2	2	2	2
16	.	.	.	.	.	.
17	.	.	.	.	.	.
18	.	.	.	.	.	.
19	.	.	.	.	.	.
20	1	1	1	1	1	1

Fig. 2. Normalised matrix generated from the matrix in figure 1

To further clarify, the matrixes are used in the following manner. For any given piece size, there will be a column of scores, which correspond to the preference of which bin to put the piece into. For example, in figure 1, if we must pack a piece of size 6 next, then the piece will be put into a bin with an emptiness of 6, as this

emptiness has the highest score (7.00) in that column of the matrix. However, if no available bin has an emptiness of 6, then the piece will be put into the bin with emptiness 7, as this has the second highest score and is therefore the second preference, and so on. One empty bin is always available.

5 Genetic Programming Parameters

In this paper, we analyse 50 runs of a GP system with the parameters shown in table 1. We use the ECJ (Evolutionary Computation in Java) system, which is a mature and well known toolkit for genetic algorithms. We do not make any claims about the quality of these parameters, we chose them because we wish to analyse the mutation operator in the standard ECJ distribution.

Table 1. The GP parameters

Population Size	10000
Generations	100
Crossover Probability	0.85
Reproduction Probability	0.05
Mutation Probability	0.1
Selection Method	Tournament Size 70
Initialisation Method	Ramped half-and-half
Initial Min and Max Tree Depth	2, 6
Max Tree Depth After Mutation	17

For the reader interested in technical implementation details, the mutation analyser was implemented as an extension of the 'MutationPipeline' class, overriding the 'produce' method. The custom produce method converts the old individual and the new individual into their matrix form, and then compares them using the distance metrics described in section 6.

The population size is set to 10000, to ensure that there are a significant number of mutations in each generation. The results do not include those where the mutation fails and the parent is just copied (due to standard ECJ mechanisms). The standard tournament size is set to 7, for the standard ECJ population size of 1024. As we are using a larger population size of 10000, we increase the tournament size to 70.

The mutation operator is subtree mutation. The selection of a node is done probabilistically, with a 0.1 probability of selecting a terminal, and 0.9 probability of selecting a non-terminal node. The subtree is replaced by the 'Grow' method [17].

6 Distance Metrics

To analyse the effects of mutation on the policies, we must define a distance metric to measure the distance between two matrices. In this paper we employ three distance metrics, all of which operate on the normalised versions of the matrices. Only

the active parts (see Section 4) of the matrices are included in the distance calculations. To illustrate the three metrics, we use the simple example of UBP(11, 4, 5, 500). With bins of 11 capacity, and pieces between 4 and 5 inclusive (For the results section, our experiments are performed on instances of UBP(20, 5, 10, 500)). Two policy matrices for this problem are shown in figures 3 and 4. For each column, each heuristic value (one per emptiness value) represents a preference of where to put the piece. The highest value represents the first preference, the second highest represents the second preference, and so on.

E \ S	4	5
4	2	.
5	4	4
6	1	3
7	5	2
8	.	.
9	.	.
10	.	.
11	3	1

Fig. 3. Example Matrix A

E \ S	4	5
4	3	.
5	4	1
6	1	2
7	5	3
8	.	.
9	.	.
10	.	.
11	2	4

Fig. 4. Example Matrix B

6.1 Metric 1

This metric simply counts the number of preferences that are different in the normalised matrices. In the example of figures 3 and 4, this metric would give the two matrices a difference value of 6. For the problem instances we address in this paper, the matrices are larger, and so a score of 57 means the two matrices are completely different. A score of 0 means that the two matrices are identical.

6.2 Metric 2

Recall that for any given piece size, there will be a column of heuristic values, which correspond to the preference of which available residual space to put the piece into. For example, consider the first columns from both matrices in figures 3-4. These columns represent the preference order for the piece size 4. We show the columns here as rows (excluding the inactive positions):

Column 1 of matrix 1: 2, 4, 1, 5, 3
Column 1 of matrix 2: 3, 4, 1, 5, 2

We see that for both columns, the preference is to put the piece into a bin with a residual capacity of 7, as this has the highest value (5). For both columns, if a bin does not exist with 7 units of space left, the second choice of bin would have a residual capacity of 5. We can write the preference order of residual capacities like this:

Residual capacity preference order 1: 7, 5, 11, 4, 6
Residual capacity preference order 2: 7, 5, 4, 11, 6

To calculate metric 2 we iterate through each preference order list. We add one point of similarity if both matrices have the same first preference. Then we move to the second preference and add one point if that is the same, and so on. For each column, we stop when the current preference is different, and move to the next column. In the example, 7 and 5 are ranked in the same order in both lists, and the next entries are different, so these columns have a similarity of 2. The columns for the piece size 5 have no similarity, as the first preferences are different (residual capacity 5 in the first matrix, and 11 in the second matrix). Therefore, according to metric 2, these matrices have a similarity of 2.

In the problem instances we use in this paper, the maximum score is 57, as there are 57 active positions to compare. We subtract the result from 57 to put the metric on the same scale as metric 1. Therefore, a score of 57 means that the two matrices have a different first choice for every piece. A score of 0 means that the two matrices are identical.

6.3 Metric 3

This is an ordering based metric, which involves comparing the columns of the matrices in a similar way to metric 2. While metric 2 asks how many elements of the preference order are the same (until the first difference), metric 3 asks how many of the elements that follow each preference are the same, regardless of their order. We calculate a preference order for each column, the same as we calculated for metric 2. This is shown again here for the first column, for ease of reference.

Residual capacity preference order 1: 7, 5, 11, 4, 6
Residual capacity preference order 2: 7, 5, 4, 11, 6

We iterate through each preference from order 1, and find the identical preference in order 2. For every value which follows that preference in both orders, we add one point of similarity. In our example:

7 precedes 5, 11, 4 and 6 in both lists (similarity of 4).
5 precedes 11, 4, and 6 in both lists (similarity of 3).
11 precedes 6 in both lists (similarity of 1).
4 precedes 6 in both lists (similarity of 1).

The total similarity for the first column is therefore 9 (4+3+1+1). The similarity of a column is subtracted from n(n-1)/2, as this is the maximum similarity score, where n is the length of a column. We perform the same calculation for the other columns. In the instances used in this paper, Metric 3 has a minimum value of 0, representing identical matrices, and a maximum value of 251.

This metric considers each preference order as a permutation, and tells us how many swaps would be needed between any two adjacent preferences, to get from one permutation to the other.

7 Results

This section presents our results, showing the effect of the mutation operator on the policy matrices, over the 100 generations. The calculations are based on 50 runs of the GP algorithm. For example, the results for generation 6 are calculated from all 50 sets of recorded values at generation 6.

In figure 5, first consider the dotted line, which represents the 'metric 1' difference between the matrices after mutation. In each generation, all of the mutations are measured with metric 1, and the average difference is calculated. A higher value means that the mutation operator has a larger effect on the policies. The plot shows that in the first 3 generations, the effect of mutation increases. After generation 3, the effect of mutation decreases gradually.

The second line in figure 5 shows the same calculation, but excluding the mutations that cause absolutely no change to the normalised matrix. One can see that when the mutation *does* make a change to the policy, that change is generally greater in the first few generations. That change decreases until generation 13, after which the change caused by the mutation operator gradually increases.

Figure 6 shows the proportion of mutations which do not change the normalised matrix at all, and therefore do not change the behaviour of the heuristic. This plot shows that in the first generation, around 45% of mutations have no effect on the individuals. This drops by a large amount in generation 2, to around 19%. This shows that the effect of mutation changes dramatically in the first 2 generations, as the population changes from a randomly generated one, to one made from the parents of the first generation.

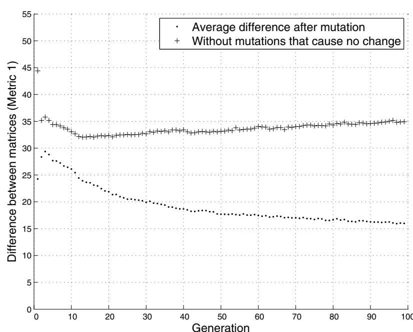


Fig. 5. Effect of mutation as measured by metric 1. The two plots show the average effects, and those not including the mutations that cause no change.

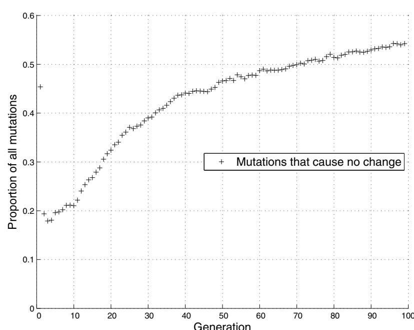


Fig. 6. Proportion of mutations per generation that cause no change in the normalised matrix, and therefore cause no change in the packing policy

Code bloat could be the cause of the downward trend in the effect of mutation. As code bloat increases the proportion of the tree which has no effect (See 'removal bias' theory [18] and 'replication accuracy' theory [15]), so the mutation operator is more likely to mutate a subtree which has no effect anyway. Of the mutations that do have an effect, the downward trend in the first 13 generations, followed by the gradual increase, is an effect which requires further research. We suspect that it involves the convergence of the population. For the one-dimensional bin packing problem, it could be the case that the population has generally converged to a good solution by around generation 13. If this is the case, then the results show that the mutation operator makes smaller changes as the population is improving. We would argue that this is a desirable quality, as it will make incremental changes to the better policies in later generations, while not changing the core functionality of the policy. Once the GP algorithm is generally improving the best individual less frequently, and the population is more stagnant, code bloat becomes a larger factor.

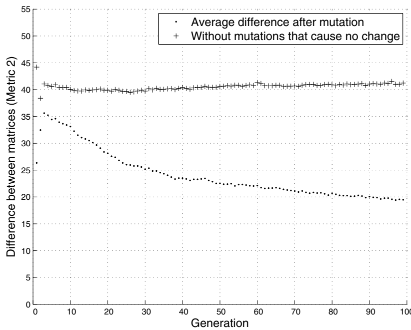


Fig. 7. Effect of mutation as measured by metric 2. The two plots show the average effects, and those not including the mutations that cause no change.

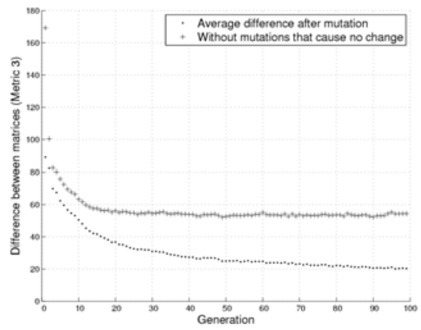


Fig. 8. Effect of mutation as measured by metric 3. The two plots show the average effects, and those not including the mutations that cause no change.

Figure 7 shows the results calculated with metric 2, which follow the same pattern, but the difference is measured as larger than metric 1. This could suggest that the mutation operator often modifies the first few choices for each piece size, but not the lower choices (for example the 10th and 11th choice of bin).

Figure 8 shows the results calculated by metric 3. Recall that this metric considers how many preferences that follow each preference are the same, regardless of the order of the preferences. It tells us how many swaps would be needed to get from one preference permutation to the other. This metric suggests that, in later generations, the mutation operator changes less the relative preference order. In the early generations, the mutation operators make very large changes in the relative ordering of the bin preferences. The effect measured by metric 3 consistently decreases throughout the run, unlike the other two metrics.

When a mutation swaps two preferences that are far apart in the preference order, metric 3 measures a larger change. For example, if the 1st and 6th preferences swap, then metric 3 measures a larger change than if the 1st and 2nd preferences swap. In contrast, metric 1 measures the same change for both of those examples, as only two preferences are different. From the results of metric 3, we can infer that the changes later in the run are more localised, and that there is no significant further reduction in the severity of mutation after generation 15.

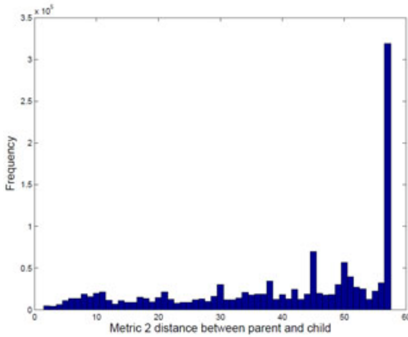


Fig. 9. Histogram of metric 2 effect of mutation

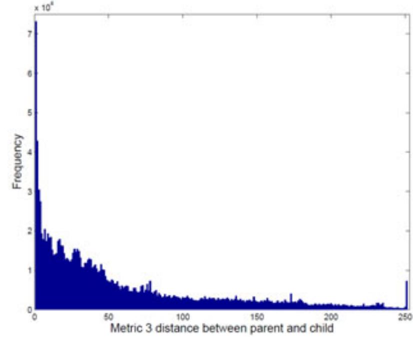


Fig. 10. Histogram of metric 3 effect of mutation

Figures 5-8 show the results per generation, but it is also interesting to consider the distribution of the mutation effects, rather than just the mean averages. Figures 9 and 10 show histograms of all the mutations from all generations in all 50 runs, not including the mutations which cause no change. Figure 9 shows the mutations measured with metric 2, and figure 10 shows the same mutations measured with metric 3. It is interesting to note that metric 2 measures the changes as mostly high, and metric 3 measures them as mostly low. From metric 2, we can say that the changes in the preference order mostly appear early in the preference order, which is the most influential part. By far, the most common difference value is 57, which represents a mutation which causes a change in the policy where, for each piece, the first choice of bin will be modified. Even though the first choices often change, figure 10 shows that in general the mutations represent just a few swaps of adjacent preferences. The difference between figures 9 and 10 show that the magnitude of the mutation depends upon which metric is used to measure it.

8 Conclusions

We have presented a method for analysing the effects of the GP mutation operator in a normal GP run. It is applicable whenever the individual is a mathematical expression, with integer variables. Because of the integer input, the expression can be represented as a matrix of values, representing the value returned for each

possible combination of inputs. In this study, the individuals were tree-based expressions which acted as index policies for the online bin packing problem. They gave a heuristic score to each bin, and the highest scoring bin received the next piece. The individuals therefore acted as policies for the handling of pieces when they arrive, depending on the bins available at the time.

The results show that the mutation operator causes large changes in the behaviour of the policies. It is generally not an operator that makes small changes to improve a policy incrementally. Over the course of the run, the effect of the mutation operator on the phenotype changes. Its effect was larger at the beginning of the run, and reduces as the run progresses. These results seem to fit with the 'replication accuracy' theory of code bloat [15], which states that fit individuals which are large enough to not be affected by mutation are more likely to survive in the population, as their behaviour is unchanged from one generation to the next. Further research could confirm this. We also plan to use the matrix analysis tool to analyse the effects of the crossover operator on the individuals.

There are many areas where this type of analysis tool could be used. For example, it could be used during a GP run to vary the severity of mutation depending on the effect it is having on the population. It could be used to assess the effectiveness of bloat control methods, which usually operate simply on the size of the trees. The analysis method presented here offers the chance to measure bloat by the effects of the mutation operator. Once developed further, this technique could be a valuable tool for effective parameter setting of the genetic operators in GP.

References

1. Allen, S., Burke, E.K., Hyde, M.R., Kendall, G.: Evolving reusable 3D packing heuristics with genetic programming. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO 2009), Montreal, Canada, pp. 931–938 (July 2009)
2. Burke, E.K., Gustafson, S., Kendall, G.: Diversity in genetic programming: an analysis of measures and correlation with fitness. IEEE Transactions on Evolutionary Computation 8(1), 47–62 (2004)
3. Burke, E.K., Hyde, M.R., Kendall, G.: Providing a memory mechanism to enhance the evolutionary design of heuristics. In: Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2010), Spain, pp. 3883–3890 (July 2010)
4. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.: Exploring Hyper-heuristic Methodologies with Genetic Programming. In: Mumford, C.L., Jain, L.C. (eds.) Computational Intelligence. ISRL, vol. 1, pp. 177–201. Springer, Heidelberg (2009)
5. Burke, E.K., Hyde, M.R., Kendall, G.: Evolving Bin Packing Heuristics with Genetic Programming. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 860–869. Springer, Heidelberg (2006)
6. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In: Proceedings of the 9th ACM Genetic and Evolutionary Computation Conference (GECCO 2007), London, UK, pp. 1559–1565 (July 2007)

7. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: The scalability of evolved on line bin packing heuristics. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), Singapore, pp. 2530–2537 (September 2007)
8. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. IEEE Transactions on Evolutionary Computation 14(6), 942–958 (2010)
9. Ekárt, A., Németh, S.Z.: A Metric for Genetic Programs and Fitness Sharing. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 259–270. Springer, Heidelberg (2000)
10. Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation 16(1), 31–61 (2008)
11. Geiger, C.D., Uzsoy, R., Aytug, H.: Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. Journal of Scheduling 9(1), 7–34 (2006)
12. Gittins, J.C.: Bandit processes and dynamic allocation indices. Journal of the Royal Statistical Society. Series B (Methodological) 41(2), 148–177 (1979)
13. Gustafson, S., Vanneschi, L.: Crossover-based tree distance in genetic programming. IEEE Transactions on Evolutionary Computation 12(4), 506–524 (2008)
14. Martello, S., Toth, P.: Lower bounds and reduction procedures for the bin packing problem. Discrete Applied Mathematics 28(1), 59–70 (1990)
15. McPhee, N.F., Miller, J.D.: Accurate replication in genetic programming. In: Eshelman, L. (ed.) Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA 1995), July 15-19, pp. 303–309. Morgan Kaufmann, Pittsburgh (1995)
16. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 2011–2018. ACM, New York (2011)
17. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. lulu.com, freely available at (2008), http://www.gp-field-guide.org.uk
18. Soule, T., Foster, J.A.: Removal bias: a new cause of code growth in tree based evolutionary programming. In: 1998 IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, USA, May 5-9, pp. 781–786 (1998)

An Ecological Approach to Measuring Locality in Linear Genotype to Phenotype Maps

Tom Seaton, Julian F. Miller, and Tim Clarke

Department of Electronics
University of York, YO10 5DD
{tas507,jfm7,tc2}@ohm.york.ac.uk

Abstract. Recent research has considered the role of locality in GP representations. We use a modified statistical technique drawn from numerical ecology, the Mantel test, to measure the locality of integer-encoded GP. Weak locality is identified in a case study on Cartesian Genetic Programming (CGP), a directed acyclic graph representation. A method of varying syntactic program locality continuously through the application of a biased mutation operator is demonstrated. The impact of varying locality under the new measure is assessed over a randomly generated set of polynomial symbolic regression problems. We observe that enforcing higher levels of locality in CGP is associated with poorer performance on the problem set and discuss implications in the context of existing models of GP genotype-phenotype maps.

Keywords: Cartesian genetic programming, Locality.

1 Introduction

The notion of locality is a well-established property of representations in genetic algorithms, known to have an impact on search performance [1–4]. Locality describes the design heuristic that small changes to a genotype, due to evolutionary operators, should lead to correspondingly small changes in phenotype. The concept can also be related to the assertion that the genotype to phenotype map (GPM) should support a strong causal relationship between the evolving data structure and its decoded expression [5]. Recently, work in genetic programming has focused on extending the original concept of locality from binary strings to standard, GP tree-based representations [6].

This paper considers the design of a novel method of measuring locality, with the aim of assessing indirect, integer-encoded GPM. Our goal is to establish a statistical approach which can then be applied to linear genotypes [7], encompassing methods such as Cartesian Genetic Programming (CGP) [8], Grammatical Evolution [9, 10] or Linear GP [11]. Characteristically, these GP encodings feature an intermediate level of mapping between genotype and fitness evaluation not traditionally incorporated in tree-based GP. The method described here adopts a long standing technique from the field of numerical ecology, the Mantel test [12–18]. The purpose of the test is to provide a means of rigorously determining the significance of measured correlations between distance matrices.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 170–181, 2012.
© Springer-Verlag Berlin Heidelberg 2012

In Section 2, we briefly review related work on measuring locality in the GA and GP literature and comment on previous definitions. Section 3 provides necessary background on the Mantel statistic and introduces an extension to enable the technique to address correlations in genotype-phenotype maps. Section 4 addresses selection of metrics on the genotype and phenotype space. Section 5 presents a set of preliminary experimental outcomes and analyses in a case study on standard Cartesian GP, over a randomised set of symbolic regression problems. We then discuss these initial results and conclude.

2 Related Work

The early work of Rothlauf is generally cited as the seminal work on locality in the study of representations [1]. Rothlauf proposed aggregating the degree of change in phenotype over the local neighbourhood of each genotype, defined with respect to the particular variational operators:

$$L = \sum_{g \in G} \sum_{p' \in adj^*(g)} d_P(p, p') - d_P^- \tag{1}$$

where $L \geq 0.0$ is the level of locality ($L = 0.0$ is maximal). In our notation we use G, d_G, P and d_P to denote the genotype space, phenotype space and respective distance metrics over each. Equation 1 measures the locality of a map $S : G \rightarrow P$ over all neighbouring pairs of genotypes in genotype space, where $adj^*(g)$ denotes the set of phenotypes which correspond to the adjacent neighbours of g in G. Distances are summed up under the phenotype metric, relative to the minimum distance in phenotype space, which we denote d_P^-. Thus, in a high locality map under Rothlauf's definition, L tends to 0 and genotypes which are neighbours in genotype space also have phenotypes which are similar under the metric applied in that space. The original expression is not normalised with respect to the size of the search space - more recently, extensions to Rothlauf's work on locality in GA for binary strings were proposed by Chiam et. al [19].

Studies of locality in GP [6, 20, 21], by contrast, have considered locality as a direct property of the mapping between genotype and corresponding fitness value. Galvan-Lopez et. al. considered a set of three definitions of locality, derived from Rothlauf's work, and systematically examined each over a set of standard GP benchmarks [6]. This approach would seem appropriate for classical tree GP, where there is no explicit intermediate state between genotype and fitness. However, for indirect GP maps which feature distinct phenotypes, we argue that an understanding of locality should also be sought at the intermediate level. Furthermore, current measures of locality by definition do not consider any relationships at genotype distances beyond the immediate neighbourhood. The method presented in this paper explores an alternative to these aggregative approaches and studies directly the degree of correlation between genotype distances and phenotype distances. There exists some commonality with the method of fitness distance correlation (FDC) extensively addressed in both the GA and GP literature [22]. However, fitness distance correlation develops a measure of problem

difficulty, by considering distances to the generally unknown optimum in fitness space. We are instead preoccupied with understanding locality at the syntactic level - the changes that are introduced into program structure by variations in integer genotype. This enables a problem-independent view, focusing on analysis of the representation and search operators, rather than addressing performance against a particular fitness landscape.

3 The Mantel Test

The Mantel Test is a general, non-parametric statistical resampling technique used in the exploration of correlations between two triangular distance matrices [12] Historically, the test was designed to address the analysis of spatial and temporal data from disease clustering. It has seen considerable application in numerical ecology [14–17] and on genetic and linguistic data [18]. In a mathematical sense, the Mantel test provides a permutation-based method of determining the statistical significance of linear or monotonic relationships. The test is applicable in situations when we wish to determine whether a correlation exists in the distances between elements sampled between two metric spaces. Note in particular that it is not appropriate to use standard significance tests because distances derived from the same element cannot generally be considered independent of each other [18]. The technique is applied between two square distance matrices of size n, labelled $\mathbf{X}$ and $\mathbf{Y}$. The matrices contain the pair-wise differences calculated between all elements of a sample under two measures of distance d_X and d_Y. By way of illustration, in the ecological context $\mathbf{X}$ might represent the geographical distances between samples of a species and $\mathbf{Y}$ corresponding measured genetic distances. Differences are assumed to adhere to the symmetry property of a metric, so both matrices are symmetric with zeros along the diagonal. The original, 'standardised mantel statistic' is then given by the expression [23]

$$r_M = \frac{1}{s-1} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(\frac{X_{i,j} - \bar{X}}{\sigma_X} \right) \left(\frac{Y_{i,j} - \bar{Y}}{\sigma_Y} \right) \tag{2}$$

where r_M is the linear correlation coefficient obtained, $\bar{X}$, $\bar{Y}$ and σ_X, σ_Y are the mean and standard deviation calculated for $\mathbf{X}$ and $\mathbf{Y}$ respectively and $s = n(n-1)/2$. This is equivalent to calculating the Pearson-product moment (linear correlation) over the upper-half of the matrix.

3.1 Significance Testing on Genotype-Phenotype Maps

For r_M to be a useful statistic under sampling, significance testing should be carried out against the null hypothesis, H_0 that the distances in $\mathbf{X}$ and $\mathbf{Y}$ are uncorrelated. A key realisation of the Mantel test is that rows and columns of the matrix are exchangeable under the null hypothesis. That is, we expect to be able to freely rearrange the labels of each set of distances. By permuting the rows (and corresponding columns) of $\mathbf{X}$ and recalculating r_M, a permutation

distribution can be constructed from which the significance of correlations in the unpermuted data is obtained. Given that the null hypothesis is true, we would expect that the unpermuted data should lie somewhere in the center of this range. The test proceeds by obtaining the original unpermuted coefficient r_M^0 and a set of coefficients under permutation of $\mathbf{X}$, denoted $r = \{r_M^1 ... r_M^N\}$, where N is the total number of permutations. Let $\chi \subseteq r$ such that $x \in \chi \geq r_M^0$. The probability of accepting the null hypothesis in the presence of an apparent positive correlation is then given by the one sided test

$$P(H_0|r) \approx \frac{||\chi||}{N} \tag{3}$$

that is the number of instances in which the recalculated coefficient equals or exceeds r_M^0, divided by the total number of permutations. A similar test can be carried out for the case of negative correlation.The test does not necessarily have to support a linear model: it may be appropriate to compute r_M using an alternative statistic, such as Spearman rank-based correlation, using the permutation test in exactly the same fashion. The result converges monotonically on the true significance at large N. In practice, the number of permutations recommended in the literature varies, but a value in the range of 1000-10000 permutations is typically suggested [23].[1]

For the application of the Mantel statistic to artificial GPM, a method is required to calculate it over particular distance intervals. This is to establish whether a correlation exists only for closer, or more distant, genotypes. A similar situation arises in numerical ecology, where correlations may be limited by time, or by geographic distance. Previous derived techniques of the Mantel statistic have consider correlation as a function of range, such as the 'Mantel correlogram', which applies a model matrix to examine correlations over particular distance classes [14]. We adopt a simplified approach, explicitly sub-dividing the distance matrix. Let $\mathbf{X_U}$ be the upper triangle of $\mathbf{X}$. A set of distance classes are selected such that each distance class $\mathbf{D}_{p,q}$ is a subset of the elements of $\mathbf{X_U}$ where $p \leq X_{i,j} < q$. Hence, a distance class contains the elements over which r_M is computed which fall within the range (p, q). The corresponding set of distances at the same index positions in $\mathbf{Y_U}$ are also found. The coefficient r_M is calculated seperately for each distance class and significance values derived as before, by permuting the original matrix and recomputing r_M over that interval.

4 Distance Metrics under the Mantel Statistic

To derive distances between genotypes and phenotypes, an appropriate metric must be selected for each space. Which metrics are suitable is informed by the choice of representation and variation operators. For this initial analysis, we neglect crossover and focus on the mutation operator. Numerous proposals have been put forward for appropriate metric distance measures in GA/GP genotype spaces, see for example the review in [25]. These have included classes such as:

[1] Standard methods for carrying out permutation tests are supported in numerical ecology statistical packages such as `ecodist` and `vegan` [24], in *R*.

M1. *Edit distances* (e.g. Hamming, Levenstein in the case of strings and tree or graph edit distances respectively. [6, 26])
M2. *Subtree distance* (e.g. Tree Alignment [27], Keijzer distance. [28])
M3. *Information compression* (e.g. Normalised Compression Distance. [6])
M4. *Probabilistic measures* (e.g. Subtree Crossover Operator. [29])

Although it is typically practical to define strict metrics for the genotype space under the assumptions made in M1, M2 and M3, those measures based directly on probabilities usually violate one or more of the metric criteria[2]. Calculation of the Mantel statistic only requires that measures adhere to the symmetry criterion [23]. We are therefore in principle free to select from amongst each of the above classes of metric. As noted in [6], there is no priori knowledge of which distance measures are most appropriate to describe differences on program spaces. The approach described here chooses distance measures of types (3) and (4). Our justification for this is one of pragmatism: to avoid being tied to representation specific measures in our analysis and because of the potential complexity of computing edit distances on large graphs (the graph edit distance problem is NP-hard in general). A convenient measure for integer genotypes is the expected number of independent attempts that would be required to generate one genotype from another through a single mutation. Assuming that the two genotypes are mutually reachable (Definition 1), then this is just the inverse of the probability of mutating between both genotypes. We refer to this semimetric as the expected variation distance $\bar{M}$. The measure has the advantage that it defines distance based on the actual transition probability.

Definition 1 (Mutually reachable genotypes). *A pair of genotypes (g, g') are mutually reachable under some variation operator V, given that the probability of deriving g' in a single operation $V(g)$ is greater than zero.*

Definition 2 (Expected variation distance). *A function $\bar{M} : (g, g', V) \to \mathbb{Z}$ on a pair of mutually reachable genotypes (g, g') where $\bar{M}$ gives the expected number of independent single operations on g such that there is an instance $V(g) = g'$. If $g = g'$, we define $\bar{M}(g, g, V) \equiv 0$.*

Derivation for CGP with Uniform Mutation Operator. We can illustrate this approach by calculating the expected variation distance between a pair of standard feed-forward CGP genotypes g and g'. Genotypes are assumed to be equal sized integer strings of length n, which represent a single row with feedforward connections (see [8] for details), where output is derived from the right-most node. Assume there are x matching integers between g and g' and $n - x = y$ different values. The genotype is split into integers corresponding to connections and functions. From the y different integer values, we have a subset of size y_F different values corresponding to functions and y_C values corresponding

[2] A metric function d on metric space Q satisfies: 1. $d(x, y) \geq 0$; 2. $d(x, y) = 0$ iff $x = y$; 3. $d(x, y) = d(y, x)$; 4. $d(x, y) \geq d(x, z) + d(z, y)$ where $x, y, z \in Q$. We adopt the conventional term semimetric when the triangle-inequality is relaxed.

Table 1. CGP Search Parameters

Representation	CGP	Crossover	No
Nodes	10	Selection Strategy	$(\mu + \lambda) = (4 + 6)$
Structure	Single row feed-forward	Population Size	10
Function Set	$\{+,-,*,\div\}$	Fitness Samples	$10 \in \{-2:2\}$
Terminal Set	$\{0,1\}$	Max Generations	2000
Mutation Rate	0.15	Runs	500

to connections. Assume a mutation operator acts on all values with uniform probability m, where a mutation changes the allele to any other feasible integer. Then, the probability of x values remaining the same is $(1-m)^x$. The probability of y_F values from g mutating to the same function as that in g' is $(\frac{m}{F-1})^{y_F}$, where F is the number of possible function choices. Let $\mathbf{y_C}$ be the set of integer values which contribute to y_C. Each connection $i \in \mathbf{y_C}$ has $c_i - 1$ possible alternatives (where c_i is in general the total number of inputs, plus all previous nodes). Thus the probability of obtaining the same set of connections is $m^{y_C} \prod_{i \in \mathbf{y_C}} \frac{1}{c_i - 1}$. The total probability u of mutating from one CGP genotype to another is therefore

$$u(g, g') = (1-m)^x \cdot \left(\frac{m}{F-1}\right)^{y_F} \cdot m^{y_C} \prod_{i \in \mathbf{y_C}} \frac{1}{c_i - 1} \tag{4}$$

Taking the inverse and collecting terms gives the expected number of independent mutations required, $\bar{M} = \frac{1}{u}$:

$$\bar{M} = \frac{(F-1)^{y_F}}{m^y \, (1-m)^x} \prod_{i \in \mathbf{y_C}} c_i - 1 \tag{5}$$

Distances between genotypes under uniform mutation can be computed in other integer representations such as grammar GP in a similar fashion.

5 Experiment

To test our approach, 50 biarity tree samples were obtained from a representative CGP genotype space using an arithmetic function set (including the protected division operator). Table 1 summarises the parameters used to initialise each genotype. Sample biarity trees were produced recursively under the uniform mutation operator to a depth of 7 mutations, generating 511 genotypes per sample.[3] The expected variation distance $\bar{M}$ was obtained pair-wise for all members of each sample. The approach provided a set of 50 corresponding matrices each containing ∼150000 genotype distances. CGP phenotypes are directed acyclic graphs. Measurements of the syntactic distance between CGP phenotypes (d_P) were derived using the Normalised Compression Distance (NCD) (Equation 6) adapting the procedure of [6] for tree GP, such that

[3] Conceptually, this is a method of biased sampling similar to chain-referral sampling, adopted extensively in sociological research [30]. Sampling via the mutation operator generates trees which partially span the local neighbourhood for each genotype.

$$d_P(p,p') = \frac{C(pp') - min(C(p), C(p'))}{max(C(p), C(p')} \tag{6}$$

where C is a function giving the length in bits of the string representation, under UTF8, of the argument for a particular compressor. Each phenotype was decoded into the prefix string representing the corresponding encoded arithmetic expression. This expression excludes neutral nodes (junk) which do not contribute to the phenotype. Pair-wise application of the NCD gives a measure of similarity between phenotypes in the range of $\{0.0 : 1.0\} + \epsilon$, using the `gzip` algorithm (where $\epsilon \approx 0.1$, an error term induced because the compression is not ideal). To provide a controllable method of exploring the impact of changing locality in a representation, an intermediary bias $u_{\alpha\beta}$ was introduced to the uniform mutation operator:

$$u_{\alpha\beta}(p,p') = \left(1 + e^{-\alpha(d_P(p,p')-\beta)}\right)^{-1} \tag{7}$$

The bias $u_{\alpha\beta}$ was employed to change the expected mutation distance between each pair of genotypes and is a standard sigmoid function, adjusted by a scaling parameter α and translation β respectively. For each application of the mutation operator to a genotype, $u_{\alpha\beta}$ defined the probability that a proposed set of mutations will be accepted. The process is repeated until an acceptable mutation is found and returned by the operator. To first order, this gives an adjusted expected variation distance, where

$$\bar{M}_{\alpha\beta}(p,p')^{-1} \approx u_{\alpha\beta}(p,p') \times u(p,p') \tag{8}$$

Hence by scaling α, the likelihood that mutations will result in phenotypes which are syntactically similar can be defined. Varying the mutation bias equates to scaling the locality of the mapping. The threshold value assumed in the sigmoid function is set to an intermediate level of similarity, $\beta = 0.2$. An example (for one CGP sample) is given in Figure 1. The graphs are scatterplots, binned into hexagons, illustrating qualitatively the distribution observed between the log-scaled expected variation distance $\bar{M}$ and normalised compression distance d_P. The result is shown between two similar maps at low locality ($\alpha = 20, 10$) and at high locality ($\alpha = -10, -20$). This directly compares the change in locality induced by the bias. Inspecting the scatter graphs appears to indicate a weakly positive trend, apparent over short distances. This follows from the decreasing likelihood of making larger syntactic changes to the phenotype, under the uniform mutation operator. Relatively probable mutations, $\bar{M} \sim$ [0 - 20] correspond to smaller changes in compression distance, $d_P \sim [0.1 - 0.3]$. The majority of distances observed in the region $\bar{M} \sim$ [20 - 40] (between genotypes situated on lower branches of the sample) occur with lower probability and correspond to greater variation in syntactic change.

Using the Mantel test, we can validate these qualitative observations. Figure 2 shows the range of corresponding Mantel coefficients r_M calculated over all samples, for linear correlation, as a function of distance. It can be inferred that an overall weak positive correlation exists in the CGP mapping, which falls off as

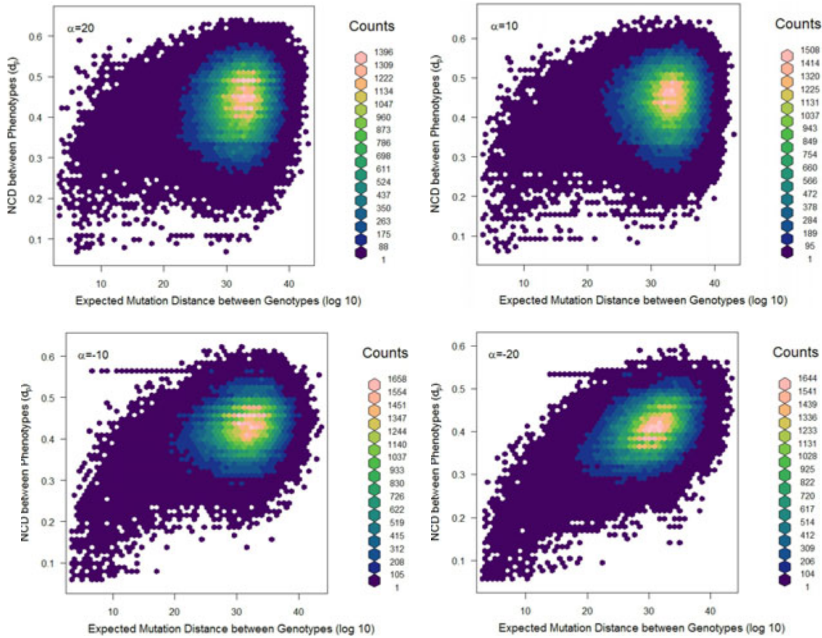


Fig. 1. Illustration of the effect of the NCD mutation bias on genotype-phenotype correlation. Top left: Lowest locality. Bottom right: Highest locality.

a function of genotype distance. A set of 1000 permutations was then generated for each distance matrix to test significance at $P(H_0) < 0.005$, for a set of 8 distance classes from $\bar{M} = 0.0 : 40.0$. The correlations found to be significant under permutation are labelled (*). Inclusion of the Mantel test therefore gives a firm basis from which to reject the null hypothesis and accept the correlation. The effects of the mutation bias are also apparent (contrast positive α with negative α).

To explore the relationship between syntactic locality and performance, a preliminary experiment was carried out using a randomly generated selection of 38 symbolic regression problems. The problem instances were restricted in complexity to simple 5th order polynomials with integer coefficients in the range of {-2:2}. These are basic problems known to be solvable consistently using only the simple CGP representation analysed, without the requirement for additional features such as modularity. Five instances of each problem were considered, applying the mutation bias with $\alpha = \{-20, -10, 0, 10, 20\}$ and $\beta = 0.2$. Fitness was evaluated by deriving the euclidean distance over the set of uniformly distributed sample points. Other parameters (Table 1) were informed per common previous estimates in CGP [8]. The parameters have not been optimised to account for interaction with the mutation bias. Table 2 shows the corresponding probability of success η at each locality level after 2000 generations, estimated in each instance from the fraction of 500 runs which successfully recovered the expression. For the 21 polynomial problems with an average success

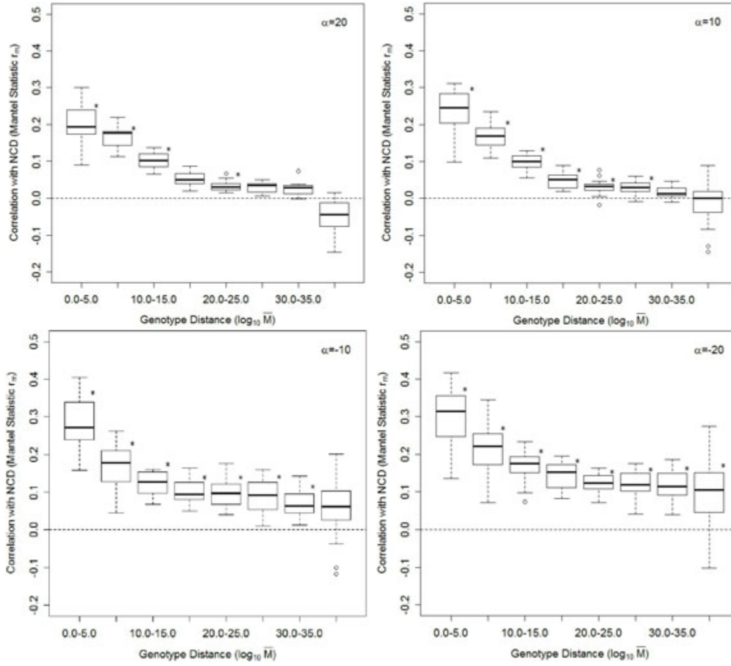


Fig. 2. Measured Mantel correlation for CGP with different levels of mutation bias. Top left: Lowest locality. Bottom right: Highest locality.

probability $\bar{\eta}$ greater than 10% (denoted with a †), a general tendancy can be observed towards better performance at lower levels of locality. Of this subset, in 19 of the 20 cases the probability of success was higher for $\alpha = 20$ than $\alpha = -20$. In the remaining cases with success probability below 10%, no measurable trend is observable outside of experimental error. All problems were solved successfully over at least one set of runs.

6 Discussion

The weak correlation observed between genotype and phenotype distances in Figures 1 and 2 is consistent with the variation in structure that small mutations can impose in this representation. Altering a single node connection in CGP may cause a large number of functions to be disconnected. Similarly, if the same node is connected to many neighbours, then adjusting it will produce a disproportionate change to the syntax of a program. There is therefore an overall tendancy for parents to produce syntactically similar offspring, but this is offset by the potential for large structural change. The relatively small impact of the mutation bias also suggests this relationship is difficult to suppress, given that it is a direct consequence of utilising a graph-based structure.

Table 2. CGP success probability η with respect to locality

Polynomial Expression	α					$\bar{\eta} \geq 0.1$?
	-20	-10	0	10	20	
$-1 - 2x^3 + x^4$	0.236	0.268	0.286	0.304	0.288	†
$-2 - 2x - x^2 - x^4$	0.102	0.116	0.158	0.142	0.174	†
$-2 - x^3$	0.254	0.314	0.386	0.404	0.450	†
$-2x^2 + 2x^4 + 2x^5$	0.152	0.196	0.226	0.196	0.244	†
$-2x^2 - 2x^3$	0.556	0.570	0.594	0.616	0.676	†
$-2x^3 + x^4$	0.646	0.758	0.792	0.814	0.790	†
$-2x^3 + x^5$	0.196	0.278	0.350	0.360	0.394	†
$-x^2 + x^3 + 2x^4$	0.684	0.722	0.790	0.840	0.866	†
$-x - x^4$	0.626	0.744	0.792	0.840	0.868	†
$1 - 2x^2 - 2x^3$	0.556	0.570	0.594	0.616	0.676	†
$1 - x + 2x^3 + x^4$	0.290	0.324	0.370	0.322	0.326	†
$1 - x + x^2 - x^3$	0.766	0.840	0.878	0.894	0.924	†
$1 + x^3 + 2x^5$	0.170	0.214	0.198	0.198	0.216	†
$2 + 2x - x^2 - 2x^3$	0.228	0.226	0.260	0.248	0.272	†
$2 + x^2 - x^3$	0.422	0.486	0.516	0.516	0.572	†
$2x^2 - x^3 + 2x^4$	0.382	0.446	0.492	0.524	0.522	†
$2x^2 - x^3 + x^4 - x^5$	0.198	0.256	0.232	0.290	0.282	†
$2x^2 + x^3 - 2x^4 - x^5$	0.504	0.592	0.540	0.536	0.524	†
$2x + x^2 - 2x^3$	0.202	0.216	0.206	0.230	0.276	†
$x + 2x^2 - 2x^3$	0.610	0.672	0.642	0.670	0.684	†
$-2 - 2x^2 - 2x^5$	0.034	0.042	0.038	0.014	0.028	-
$-x + 2x + x^4 - 2x^5$	0.122	0.122	0.112	0.110	0.116	†
$-2 + 2x - x^2 - 2x^3 + 2x^5$	0.002	0.006	0.000	0.000	0.000	-
$-2 + x - x^2 - 2x^4 - 2x^5$	0.001	0.014	0.004	0.004	0.040	-
$-2x^5 - 2x^3 - 2x^2 - 2x - 1$	0.016	0.016	0.004	0.040	0.008	-
$-2x + 2x^3 + 2x^4 + 2x^5$	0.064	0.050	0.040	0.048	0.040	-
$-2x + 2x^3 + x^4 - 2x^5$	0.006	0.020	0.006	0.006	0.006	-
$-2x + 2x - 2x^2 - x^4 - 2x^5$	0.002	0.002	0.000	0.000	0.002	-
$-2x + x^2 - 2x^3 + 2x^4 - 2x^5$	0.006	0.010	0.004	0.006	0.008	-
$1 - 2x + x^2 - x^3 - x^5$	0.120	0.098	0.092	0.084	0.046	-
$1 - x^3 + 2x^4 - 2x^5$	0.046	0.036	0.030	0.038	0.022	-
$1 + 2x^2 - x^3 + 2x^4 - 2x^5$	0.020	0.044	0.024	0.018	0.006	-
$1 + 2x - x^2 + 2x^5$	0.106	0.112	0.092	0.086	0.082	-
$2 - 1x - 2x^2 + 2x^3 - x^4 - x^5$	0.008	0.002	0.002	0.002	0.002	-
$2 + 2x^2 + x^3 - 2x^4 - x^5$	0.070	0.068	0.050	0.048	0.042	-
$2 + x - 2x^2 - 2x^3 + x^5$	0.038	0.034	0.030	0.030	0.042	-
$2 + x - x^2 - x^4 - x^5$	0.052	0.024	0.032	0.032	0.028	-
$2 + x - x^2 + x^4 - x^5$	0.052	0.024	0.032	0.032	0.028	-

The trend of the symbolic regression results implies, somewhat counter-intuitively, that higher levels of correlation between genotype and phenotype distance tended to produce poorer performance in CGP. We consider three feasible explanations. Firstly, it is likely that the constraints imposed by high locality have restricted the diversity of the search, which may render intermediary schema difficult to reach. Secondly, in Rothlauf's model of locality, poorer performance under higher locality can be associated with fitness landscapes which are misleading [1] or deceptive (for example, GA trap functions). Further investigation of the fitness landscapes for these specific problem instances would be required to determine whether this is the case for this representation. Thirdly, it is unclear how features of the CGP genotype-phenotype map not addressed here, such as high redundancy, or structural bias [26] contribute to the trend.

In practice, using locality as a general performance predictor, or as a method of directly tuning existing genotype-phenotype maps, is clearly a challenging issue in integer encoded GP. Bypassing the intermediary stage and relating genotype and fitness values may lead to better outcomes on individual problems, but provides limited guidance for improving GP representations in general. Despite these outstanding problems, from this initial work we anticipate that the Mantel

test will prove a useful addition to existing approaches in statistical analyses of GP genotype-phenotype maps. It is encouraging that a technique founded in ecology can also contribute to the study of a complex artificial system.

7 Conclusions

The Mantel test, a statistical technique from numerical ecology, was adopted to analyse the locality of GP maps. As a case study, we examined an established integer representation, CGP. It was observed that a weakly positive correlation exists for CGP over short genotype distances, when using arithmetic function sets. We introduced a method of scaling the locality of a CGP genotype-phenotype map, by providing a bias into the mutation operator based on the normalised compression distance between phenotypes. To our knowledge, this is the first instance of explicitely controlled locality explored within a graph-based representation. The effect of varying locality on performance was measured for randomly generated polynomial symbolic regression problems. Higher locality was associated with reduced performance over 19 instances. We infer that employing less local maps may be advantageous on these classes of problem. In the future, we intend to test the robustness of this approach by applying it to other non-standard genotype-phenotype maps, such as grammar GP. Direct comparisons with alternative locality measures would also be appropriate.

Acknowledgements. Particular thanks are due to Dr. Dan Franks and other members of the York Centre for Complex Systems Analysis for advice concerning the Mantel statistic.

References

[1] Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms, pp. 33–96. Springer, Heidelberg (2006)
[2] Gottlieb, J.: Empirical Analysis of Locality, Heritability and Heuristic Bias in Evolutionary Algorithms: A Case Study for the Multidimensional Knapsack Problem. Evolutionary Computation 43, 441–475 (2004)
[3] Gen, M., Cheng, R.: Genetic Algorithms and Engineering Optimisation. John Wiley and Sons, Inc. (2000)
[4] Rothlauf, F., Goldberg, D.E.: Pruefer Numbers and Genetic Algorithms: A Lesson on How the Low Locality of an Encoding Can Harm the Performance of GAs. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 395–404. Springer, Heidelberg (2000)
[5] Droste, S., Wiesmann, D.: On Representation and Genetic Operators in Evolutionary Algorithms. Technical report (SFB) 531: [249], Univ. of Dortmund (1998)
[6] Galván-López, E., McDermott, J., Brabazon, A.: Defining locality as a problem difficulty measure in genetic programming. Genetic Programming and Evolvable Machines, 1–37 (2011)
[7] Oltean, M., Grosnan, C., Diosan, L., Mihaila, C.: Genetic Programming with Linear Representation: A Survey. Int. J. on Artificial Intelligence Tools, 197–238 (2008)
[8] Miller, J.F. (ed.): Cartesian Genetic Programming. Springer, Heidelberg (2011)

[9] Rothlauf, F., Oetzel, M.: On the Locality of Grammatical Evolution. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 320–330. Springer, Heidelberg (2006)
[10] Fagan, D., O'Neill, M., Galván-López, E., Brabazon, A., McGarraghy, S.: An Analysis of Genotype-Phenotype Maps in Grammatical Evolution. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 62–73. Springer, Heidelberg (2010)
[11] Brameier, M.F., Banzhaf, W.: Linear Genetic Programming. Genetic and Evolutionary Computation. Springer, Heidelberg (2007)
[12] Mantel, N.: The detection of disease clustering and a generalized regression approach. Cancer Research 27, 209–220 (1967)
[13] Dietz, E.J.: Permutation tests for association between two distance matrices. Systematic Zoology 32, 21–26 (1983)
[14] Oden, N.L., Sokal, R.R.: Directional autocorrelation: an extension of spatial correlograms to two dimensions. Systematic Biology 35, 608 (1986)
[15] Legendre, P., Fortin, M.-J.: Spatial pattern and ecological analysis. Vegetatio 80, 107–138 (1989)
[16] Legendre, P., Lapointe, F.J., Cagrain, P.: Modeling brain evolution from behavior: a permutational regression approach. Evolution 48, 1487–1499 (1994)
[17] Lichstein, J.W.: Multiple regression on distance matrices: a multivariate spatial analysis tool. Plant Ecology 188, 117–131 (2006)
[18] Legendre, P., Fortin, M.-J.: Comparison of the Mantel test and alternative approaches for detecting complex multivariate relationships in the spatial analysis of genetic data. Molecular Ecology Resources, 831–844 (2010)
[19] Chiam, S.C., Tan, K.C., Goh, C.K., Al Mamun, A.: Improving locality in binary representation via redundancy.. IEEE Trans. on Sys. Man. and Cybernetics (B) 38, 808–825 (2008)
[20] McDermott, J., Galván-Lopéz, E., O'Neill, M.: A Fine-Grained View of GP Locality with Binary Decision Diagrams as Ant Phenotypes. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 164–173. Springer, Heidelberg (2010)
[21] Krawiec, K.: Semantically Embedded Genetic Programming. In: Genetic and Evolutionary Computation Conference, Dublin, Ireland, pp. 1379–1386 (2011)
[22] Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proc. of the 6th Int. Conference on Genetic Algorithms, vol. 129, pp. 184–192. Citeseer (1995)
[23] Legendre, P., Legendre, L.: Numerical Ecology, 2nd edn. Developments in Environmental Modelling. Elsevier (1998)
[24] Goslee, S.C., Urban, D.L.: The ecodist Package for Dissimilarity-based Analysis of Ecological Data. Journal Of Statistical Software 22 (2007)
[25] Hien, N.T., Hoai, N.X.: A Brief Overview of Population Diversity Measures in Genetic Programming. In: 3rd Asian-Pacific Workshop on Genetic Programming, pp. 128–139 (2006)
[26] Payne, A.J., Stepney, S.: Representation and Structural biases in CGP. In: IEEE Congress on Evolutionary Computation, vol. 8, pp. 1064–1071. IEEE (2009)
[27] Vanneschi, L.: Theory and Practice for Efficient Genetic Programming. PhD thesis, Univ. of Lausanne (2004)
[28] Keijzer, M.: Efficiently Representing Populations in Genetic Programming. In: Advances in Genetic Programming, vol. 2, pp. 259–278. MIT Press (1996)
[29] Vanneschi, L.: Crossover-Based Tree Distance in Genetic Programming. IEEE Transactions on Evolutionary Computation 12, 506–524 (2008)
[30] Biernacki, P., Waldorf, D.: Snowball Sampling: Problems, Techniques and Chain-Referral Sampling. Socio. Methods And Research 10, 141–163 (1981)

Coevolution in Cartesian Genetic Programming

Michaela Šikulová and Lukáš Sekanina

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Božetěchova 2, 612 66 Brno, Czech Republic
{isikulova,sekanina}@fit.vutbr.cz

Abstract. Cartesian genetic programming (CGP) is a branch of genetic programming which has been utilized in various applications. This paper proposes to introduce coevolution to CGP in order to accelerate the task of symbolic regression. In particular, fitness predictors which are small subsets of the training set are coevolved with CGP programs. It is shown using five symbolic regression problems that the (median) execution time can be reduced 2–5 times in comparison with the standard CGP.

Keywords: Cartesian genetic programming, Coevolution, Symbolic regression.

1 Introduction

Cartesian genetic programming (CGP) is a variant of *genetic programming* (GP) that uses a specific encoding in the form of directed acyclic graph and a mutation-based search [11, 10]. CGP has been successfully employed in many traditional application domains of genetic programming such as symbolic regression. It has, however, been predominantly applied in evolutionary design and optimization of logic networks.

The *fitness evaluation* is typically the most time consuming part of CGP in these applications. In the case of digital circuit evolution, it is necessary to verify whether a candidate n-input circuit generates correct responses for all possible input combinations (i.e., 2^n assignments). It was shown that testing just a subset of 2^n test vectors does not lead to correctly working circuits [6, 9]. Recent work has indicated that this problem can partially be eliminated in real-world applications by applying formal verification techniques [15].

In the case of *symbolic regression*, k fitness cases are evaluated during one fitness function call, where k typically goes from hundreds to ten thousands. The time needed for evaluating a single fitness case depends on a particular application. Usually, the goal of GP system design and GP parameters' tuning is to obtain a solution with predefined accuracy and robustness using a minimum number of evaluated fitness cases or fitness function calls. In order to reduce the evaluation time, *fitness approximation* techniques have been employed. One of them is *fitness modeling* which uses fitness models with different degrees of sophistication to reduce the fitness calculation time [7]. It is assumed that the

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 182–193, 2012.
© Springer-Verlag Berlin Heidelberg 2012

fitness model can be constructed and updated in a reasonable time. The motivation for fitness modeling can be seen not only in reducing the complexity of fitness evaluation but also in avoiding the explicit fitness definitions, coping with noisy data, smoothing the fitness landscape and promoting diversity [14]. Fitness modeling is typically based on machine learning methods, subsampling of training data or partial evaluation.

Fitness prediction is a low cost adaptive procedure utilized to replace fitness evaluation. A framework for reducing the computation requirements of symbolic regression using fitness predictors has been introduced for standard genetic programming by Schmidt and Lipson [14]. Their method combines fitness prediction with coevolution to eliminate disadvantages of a classic fitness modeling, in particular the effort needed to train a fitness model and adapt the level of approximation and accuracy. The method utilizes a coevolutionary algorithm which exploits the fact that one individual can influence the relative fitness ranking between two other individuals in the same or a separate population [5]. Coevolving the training samples as the method of fitness modeling in GP has been studied in many aplication domains [2, 3, 4, 8] and in the symbolic regression problem [1, 12, 13, 14].

The goal of this paper is to introduce coevolving fitness predictors to CGP and show that by using them, the execution time of symbolic regression can significantly be reduced. The proposed coevolution of CGP programs and fitness predictors in the symbolic regression problem uses two populations evolving concurrently. Properties of individuals in the population of candidate programs change in response to properties of individuals in the population of fitness predictors and vice versa. It is expected that CGP which has been accelerated using coevolution will be implemented on a chip in our future work. Hence the proposed approach will also be useful for evolvable hardware purposes. Note that hardware implementation of CGP is straightforward which is not the case of tree-based GP [10].

The proposed coevolutionary CGP method is compared with a standard CGP on five symbolic regression problems. A brief comparison of CGP and tree-based GP is also performed on selected benchmark problems.

The rest of the paper is organized as follows. Section 2 introduces Cartesian genetic programming and its application to the symbolic regression problem. In Section 3, a new coevolutionary approach to CGP is presented. Section 4 compares the proposed coevolutionary algorithm with the standard CGP on five test problems. Experimental results are discussed in Section 5. Finally, conclusions are given in Section 6.

2 Cartesian Genetic Programming

In standard CGP (chapter 2 of [10]), a candidate program is modeled as an array of n_c (columns) $\times n_r$ (rows) of programmable elements (nodes). The number of primary inputs, n_i, and outputs, n_o, of the program is fixed. Each node input can be connected either to the output of a node placed in previous l columns or

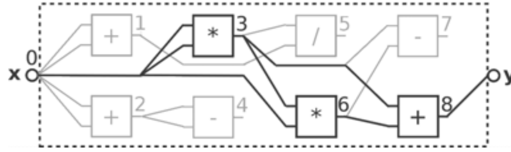


Fig. 1. A candidate program in CGP, where $l = 4$, $n_c = 4$, $n_r = 2$, $n_i = 1$, $n_o = 1$, $n_a = 2$, $\Gamma = \{+ (1), - (2), * (3), / (4)\}$ and chromosome is: $0, 0, 1$; $0, 0, 1$; $0, 0, 3$; $2, 2, 2$; $3, 1, 4$; $3, 0, 3$; $3, 6, 2$; $3, 6, 1$; 8

to one of the program inputs. The l-back parameter, in fact, defines the level of connectivity and thus reduces/extends the search space. Feedback is not allowed. Each node is programmed to perform one of n_a-input functions defined in the set Γ. Each node is encoded using $n_a + 1$ integers where values $1 \ldots n_a$ are the indexes of the input connections and the last value is the function code. Every individual is encoded using $n_c \cdot n_r \cdot (n_a + 1) + n_o$ integers. Figure 1 shows an example of a candidate circuit. While the primary inputs are numbered $0 \ldots n_i - 1$ the nodes are indexed $n_i \ldots n_c n_r + n_i - 1$.

A simple (1+λ) evolutionary algorithm is used as a search mechanism. It means that CGP operates with the population of $1 + \lambda$ individuals (typically, λ is between 1 and 20). The initial population is constructed either randomly or by a heuristic procedure. Every new population consists of the best individual of the previous population (so-called parent) and its λ offspring. However, as a new parent an offspring is always chosen if it is equally as fit or has better fitness than the parent. The offspring individuals are created using a point mutation operator which modifies h randomly selected genes of the chromosome, where h is the user-defined value. The algorithm is terminated when the maximum number of generations is exhausted or a sufficiently working solution is obtained.

For symbolic regression problems, the goal of evolution is usually to minimize the *mean absolute error* of a candidate program response y and target response t. The fitness function (taking candidate program s as its argument) is then defined

$$\mathrm{f}(s) = \frac{1}{k} \sum_{j=1}^{k} |y(j) - t(j)| \tag{1}$$

where k is the number of fitness cases. Alternatively, the *number of hits* can represent the fitness value. The number of hits is defined

$$\mathrm{f}(s) = \sum_{j=1}^{k} \mathrm{g}(y(j)), \text{ where} \tag{2}$$

$$\mathrm{g}(y(j)) = \begin{cases} 0 \text{ if } |y(j) - t(j)| \geq \varepsilon \\ 1 \text{ if } |y(j) - t(j)| < \varepsilon \end{cases} \tag{3}$$

and ε is a user-defined acceptable error.

3 Coevolution of Fitness Predictors in CGP

The aim of coevolving fitness predictors and programs is to allow both solutions (programs) and fitness predictors to enhance each other automatically until a satisfactory problem solution is found. We propose to adopt Schmidt's and Lipson's approach [14] using CGP for the task of symbolic regression. Figure 2 shows the overall scheme of the proposed method. There are two concurrently working populations: (1) candidate programs (syntactic expressions) evolving using CGP and (2) fitness predictors evolving using a genetic algorithm.

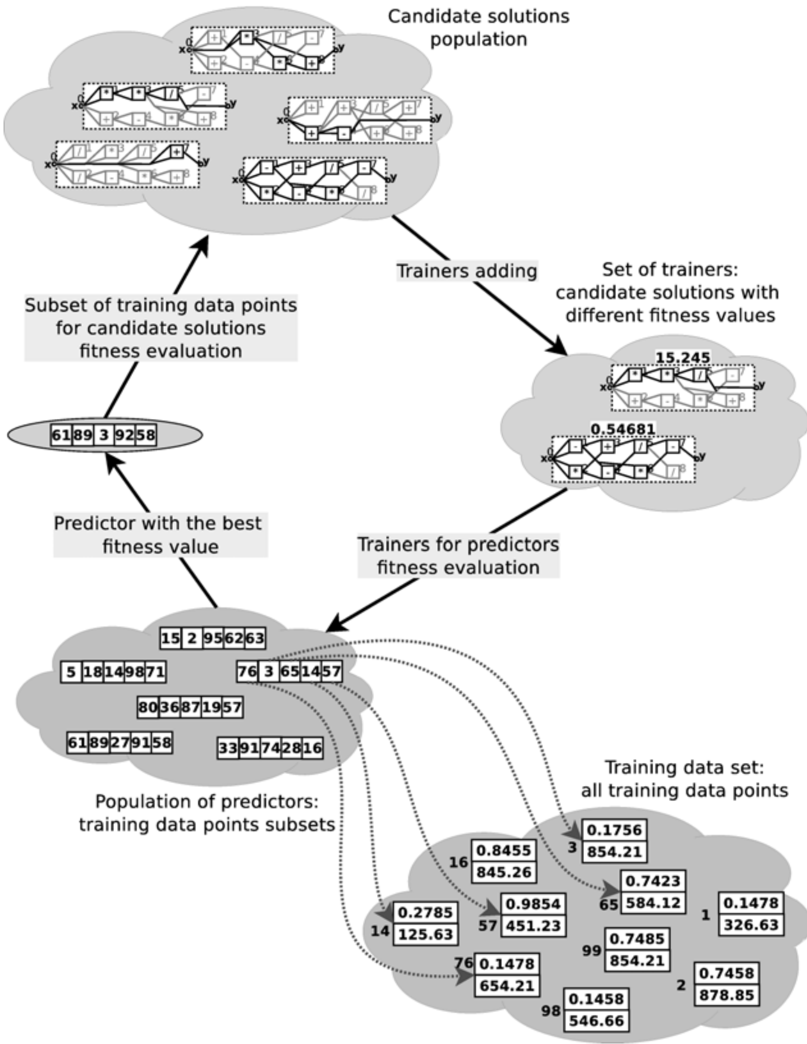


Fig. 2. Coevolution of candidate programs and fitness predictors

3.1 Population of Candidate Programs

Evolution of candidate programs is based on principles of CGP as introduced in Section 2. The fitness function for CGP is defined as the relative number of hits. There are, in fact, two fitness functions for candidate program s. While the exact fitness function $f_{exact}(s)$ utilizes the complete training set, the predicted fitness function $f_{predicted}(s)$ employs only selected fitness cases. Formally,

$$\mathrm{f}_{\mathrm{exact}}\left(s\right)=\frac{1}{k}\sum_{j=1}^{k}g\left(y\left(j\right)\right) \quad (4)$$

$$\mathrm{f}_{\mathrm{predicted}}\left(s\right)=\frac{1}{m}\sum_{j=1}^{m}g\left(y\left(j\right)\right) \quad (5)$$

where k is the number of data points in the training set and m is the number of data points in the fitness predictor (i.e., m is the size of a subset of the training set).

3.2 Set of Trainers

The set of trainers which contains several candidate programs is used to evaluate fitness predictors. The proposed implementation differs from [14] in the organization and update strategy. In particular, the set of trainers is divided into two parts. The first part is periodically updated from the population of candidate programs (the best-scored candidate program is sent to the trainers set if its fitness value differs from the best-scored candidate program in the previous generation) and the second part is periodically and randomly generated to ensure genetic diversity of the set of trainers. The size of trainers set is kept constant during evolution. For every new selected or generated trainer, the exact fitness is calculated and the new trainer replaces the oldest one in the corresponding part of the trainers set.

3.3 Population of Fitness Predictors

Fitness predictor is a small subset of training data. An optimal fitness predictor is sought using a simple genetic algorithm (GA) which operates with a population of fitness predictors. Every predictor is encoded as a constant-size array of pointers to elements in the training data. In addition to one-point crossover and mutation, a randomly selected predictor replacing the worst-scored predictor in each generation has been introduced as a new genetic operator of GA. The fitness value of predictor p is calculated using the *mean absolute error* of the exact and predicted fitness values of trainers

$$\mathrm{f}\left(p\right)=\frac{1}{u}\sum_{i=1}^{u}\left|\mathrm{f}_{\mathrm{exact}}\left(i\right)-\mathrm{f}_{\mathrm{predicted}}\left(i\right)\right| \quad (6)$$

where u is the number of candidate programs in the trainers set. The predictor with the best fitness value is used to predict the fitness of candidate programs in the population of candidate programs.

3.4 Implementation

Two threads are used. The first one is responsible for candidate programs evolution using CGP. The second thread performs evolution of fitness predictors using a simple genetic algorithm. The coevolution is implemented as follows. The first thread randomly initializes both populations and also randomly creates the first individuals in the set of trainers. After the second thread is activated both populations are evaluated.

CGP evolution loop begins with loading the fittest training data sample from population of fitness predictors. This is performed periodically, but not in every iteration due to a slower rate of fitness predictors evolution. This results in a lower computational effort. It is not necessary to run the fitness predictor evolution as fast as the candidate program evolution, because fast changes of the best rated fitness predictor do not contribute to convergence.

The next step involves calculating the predicted fitness of all individuals in the candidate program population. The best rated individual is then selected and its number of hits is checked. If the predicted fitness value is not in the interval of acceptable fitness values, CGP will create a new population, eventually new trainer will be selected or generated. If predicted fitness value falls into the interval of acceptable fitness values, the exact fitness of candidate program is evaluated. If the exact fitness falls into the interval of acceptable fitness values, a solution is found, and coevolution is terminated. Otherwise, the update of the best rated fitness predictor is signaled and the coevolution has to continue.

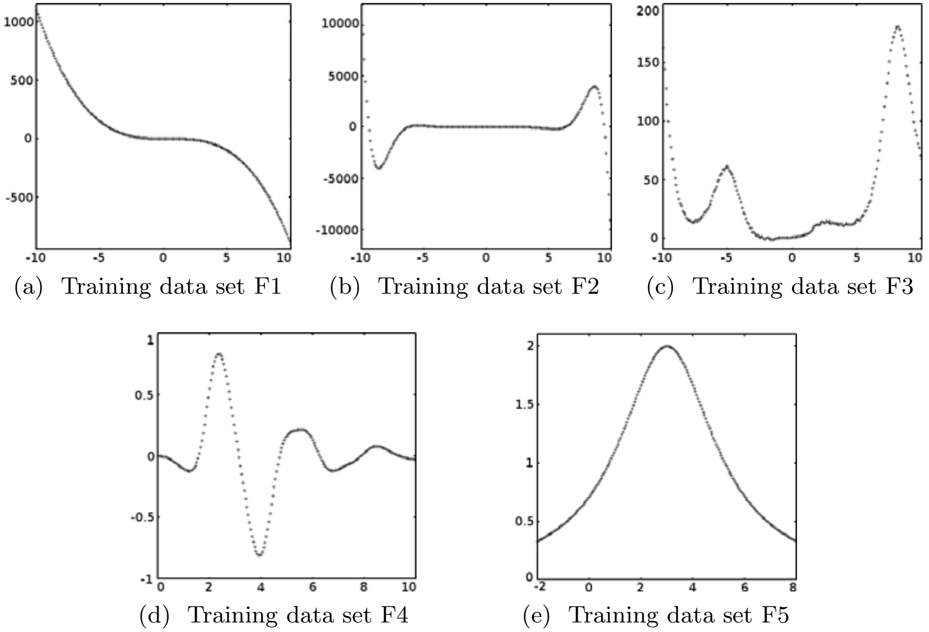
The second thread performs the evolution of fitness predictors. The fitness values of all fitness predictors are evaluated using trainers. The best rated predictor is selected and stored to shared memory. The next step involves creating of a new generation of fitness predictors by means of GA operators. Subsequently, the GA waits for a signal from the first thread. After receiving the signal, the GA loop continues with the next iteration, or if a solution is discovered, GA is terminated.

4 Results

This section presents benchmark problems, experimental setup, experimental evaluation of the proposed coevolutionary approach to CGP and its comparison with standard CGP.

4.1 Benchmark Problems

Five test functions (F1 – F5) were selected as data point sources for evaluation of the proposed method:



(a) Training data set F1 (b) Training data set F2 (c) Training data set F3

(d) Training data set F4 (e) Training data set F5

Fig. 3. Training data sets: x values on horizontal axes, $f(x)$ values on vertical axis

$$F1: f(x) = x^2 - x^3, x \in \langle -10, 10 \rangle \quad (7)$$

$$F2: f(x) = e^{|x|} \sin(x), x \in \langle -10, 10 \rangle \quad (8)$$

$$F3: f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right), x \in \langle -10, 10 \rangle \quad (9)$$

$$F4: f(x) = e^{-x} x^3 \sin(x) \cos(x) \left(\sin^2(x) \cos(x) - 1\right), x \in \langle 0, 10 \rangle \quad (10)$$

$$F5: f(x) = \frac{10}{(x-3)^2 + 5}, x \in \langle -2, 8 \rangle \quad (11)$$

In order to form a training set, 200 equidistant distributed samples were taken from each function (see Fig. 3). Functions F1, F2 and F3 are taken from [14] and functions F4 and F5 from [16]. Table 1 shows acceptable errors and the acceptable number of hits.

4.2 Experimental Setup

Table 1 shows that various settings of the components involved in the proposed coevolutionary method have been tested. Over 100,000 independent runs were performed to find the most advantageous setting which is presented in the right-most column and which is later used in all reported experiments.

Programs are evolved using CGP with the following setup: $l = n_c$, $n_r = 1$, $n_i = 1$, $n_o = 1$, every node has two inputs (i_1, i_2) and $\Gamma = \{\ i_1 + i_2, i_1 - i_2, i_1 \cdot i_2, \frac{i_1}{i_2}, \sin(i_1), \cos(i_1), e^{i_1}, \log(i_1)\}$. Table 1 shows various setting of n_c, λ and h considered during parameters tuning.

Fitness predictors evolution is conducted using a simple GA. Table 1 shows numerous setting of the chromosome length, population size and genetic operators.

Other parameters of coevolution, such as the size of trainers set, frequency of trainers substitutions and predictors evolution deceleration are also given in Table 1.

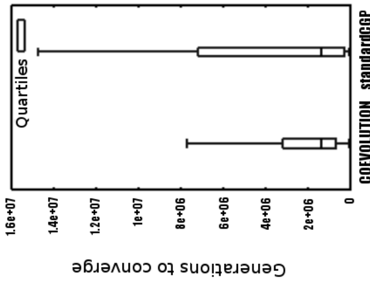
4.3 Comparison of Coevolving CGP with Standard CGP

The proposed coevolutionary algorithm was compared with standard CGP using test functions F1-F5. Parameters of both algorithms were chosen according to Table 1 and 50 independent runs were performed. Table 2 gives the resulting success rate (the number of runs giving a solution with predefined quality), the number of generations, the number of data point evaluations and time to converge calculated as median out of 50 independent runs. Figure 4 shows quartile graphs of the number of generations and data point evaluations for all five training data sets.

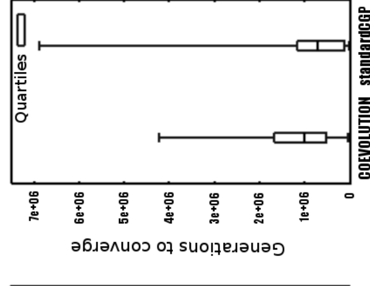
Figure 5 shows the progress of the best fitness value during a typical run on the F2 data set. It can be seen that while the progress is monotonic for the standard CGP, the coevolutionary algorithm produces very dynamic changes ending with a significant increase of the best fitness value at the end of evolution. The changes of the best fitness value are caused by updating of the best fitness predictor.

Table 1. Experimental setup

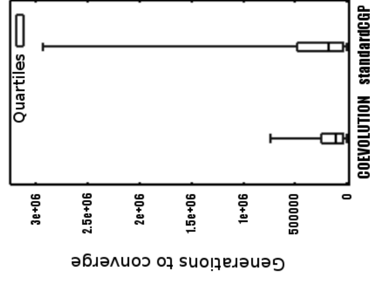
	Parameter	Tested values	Selected values
CGP	Chromosome length n_c	16, 24, 32, 64, 96, 128	32
	Population size λ	4, 8, 12, 16, 20	12
	Number of mutations h per individual	1-4, 1-8, 1-12, 1-16	1-8
Trainers, Coevolution	All trainers substitution	1 per 500 generations of CGP	500
	Trainers set size	8, 12, 16, 24, 32	8
	Predictor evolution deceleration	1 per 10, 25, 50, 100, 150, 200 generations of CGP	100
GA-Predictors	Chromosome length	4, 8, 12, 16, 24, 32, 64	12
	Population size	8, 12, 16, 24, 32, 48, 64, 96, 128	32
	Offspring creation	2-tournament selection, single point crossover	
	Mutation probability	0.2	0.2
Test functions	Acceptable error of data point	F1, F2: 0.5; F3: 1.5; F4, F5: 0.025	
	Acceptable number of hits	97 %	97 %



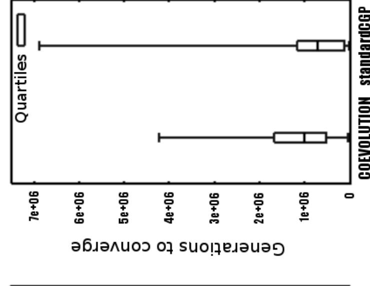
(a) Training data set F1.



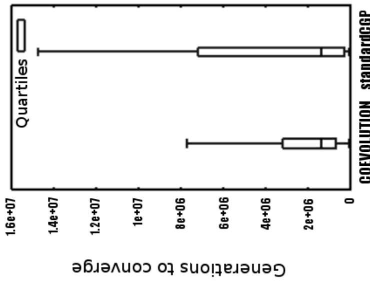
(b) Training data set F2.



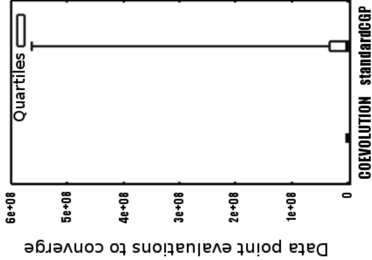
(c) Training data set F3.



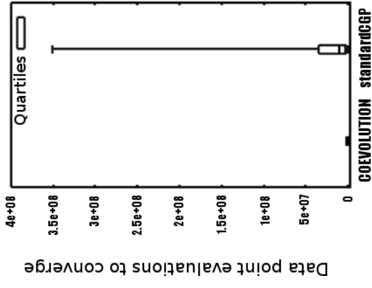
(d) Training data set F4.



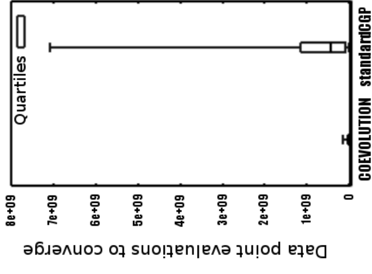
(e) Training data set F5.



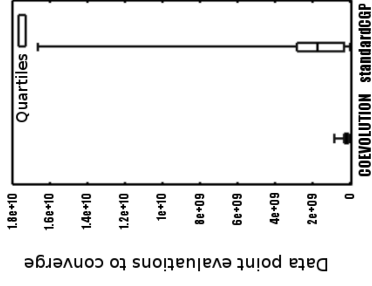
(f) Training data set F1.



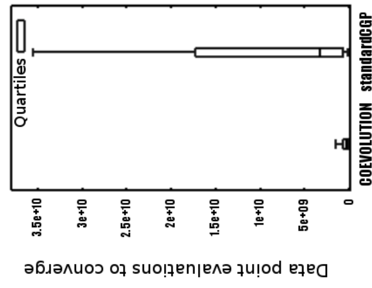
(g) Training data set F2.



(h) Training data set F3.



(i) Training data set F4.

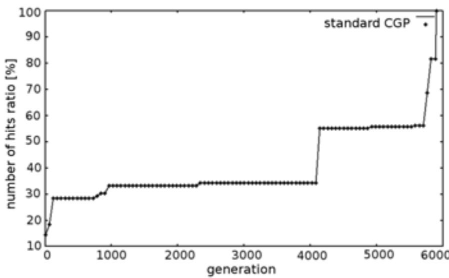


(j) Training data set F5.

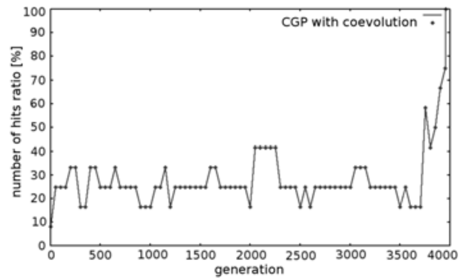
Fig. 4. Comparison of standard CGP and CGP with coevolution: Number of generations and number of data point evaluations to converge

Table 2. Comparison of standard CGP and CGP with coevolution for five training data sets

		F1	F2	F3	F4	F5
Success rate	stand. CGP	100 %	100 %	78 %	80 %	24 %
	coevolution	100 %	100 %	100 %	100 %	100 %
Generations to converge (median)	stand. CGP	$1.11 \cdot 10^3$	$4.46 \cdot 10^3$	$1.76 \cdot 10^5$	$7.15 \cdot 10^5$	$1.36 \cdot 10^6$
	coevolution	$2.62 \cdot 10^3$	$2.53 \cdot 10^3$	$1.10 \cdot 10^5$	$1.00 \cdot 10^6$	$1.34 \cdot 10^6$
Data point evaluations to converge (median)	stand. CGP	$2.68 \cdot 10^6$	$1.08 \cdot 10^7$	$4.24 \cdot 10^8$	$1.72 \cdot 10^9$	$3.28 \cdot 10^9$
	coevolution	$5.20 \cdot 10^5$	$5.01 \cdot 10^5$	$2.19 \cdot 10^7$	$2.00 \cdot 10^8$	$2.67 \cdot 10^8$
Time to converge (median) [s]	stand. CGP	35.4611	55.1476	98.8585	44.0388	104.6826
	coevolution	17.4588	21.1178	18.1257	17.1079	20.4529



(a) Standard CGP.



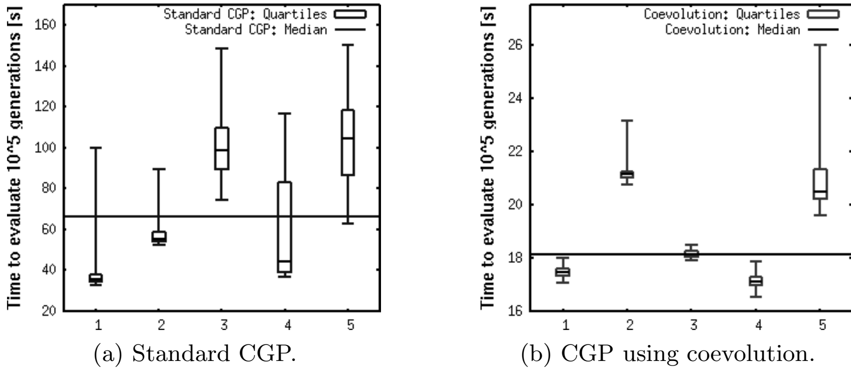
(b) CGP with coevolution.

Fig. 5. Progress of the best fitness value during a typical run for the F2 data set

5 Discussion

It can be seen from Table 2 that the proposed coevolutionary method has reached a satisfactory solution using much fewer data point evaluations than the standard CGP. The speedup measured on the Intel® Core^{TM} i5-2500 machine is between 2.03 (F1) and 5.45 (F3). Detailed analysis of execution time is shown in Fig. 6 where quartile graphs are given for 50 independent runs. However, it should be pointed out that the standard CGP evaluates 200 fitness cases in every fitness function call while the coevolutionary algorithm evaluates only 12 fitness cases. The number of generations is similar for both methods. A notable observation is that while the standard CGP was not able to produce a satisfactory solution in 23.6% runs the proposed method reached a satisfactory solution in all cases. Moreover, the results generated by multiple runs of coevolutionary CGP are more stable than those produced by the standard CGP.

There is only one data set (F2) and corresponding results of the tree-based coevolutionary GP [14] which can serve for a direct comparison with our CGP-based coevolution. While tree-based GP requires $1 \cdot 10^3$ generations and $7 \cdot 10^6$



(a) Standard CGP. (b) CGP using coevolution.

Fig. 6. Time of evolution

data point evaluations to converge, the proposed CGP-based approach requires $3 \cdot 10^3$ generations and only $5 \cdot 10^5$ data point evaluations to converge. The proposed method seems to be competitive with [14].

6 Conclusions

Symbolic regression has not been considered as a typical application domain for CGP. We have shown in this paper that CGP equipped with coevolution of fitness predictors can significantly be accelerated in this particular application. The speedup obtained for five test problems is 2.03 – 5.45 over the standard CGP. Results are also very competitive with the tree-based GP.

Our future work will be devoted to utilization of the proposed coevolutionary algorithm in other applications domains where the standard CGP has been successful so far. Another goal will be to implement the coevolutionary CGP on a chip and use it in a real-world application.

Acknowledgments. This work was supported by the Czech science foundation project P103/10/1517, the research programme MSM 0021630528, the BUT project FIT-S-11-1 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

[1] Dolin, B., Bennett III, F.H., Reiffel, G.: Co-evolving an effective fitness sample: Experiments in symbolic regression and distributed robot control. In: Proc. of the 2002 ACM Symp. on Applied Computing, pp. 553–559. ACM, New York (2002)
[2] Dolinsky, J.U., Jenkinson, I.D., Colquhoun, G.J.: Aplication of genetic programming to the calibration of industrial robots. Computers in Industry 58(3), 255–264 (2007)
[3] Gagné, C., Parizeau, M.: Co-evolution of nearest neighbor classifiers. International Journal of Pattern Recognition and Artificial Inteligence 21(5), 921–946 (2007)

[4] Harrison, M.L., Foster, J.A.: Co-evolving Faults to Improve the Fault Tolerance of Sorting Networks. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) EuroGP 2004. LNCS, vol. 3003, pp. 57–66. Springer, Heidelberg (2004)

[5] Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42(1), 228–234 (1990)

[6] Imamura, K., Foster, J.A., Krings, A.W.: The Test Vector Problem and Limitations to Evolving Digital Circuits. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 75–79. IEEE Computer Society (2000)

[7] Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing Journal 9(1), 3–12 (2005)

[8] Mendes, R.R.F., de Voznika, F.B., Freitas, A.A., Nievola, J.C.: Discovering Fuzzy Classification Rules with Genetic Programming and Co-evolution. In: Siebes, A., De Raedt, L. (eds.) PKDD 2001. LNCS (LNAI), vol. 2168, pp. 314–325. Springer, Heidelberg (2001)

[9] Miller, J.F., Thomson, P.: Aspects of Digital Evolution: Geometry and Learning. In: Sipper, M., Mange, D., Pérez-Uribe, A. (eds.) ICES 1998. LNCS, vol. 1478, pp. 25–35. Springer, Heidelberg (1998)

[10] Miller, J.F.: Cartesian Genetic Programming. Springer, Heidelberg (2011)

[11] Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)

[12] Pagie, L., Hogeweg, P.: Evolutionary consequences of coevolving targets. Evolutionary Computation 5(4), 401–418 (1997)

[13] Schmidt, M., Lipson, H.: Co-evolving fitness predictors for accelerating and reducing evaluations. In: Genetic Prog. Theory and Practice IV, vol. 5, pp. 113–130 (2006)

[14] Schmidt, M.D., Lipson, H.: Coevolution of Fitness Predictors. IEEE Transactions on Evolutionary Computation 12(6), 736–749 (2008)

[15] Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genetic Programming and Evolvable Machines 12(3), 305–327 (2011)

[16] Vladislavleva, E.: Symbolic Regression: Toy Problems for Symbolic Regression (2009-2010), http://www.vanillamodeling.com/toyproblems.html

Evolutionary Design of Message Efficient Secrecy Amplification Protocols

Tobiáš Smolka[1], Petr Švenda[1], Lukáš Sekanina[2], and Vashek Matyáš[1,⋆]

[1] Masaryk University, Faculty of Informatics, Czech Republic
[2] Brno University of Technology, FIT, IT4Innovations Centre, Czech Republic
{xsmolka,svenda,matyas}@fi.muni.cz, sekanina@fit.vutbr.cz

Abstract. Secrecy amplification protocols are mechanisms that can significantly improve security of partially compromised wireless sensor networks (e.g., turning a half-compromised network into the 95% secure one). The main disadvantage of existing protocols is a high communication overhead increasing exponentially with network density. We devise a novel family of these protocols exhibiting only a linear increase of the communication overhead. The protocols are automatically generated by linear genetic programming (LGP) connected to a network simulator. After a deep analysis of various characteristics of this new family of protocols, with a special focus on the tuning of LGP parameters, new and better group-oriented protocols are discovered by LGP. A multi-criteria optimization is then utilized to further reduce the communication overhead down to 1/2 of the original amount while maintaining the original fraction of secure links.

1 Introduction

Wireless sensor networks (WSNs) are networks of resource-constrained battery-powered nodes that can communicate over short distances via wireless radio. The applications of such networks vary from environment monitoring to battlefield management and often require resistance against unauthorized reading, modification or generation of monitored information. To achieve this goal, encryption and message authentication techniques with shared symmetric keys between the communicating parties can be used. This is, however, a challenging task, since the nodes are usually distributed in an untrusted environment. An attacker can extract all keys from a physically captured node and easily intercept large fraction of communication in the network.

Secrecy amplification (SA) is a post-deployment technique for improving the security of communication in partially compromised networks. It can be employed in situations, where there are keys established between the nodes in the network, but some of them may be compromised. SA exploits the fact that a group of neighbouring nodes can cooperate and establish a new key derived

⋆ Final work on this paper undertaken as a Fulbright-Masaryk Visiting Scholar at Harvard University.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 194–205, 2012.
© Springer-Verlag Berlin Heidelberg 2012

from multiple old keys. The new key will be secure in case when at least one of the old ones was not compromised. The concept was initially introduced in [2] and further improved in [6,8].

In this work, we present newly discovered group-oriented secrecy amplification protocols with better performance than previously published node-oriented protocols. The protocols are automatically generated by linear genetic programming (LGP) [4] connected to a network simulator. We chose LGP because it is suitable for evolution of short domain-specific programs as shown by e.g. [3]. The discovery was made possible by deeply analysing and tweaking the evolutionary search and also by introducing another criterion for protocol optimization – a total number of exchanged messages. The goal of this paper is to analyze the impact of LGP parameters on the quality of evolved protocols and find better ones.

The paper is organized as follows: The next section provides a short introduction to secrecy amplification in WSNs and reviews previous work on automatic design of these protocols with Evolutionary Algorithms. Section 3 analyses the impact of different parameters such as population size, mutation and crossover rate, length of chromosome or number of generations on the performance of LGP. Section 4 describes new protocols found in long-running experiments with tuned LGP and analyses their performance and robustness. Section 5 introduces a multi-criteria optimization and shows that further reduction of totally exchanged messages in the protocols is possible. Conclusions are given in Section 6.

2 Previous Work

Secrecy amplification (SA) was initially introduced in [2] for the Key Infection (KI) key establishment, in which the keys are exchanged between the neighbours in plaintext. In case an attacker places an eavesdropping node close to a legitimate one, it is able to intercept all the keys exchanged with that node. The concept of SA can also be used when the compromised links are distributed randomly. Such a compromise pattern may result from the probabilistic distribution scheme of Eschenauer and Gligor (EG) [7].

The protocols presented in [2] and [6] use an "absolute" identification of the nodes (e.g., node number `1`, `2`, `3`.) If there are k parties (nodes) in the protocol and n neighbours of node on average then one run of the protocol must be executed for all k-tuples of neighbours leading to $\binom{n}{k}$ executions per single node – a huge communication overhead. The number of totally exchanged messages increases exponentially with the number of neighbours and is significant for WSNs where 6-15 neighbours are usually assumed. We will denote such protocols as node-oriented (NO).

A different approach to the design of amplification protocols was presented in [8]. Identification of the parties in protocol is given by the relative distance from two distinct nodes. It is assumed that each node knows the distance to its direct neighbours. This distance can be approximated from the minimal transmission power needed to communicate with a given neighbour. If the protocol has to express the fact that two nodes N_i and N_j are exchanging a message over

the intermediate node N_k, only relative distances of such node N_k from N_i and N_j are indicated in the protocol (e.g., $N_{(0.3_0.7)}$ is a node positioned 0.3 of the maximum transmission range from N_i and 0.7 from N_j). Based on the actual distribution of the neighbours, the node closest to the indicated distance(s) is chosen as the node N_k for particular protocol run. There is no need to re-execute the protocol for all k-tuples (as for NO protocols) as all neighbours can be involved in a single execution, reducing communication overhead significantly. See [8] for a detailed description of evaluation process for group-oriented protocols.

2.1 Evolution of Amplification Protocols

In order to improve the fraction of secure links and to decrease the necessary communication overhead (the number of messages), a new method for automatic generation of protocols was introduced in [8]. The method utilized linear genetic programming and a network simulator for evaluation of candidate amplification protocols with resulting fraction of secure links taken as fitness value. The use of LGP is especially important in case of group-oriented protocols, since the design of such a protocol is not a trivial task and to the best of our knowledge, no human-designed group-oriented protocol was published yet.

Instruction Set: Each party (a real node in network) in the protocol is modelled as a computing unit with a limited number of memory slots. Each memory slot can contain either a random value, encryption key or message. Each candidate protocol is modelled as a program composed of instructions from a specific instruction set given in Table 1. This instruction set was chosen because it enables to express all previously known amplification protocols and to utilize only operations available on real sensor nodes such as TelosB [5].

Using this set of primitive instructions, a simple plaintext exchange of new key can be written as {RNG N_1 R_1; SND N_1 N_2 R_1 R_1;}, a PUSH protocol [2] as {RNG N_1 R_1; SND N_1 N_3 R_1 R_1; SND N_3 N_2 R_1 R_1;}, a PULL protocol [6] as {RNG N_3 R_1; SND N_3 N_1 R_1 R_1; SND N_3 N_2 R_1 R_1;} and a multi-hop version of PULL [6] as {RNG N_3 R_1; SND N_3 N_1 R_1 R_1; SND N_3 N_4 R_1 R_1; SND N_4 N_2 R_1 R_1;}. All these protocols are node-oriented. Group-oriented protocols are longer and more complicated, see [8].

Table 1. Instruction set for amplification protocols

Instruction	Description
NOP	No operation is performed.
RNG N_a R_i	Generate a random value on node N_a into slot R_i.
SND N_a N_b R_i R_j	Send a value from R_i on node N_a to slot R_j on N_b.
CMB N_a R_i R_j R_k	Combine values from slots R_i and R_j on node N_a and store the result to R_k (e.g., cryptographic hash function like SHA-3).
ENC N_a R_i R_j R_k	Encrypt a value from R_i on node N_a using the key from R_j and store the result to R_k.
DEC N_a R_i R_j R_k	Decrypt a value from R_i on node N_a using the key from R_j and store the result to R_k.

Previous LGP Results: LGP rediscovered previously published protocols and also new and better performing protocols were found. The best performing node-oriented 4-party secrecy amplification protocol found [8] consists of 10 effective instructions with performance shown in Figure 3. Note that the LGP objective was to optimize the number of secure links only, not the number of messages. Additionally, only limited computing resources were available to obtain these results. We will show in next sections that better protocols (in terms of the number of secure links and messages needed) can be obtained with improved LGP settings and more computational resources.

3 LGP Tuning and Exploring the Design Space

This section describes the initial version of LGP and network simulator, together with heavily resource-consuming experiments conducted to determine the most suitable parameters of LGP that are necessary for finding new group-oriented protocols in Section 4. Distributed computation via BOINC (Berkeley Open Infrastructure for Network Computing) [1] with around 250 CPU cores was used to provide the performance necessary for all experiments[1].

3.1 Experimental Setup

Basic LGP Setup: LGP operates with the instruction set given in Table 1. The size of chromosome is 100 instructions. Every node contains 12 memory slots. The initial population is generated randomly. The mutation operator randomly picks an integer and generates a new value at its position. The crossover operator is applied at the level of instructions. A new population is formed using a tournament selection. The fitness function is defined as a fraction of secure communication links. The impact of various parameters of LGP on the performance is investigated in Section 3.2.

Network Simulation: Candidate protocols are simulated in a network of 100 legitimate nodes. During the evaluation, each amplification protocol was independently executed on 5 deployments, each of which with different placement of the nodes. This way the candidate protocol was prevented from optimizing on one particular network deployment and provided results usable also in networks with a higher number of nodes (we kept the number of nodes intentionally low so network simulation is executed fast enough). The nodes were always placed uniformly over a square area and each node had approx. 10 legitimate neighbours on average (over all deployments).

The fraction of initially secured links was intentionally set to 30%, so it is reasonably difficult to increase the fraction of secure links. In the EG compromise

[1] Raw data in searchable format from all experiments are available at web page `http://www.fi.muni.cz/~xsvenda/papers/EuroGP2012/` and additional experiments with examples of protocols found will be available in parallel technical report.

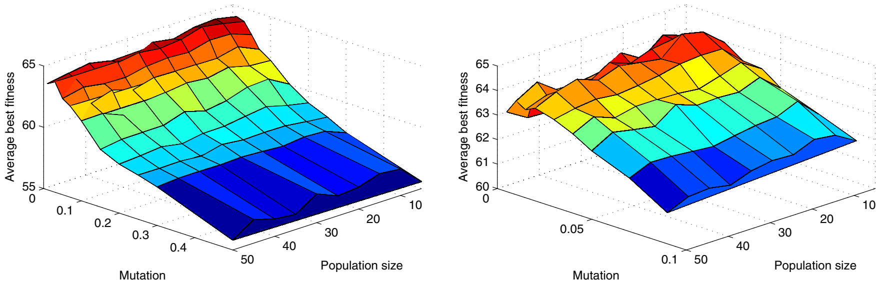


Fig. 1. Average of the best fitness values calculated from 20 independent runs using different mutation probabilities and population sizes: (a) all experiments, (b) zoomed

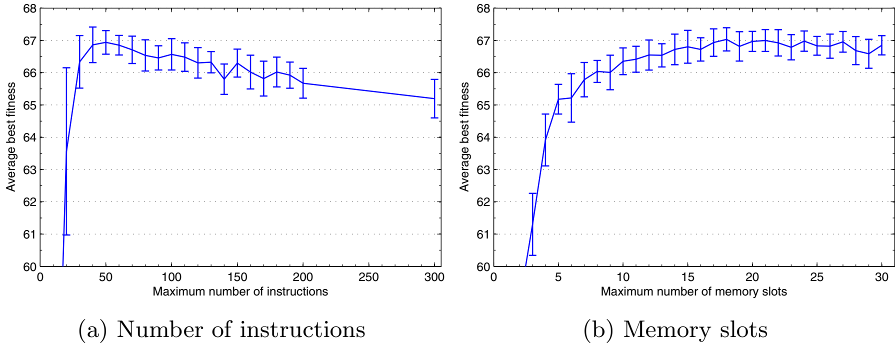
pattern, this number was used directly when deciding whether the link was initially compromised or not. In the KI pattern [2], the portion of compromised links was affected by the number of attacker's nodes in the network. This number was determined experimentally (30 attacker nodes), so the resulting fraction of compromised links was close to the desired level.

3.2 LGP Performance

Population Size and Mutation Rate: First, the most suitable values of the population size and probability of mutation were sought. We tested 10 different combinations of the number of generations (num_{gen}) and population size (pop_{size}) for 20 different mutation probabilities. The values of num_{gen} and pop_{size} were chosen so that value $num_{gen} * pop_{size}$ and consequently also the number of fitness evaluations remains constant ($num_{gen} * pop_{size} = 40\ 000$). Each combination of parameters (p_{mut}, num_{gen}, pop_{size}) was instantiated in 20 independent runs and the averages of the best fitness values are shown in Figure 1a and Figure 1b. Clearly, LGP performs better for small probability of mutation (between 0.005 and 0.05) and smaller population sizes (between 10 and 20). Values $p_{mut} = 0.02$ and $pop_{size} = 15$ were used in next experiments based on these results.

Similarly, the effect of crossover (p_{cross}) was investigated using 20 independent runs where p_{cross} was between 0 and 1. This experiment showed that the crossover has no significant impact on performance and thus crossover has not been used in other experiments.

Chromosome Length: Long chromosomes imply large search spaces that are usually difficult to search. They also lead to long programs that can take a considerable time to be executed. Previous work utilized 200 instructions; however only around 10 instructions were effectively used in evolved protocols [8]. In order to find a reasonable value for the maximum length of chromosome (ins_{max}), we fixed all the parameters except ins_{max} and num_{gen}. The values for ins_{max} and



(a) Number of instructions

(b) Memory slots

Fig. 2. Best fitness obtained from 20 independent runs for various limits in the number of instructions (a) and memory slots (b)

num_{gen} were chosen such that the value $ins_{max} * num_{gen}$ remains constant and equals to 1 200 000 (and therefore, the overall computational time is constant for different runs).

The average fitness value computed using the best fitness values obtained from 20 independent runs is depicted in Figure 2a. The experiment shows a sharp drop in the achievable fitness when less than 30 instructions ins_{max} are available. The performance is increasing for values of ins_{max} between 40 and 100 and slowly decreasing with bigger ins_{max} afterwards. This result is correlated with observed average number of effective instructions (i.e., the instructions that actually contribute to the fitness value, usually around 30, in our case 32 instructions at maximum) in best protocols such as EG_{best} evolved with $ins_{max} = 100$ or more. Note that doubling the number of available instructions ins_{max} will roughly double the time necessary to simulate a single protocol, but it will not impact the resulting protocol as usually only around 30 instructions are effective and we can automatically identify them. On the basis of these results, the following experiments were initialized to allow 50 (100 in some cases) instructions at maximum. The evaluation time has been significantly reduced without affecting the quality of evolved protocols.

Memory Slots: In the next experiment we analyzed the impact of limiting the number of memory slots ($slot_{max}$) which the evolved protocols can use. LGP parameters remain identical to the previous experiment ($ins_{max} = 100$). The average fitness values are depicted in Figure 2b for $slot_{max} = 1 \ldots 30$. The experiment revealed that the protocols require at least 10 memory slots to achieve a reasonable performance.

Number of Generations: The fitness increases with additional generations, but only to some extend. In order to determine the number of generations needed for reaching a reasonably performing protocol, 20 runs were executed for 53 340 generations ($ins_{max} = 100$, $slots_{max} = 12$). In this experiment, all runs reached

60% secure links after 1 067 generations, 65% after 8 529 generations and 66% after 15 991 generations. These 20 runs reached 67.72% on average, the worse one stagnated at 66.8%. In comparison, 60% secure links is the level which no random search was able to reach even after 14 000 generations (assuming the same population size).

4 Discovering New Group-Oriented Protocols

The search for new protocols utilized the best-performing parameters found in previous experiments. Complete setting is given in Table 2. For each of the compromise scenarios (KI, EG), we performed 20 independent runs and allowed a sufficient number of generations. The best performing protocol for each scenario was then taken and further analysed.

4.1 Long-Running Experiments

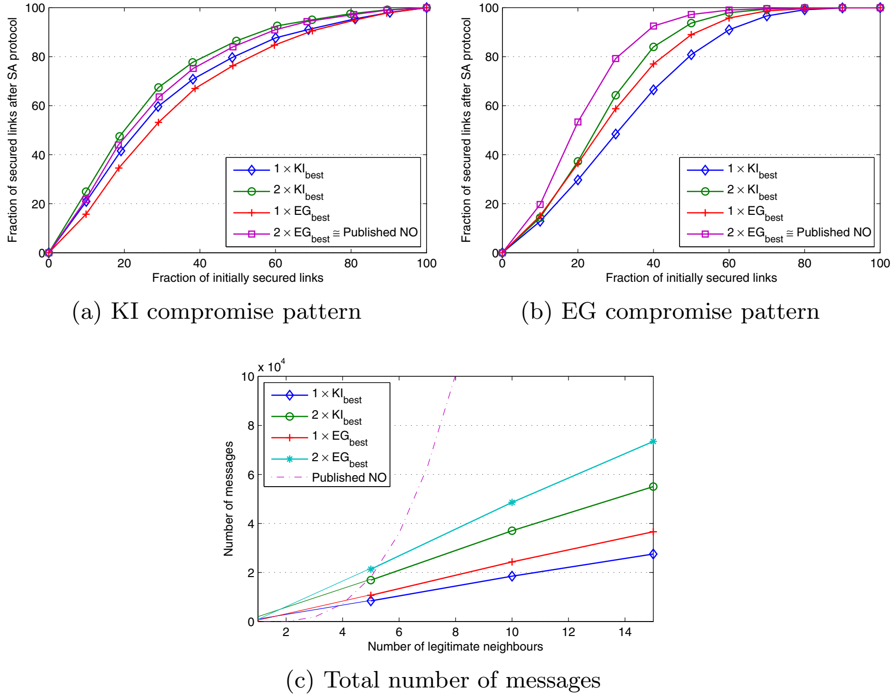
The best protocols are denoted as KI_{best} and EG_{best}. Protocol KI_{best} was discovered after 125 hours of computation in 330 641 generations (note that one generation required 1.4 seconds on a single 3000+ MHz core). EG_{best} was found after 87 hours in 165 365 generations (1.9 sec/generation on a single 3000+ MHz core). The protocols exhibit 69.12% (KI_{best}) and 60.07% (EG_{best}) secured links on average across 5 deployments they were trained for.

4.2 Performance of Evolved Secrecy Amplification Protocols

The protocols KI_{best} and EG_{best} performed very well in the deployment(s) they were trained for. However, in order to get more accurate estimate of protocols' performance, one needs to test them on different deployments. We did so by

Table 2. Parameters of LGP and network simulator used in long running experiments

		KI	EG
Simulator	Number of deployments	5	5
	Number of legitimate nodes	100	100
	Number of malicious nodes	30	-
	Average number of legitimate neighbours	9.88	10.28
	Average number of initially secured links	32.5%	30.7%
LGP	Probability of mutation	0.02	0.02
	Probability of crossover	0.00	0.00
	Size of population	15	15
	Maximum number of instructions	50	100
	Number of memory slots	12	12
	Number of generations	1 216 000	200 000
	Average time of single run on 1 CPU	467h	106h



(a) KI compromise pattern

(b) EG compromise pattern

(c) Total number of messages

Fig. 3. The performance of discovered secrecy amplification protocols. $2 \times EG_{best}$ stands for two repetitions of EG_{best}

evaluating each of the protocols in 100 random networks for each of 9 levels of compromise (10% ... 90% stepped by 10%), two compromise patterns (KI, EG) and three different average numbers of legitimate neighbours (5, 10, 15). One additional repetition of amplification was also tested. In total, we evaluated each of the protocols in 10 800 independent scenarios.

The comparison of average reached fraction of secured links after secrecy amplification in networks with 10 neighbours are shown in Figure 3. Two amplification repeats of discovered protocols have almost identical performance to our previously published node-oriented protocol [8]. Similar results were obtained also for 5 and 15 neighbours – with higher number of neighbours the differences between protocols' average performance are becoming larger.

The protocol evolved for KI and then used in EG performs significantly worse than the protocol evolved directly for EG scenario (and vice versa). This implies that the structure of the problem solved for KI and EG is different and a separate protocol should be evolved for different initial key distribution method. Note that such a result is not problematic for the network owner as used scenario (KI, EG or other) is known to him/her in advance and the owner can therefore select/evolve a corresponding protocol.

Note that the best performing group-oriented protocols are able to increase the fraction of secure links from 30% to almost 70% in KI or 80% in EG. In real deployment, one would usually like to achieve 85% or more secure links to provide a strong majority of secure links. Majority voting will then almost always outcompete a potential attacker. Such a percentage can be achieved when initial fraction of secure links is 40-50% in KI and 30-40% in EG scenario.

Figure 3c shows the total number of required messages for mentioned protocols. Note that the total number of messages in the group oriented protocols is independent of compromise scenario (the number of transmitted messages is always the same).

4.3 Robustness of Discovered Protocols

Protocol robustness against the change in underlying parameters w.r.t. parameters used during evolution is examined in this section. We focused on robustness against the change in key distribution method (KI_{best} used in EG scenario), the change in initial fraction of secure links and the change in layout of nodes in a particular deployment.

The average performance of the protocols was already shown in Figure 3. However, from the averages one can not conclude directly whether the discovered protocols are robust against changes in deployment. The EG scenario was chosen for the analysis because it allows to precisely set the fraction of initially secured links[2].

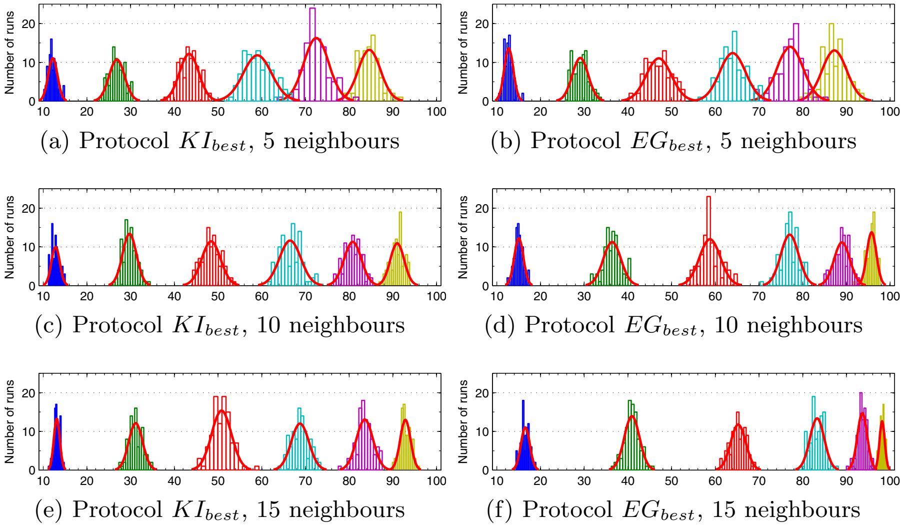
The results of the analysis are depicted in Figure 4. There are 6 histograms for each of the protocols and the average number of neighbours. Each histogram shows the distribution of resulting fraction of secure links for a given fraction of initially secured links (from left to right $10\% \dots 60\%$).

For 100 evaluations the distributions are similar to the normal one. It can be also seen that KI_{best} has slightly worse performance than EG_{best} for all tested configurations of neighbours (note that the experiment was performed for EG scenario). With a higher number of neighbours, the differences between protocols and also the performance of individual protocols are increasing.

5 Multi-criteria Optimization

Results presented in [8] and extended in Section 4.2 were obtained with the fitness function reflecting only the fraction of secure links without taking into account the number of exchanged messages. Although the overall number of exchanged messages is relatively small for group-oriented protocols, natural question is if this number can further be decreased by additional optimization.

[2] The initial fraction of secured links in KI scenario depends on the number of attacker's nodes. Particular layout of nodes in deployment slightly varies between different deployments.



(a) Protocol KI_{best}, 5 neighbours (b) Protocol EG_{best}, 5 neighbours

(c) Protocol KI_{best}, 10 neighbours (d) Protocol EG_{best}, 10 neighbours

(e) Protocol KI_{best}, 15 neighbours (f) Protocol EG_{best}, 15 neighbours

Fig. 4. Fraction of secured links reached for a given fraction of initially secured links (histograms represent from left to right $10\% \ldots 60\%$)

5.1 Weighted Fitness

The communication overhead is measured as a counterpart to fraction of messages transmitted by the protocol (lower the better) during simulation to the theoretical maximum of messages when every instruction in protocol would send one message.

We propose to combine the fraction of secure links f_1 and fraction of messages f_2 using two weighting coefficients w_1 and w_2 ($fitness = w_1 f_1 + w_2 f_2$). In order to analyze the impact of different ratios of weights, 20 independent runs were executed for multiple different weights ($90 : 10 \ldots 0 : 100$), always spanning 2 000 generations. We also set the lower bound for fraction of secured links to 50% so the evolution was forced to search only for meaningful protocols (50% can be easily achieved even by a random search).

The results of experiments are shown in Figure 5. From left to right, LGP was forced to optimize the protocols more and more for the fraction of secured links. It can be clearly seen that with any additional increase of the security in the network, the total number of messages is also non-trivially increased. However, this increase is non-linear when the weight assigned to fraction of secure links is higher than 90. Based on this result, we decided to perform another long search for a message-optimal group-oriented protocol and compare its performance with previously found KI_{best}.

5.2 Optimizing the Number of Messages

Previous experiments documented a high correlation between the number of secured links and the number of messages used. Since we were interested in

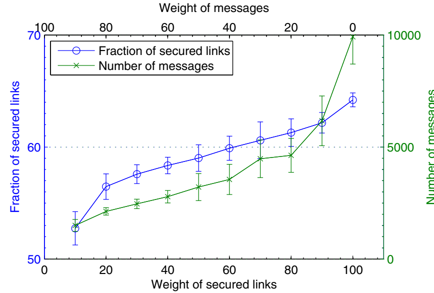


Fig. 5. Impact of criteria weights on fraction of secure links and the number of messages

protocols with a reasonable security, we decided to use the weights 90 : 10 in favour of secured links (according to Figure 5). In order to find a message-optimized protocol, we executed 20 independent runs and took the best performing protocol (denoted as KI_{msg}). It was found after 90 hours in 234 000 generations.

We evaluated the performance of KI_{msg} against KI_{best} and previously published node-oriented protocols in the setup that was described in Section 4.2. Surprisingly, the evaluation over 100 different networks and different fractions of initially compromised network showed that KI_{msg} has comparable performance in KI to previously found KI_{best} protocol. More importantly, with two amplification repeats instead of one it also exhibits almost identical performance to previously published node-oriented protocols. However, since the protocol was optimized not only for security but also for low total number of messages, KI_{msg} uses only 50% of messages to achieve similar performance.

6 Conclusions

Secrecy amplification protocols turned to be one of the most promising ways how a WSN with a significant number of compromised links can be turned into secure one for the price of additional messages exchanged. Human-designed and message intensive node-oriented version [2] of these protocols were extended by group-oriented approach in [8].

In this paper, we performed a detailed analysis of LGP in the task of evolutionary design of group-oriented protocols. By careful setting of LGP parameters, suitable setting of network simulator parameters and utilization of distributed computation, new protocols were discovered that outperform the previously published ones. The analysis of robustness of discovered protocols in scenarios different from those available during evolution confirmed that group-oriented protocols are robust against the change in the initial fraction of secure links. We have observed that these protocols are less robust against the change in selection of initial key distribution method. However, this selection is under control of the

network owner who can, therefore, select/optimize the group-oriented protocol for preferred key distribution.

Additionally, we focused on further reduction of the communication overhead. A multi-criterial optimization was conducted where not only secure links but also the number of messages was optimized. It was shown that newly found group-oriented protocols outperform the node-oriented protocols while still requiring an order of magnitude less messages, e.g., ± 1/20 in common scenarios. Future work will be devoted to applying truly multi-criteria optimization algorithms such as NSGA-II and implementation of evolved protocols on real nodes.

Lukáš Sekanina was supported by the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070. Other authors were supported by the GAP202/11/0422 and Centre of Excellence GAP202/12/G061 of the Czech Science Foundation. The experiments were supported by computational resources at Institute of Computer Science, Masaryk University.

References

1. Anderson, D.: Boinc: A system for public-resource computing and storage. In: Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing, pp. 4–10. IEEE (2004)
2. Anderson, R., Chan, H., Perrig, A.: Key infection: Smart trust for smart dust. In: ICNP 2004, pp. 206–215. IEEE (2004)
3. Bernardi, P., Sánchez, E., Schillaci, M., Squillero, G., Reorda, M.S.: An effective technique for minimizing the cost of processor software-based diagnosis in socs. In: IEEE DATE 2006: Design, Automation and Test in Europe, pp. 412–417 (2006)
4. Brameier, M., Banzhaf, W.: Linear Genetic Programming. Springer, Berlin (2007)
5. CrossBow: Telosb, http://www.willow.co.uk/TelosB_Datasheet.pdf
6. Cvrcek, D., Svenda, P.: Smart dust security-key infection revisited. Electronic Notes in Theoretical Computer Science 157(3), 11–25 (2006)
7. Eschenauer, L., Gligor, V.: A key-management scheme for distributed sensor networks, pp. 41–47 (2002)
8. Švenda, P., Sekanina, L., Matyáš, V.: Evolutionary design of secrecy amplification protocols for wireless sensor networks. In: Proceedings of the Second ACM Conference on Wireless Network Security, pp. 225–236. ACM (2009)

Automatic Design of Ant Algorithms with Grammatical Evolution

Jorge Tavares[1] and Francisco B. Pereira[1,2]

[1] CISUC, Department of Informatics Engineering, University of Coimbra
Polo II - Pinhal de Marrocos, 3030 Coimbra, Portugal
[2] ISEC, Quinta da Nora, 3030 Coimbra, Portugal
jorge.tavares@ieee.org, xico@dei.uc.pt

Abstract. We propose a Grammatical Evolution approach to the automatic design of Ant Colony Optimization algorithms. The grammar adopted by this framework has the ability to guide the learning of novel architectures, by rearranging components regularly found on human designed variants. Results obtained with several TSP instances show that the evolved algorithmic strategies are effective, exhibit a good generalization capability and are competitive with human designed variants.

1 Introduction

Ant Colony Optimization (ACO) identifies a class of robust population-based optimization algorithms, whose search behavior is loosely inspired by pheromone-based strategies of ant foraging [1]. The first ACO algorithm, Ant System (AS), was proposed by Dorigo in 1992. Since then, many different variants have been presented with differences, e.g., in the way artificial ants select components to build solutions or reinforce promising strategies. Researchers and practitioners aiming to apply ACO algorithms to a specific optimization situation are confronted with several non-trivial decisions. Selecting and tailoring the most suitable variant for the problem to be addressed and choosing the best parameter setting helps to enhance search effectiveness. However, making the right decisions is difficult and requires a deep understanding of both the algorithm's behavior and the properties of the problem to solve.

In recent years, several automatic ACO design techniques were proposed to overcome this limitation. Reports in literature range from approaches to autonomous/adaptive parameter settings to the configuration of specific algorithmic components [2]. Two examples of automatic design are the evolution of pheromone update strategies by Genetic Programming (GP) [3,4] and the synthesis of an effective ACO for multi-objective optimization [5].

In this paper we present a complete framework to evolve the architecture of a full-fledged ACO algorithm. Grammatical Evolution (GE) [6] is adopted as the design methodology, as it allows for a simple representation and modification of flexible algorithmic strategies. The grammar used by the GE algorithm defines both the potential components that can be adopted when designing an ACO algorithm and also the general structure of the optimization method. We describe

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 206–217, 2012.
© Springer-Verlag Berlin Heidelberg 2012

a set of experiments that deal with the discovery of ACO architectures for the optimization of the Traveling Salesperson Problem (TSP). The analysis of results helps to understand if the design algorithm converges to ACO architectures regularly applied to the TSP, or, on the contrary, evolves novel combinations of basic components that enhance the effectiveness of the optimization. Additionally, we address the generalization ability of the learned architectures.

The paper is structured as follows: in section 2 we present a general description of ACO algorithms. Section 3 comprises a presentation of the system used to evolve architectures, whereas section 4 contains the experimentation and analysis. Finally, in section 5 we summarize the conclusions and highlight directions for future work.

2 Ant Colony Optimization

AS was the first ACO algorithm and it was conceived to find the shortest path for the well-known TSP, although it was soon applied to different types of combinatorial optimization problems [1]. To apply an ACO algorithm to a given problem, one must first define the solution components. A connected graph is then assembled by associating each component with a vertex and by creating edges to link vertices. Ants build solutions by starting at a random vertex and by stochastically selecting edges to add new components. The probability of choosing an edge depends on the heuristic information and pheromone level of that specific path. Higher pheromone levels signal components that tend to appear in the best solutions already found by the colony. After completing a solution, ants provide feedback by depositing pheromone in the edges they just crossed. The amount of pheromone deposited is proportional to the quality of the solution. To avoid stagnation, pheromone trail levels are periodically decreased by a certain factor. Following these simple rules until a termination criterion is met, a solution to the problem will emerge from the interaction and cooperation made by the ants.

Algorithm 1. Ant Colony Optimization

```
init_parameters
init_pheromone
while not_termination do
   generate_solutions
   pheromone_update
   daemon_actions
end while
return best_solution
```

Algorithm 1 presents the pseudo-code of a general ACO algorithm. Daemon actions include a set of optional actions, e.g., the application of local search or pheromone matrix restart.

3 The Evolutionary Framework

The framework used to evolve the architecture of an ACO algorithm contains two components: a GE engine and a compiler/evaluator of ACO architectures. GE is a GP branch that allows for an efficient evolution of complete algorithmic structures. It relies on a grammar composed by a set of production rules in a Backus-Naur form, defining both the components that can appear and the overall organization of the algorithm to be evolved. The GE iterative search process is somehow decoupled from the actual programs being evolved, since individuals are codified as integer vectors. This allows for the application of a straightforward evolutionary algorithm to perform optimization. Whenever a solution needs to be evaluated, a complementary mapping process uses the grammar to decode an integer solution into a syntactically correct program (see [6] for details).

The quality of a GE individual is directly related to its ability to find good solutions for the TSP. In fitness assignment, the first step is to apply an ACO compiler to assemble a decoded GE individual into a running ACO. Then, the evolved architecture is executed and tries to solve a given TSP instance. The result of the optimization is assigned as the fitness value of that GE individual.

3.1 Grammar Definition

The grammar creates individuals containing an initialization step followed by an optimization cycle. The first stage consists in the initialization of the pheromone matrix and the selection of parameters. The main loop comprises the construction of the solutions by the ants, trail evaporation, trail reinforcement and daemon actions. Each component contains several alternatives to implement its specific task. Additionally, most blocks are optional. Different values for the parameters required by an ACO algorithm are also defined in the grammar, which allows the GE engine to automatically adapt the most suitable setting for a given architecture. The research described in this paper focus on the relevance assessment of existing ACO components and on the discovery of novel interactions that might help to enhance optimization performance. As such, daemon actions do not consider the possibility of applying local search, as the effect of this operation would dilute the influence of the remaining components.

The grammar defined is constrained in the sense that production rules enforce a global structure that is similar to existing ACO architectures. Components, e.g., solution construction, reinforcement or evaporation, can only appear in a pre-specified order and certain blocks cannot be used more than once. This is a deliberate design option. With this framework, the GE system is able to generate all main ACO algorithms: Ant System (AS), Elitist Ant System (EAS), Rank-Based Ant System (RAS), Ant Colony System (ACS) and Max-Min Ant System (MMAS). Additionally, the search space contains many other combinations of blocks that define alternative ACO algorithms. The experiments described in the next section will provide insight into the search behavior of GE in this task. We will analyze if search converges to manually designed ACO architectures regularly applied to the TSP or, on the contrary, if it discovers novel combinations of

blocks leading to the automatic design of optimization algorithms with enhanced effectiveness. The grammar for the GE engine is:

```
⟨aco⟩ ::= (aco ⟨parameters-init⟩ ⟨optimization-cycle⟩)
⟨parameters-init⟩ ::= (init ⟨pheromone-matrix-init⟩ ⟨choice-info-matrix-init⟩)
⟨pheromone-matrix-init⟩ ::= (init-pheromone-matrix ⟨trail-amount⟩)
⟨trail-amount⟩ ::= ⟨initial-trail⟩ | (uniform-trail ⟨trail-min⟩ ⟨trail-max⟩)
⟨initial-trail⟩ ::= ⟨trail-min⟩ | ⟨trail-max⟩ | (tas) | (teas ⟨rate⟩) | (tras ⟨rate⟩ ⟨weight⟩)
      | (tacs) | (tmmas ⟨rate⟩)
⟨choice-info-matrix-init⟩ ::= (init-choice-info-matrix ⟨alpha⟩ ⟨beta⟩)
⟨optimization-cycle⟩ ::= (repeat-until ⟨loop-ants⟩ ⟨update-trails⟩ ⟨daemon-actions⟩)
⟨loop-ants⟩ ::= (foreach-ant make-solution-with ⟨decision-policy⟩
      (if ⟨bool⟩ (local-update-trails ⟨decay⟩)))
⟨decision-policy⟩ ::= (roulette-selection) | (q-selection ⟨q-value⟩) | (random-selection)
⟨update-trails⟩ ::= (progn ⟨evaporate⟩ ⟨reinforce⟩)
⟨evaporate⟩ ::= (do-evaporation
      (if ⟨bool⟩ (full-evaporate ⟨rate⟩))
      (if ⟨bool⟩ (partial-evaporate ⟨rate⟩ ⟨ants-subset⟩)))
⟨reinforce⟩ ::= (do-reinforce
      (if ⟨bool⟩ (full-reinforce))
      (if ⟨bool⟩ (partial-reinforce ⟨ants-subset⟩))
      (if ⟨bool⟩ (rank-reinforce ⟨many-ants⟩))
      (if ⟨bool⟩ (elitist-reinforce ⟨weight⟩))
⟨daemon-actions⟩ ::= (do-daemon-actions
      (if ⟨bool⟩ (update-pheromone-limits ⟨update-min⟩ ⟨update-max⟩))
      (if ⟨bool⟩ (restart-check)))
⟨ants-subset⟩ ::= ⟨single-ant⟩ | ⟨many-ants⟩
⟨single-ant⟩ ::= (all-time-best) | (current-best) | (random-ant) | (all-or-current-best
      ⟨probability⟩)
⟨many-ants⟩ ::= (all-ants) | (rank-ant ⟨rank⟩)
⟨update-min⟩ ::= ⟨trail-min⟩ | (mmas-update-min)
⟨update-max⟩ ::= ⟨trail-max⟩ | (mmas-update-max ⟨rate⟩)
⟨weight⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | (n-ants) | (max-rank ⟨rank⟩)
⟨trail-min⟩ ::= 0.000001
⟨trail-max⟩ ::= 1.0
⟨alpha⟩ ::= 1 | 2 | 3
⟨beta⟩ ::= 1 | 2 | 3
⟨q-value⟩ ::= 0.7 | 0.75 | 0.8| 0.85 | 0.9 | 0.95 | 0.98 | 0.99
⟨decay⟩ ::= 0.01 | 0.025 | 0.05 | 0.075 | 0.1
⟨rate⟩ ::= 0.01 | 0.1 | 0.25 | 0.5 | 0.75 | 0.9
⟨probability⟩ ::= 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9
⟨rank⟩ ::= 5 | 10 | 25 | 50 |75
⟨bool⟩ ::= t | nil
```

3.2 Related Work

There are several efforts for granting bio-inspired approaches the ability to self adapt their strategies. On-the-fly adaptation may occur just on the parameter

settings or be extended to the algorithmic components. Grammar-based GP and GE have been used before to evolve complete algorithms, such as evolving data mining algorithms [7] and local search heuristics for the bin-packing [8].

In the area of Swarm Intelligence, there are some reports describing the self-adaptation of parameter settings (see, e.g., [9,10]). Poli et al. [11] use GP to evolve the equation that controls particle movement in Particle Swarm Optimization (PSO). Diosan and Oltean also worked on the evolution of PSO structures [12]. On the topic of ACO evolution, Runka [13] applies GP to evolve the probabilistic rule used by an ACO variant to select the solution components in the construction phase. Tavares et al. applied GP and Strongly Typed GP to evolve pheromone update strategies [3,4,14]. Finally, López-Ibáñez and Stützle synthesize existing multi-objective ACO approaches into a flexible configuration framework, which is then used to automatically design novel algorithmic strategies [5].

4 Experiments and Analysis

In this section, our purpose is to gain insight into the ability of GE to automatically design effective ACO algorithms. Selected TSP instances from the TSPLIB[1] are used to validate our approach. For all experiments we performed 30 independent runs. Results are expressed as a normalized distance to the optimum.

4.1 Learning the Architectures

In the first set of experiments, we address the ability of GE to learn ACO architectures using the previously described grammar. We adopt the strategy proposed by [3,4] to evaluate the individuals generated by the GE engine. In concrete, the ACO algorithm encoded in a solution is used to optimize a predetermined TSP instance (1 single run comprising of 100 iterations). The fitness value of that individual is given by the best solution found. This minimal evaluation methodology was adopted mainly due to efficiency reasons, since assigning fitness to an evolved architecture is a computational intensive task.

For all tests, the GE settings are: Population size: 64; Number of generations: 25; Individual size: 128 (with wrap); One-Point crossover with rate: 0.7; Standard Integer-Flip mutation with rate: 0.05; Tournament selection with tourney size: 3; Elitist strategy. Three distinct TSP training instances were selected to learn ACO architectures: eil76, pr76 and gr96 (the value represents the number of cities). Evolved individuals encode nearly all parameters required to run the ACO algorithm in the fitness assignment step. The exceptions are the size of the colony and the λ parameter for the branching factor. We set the number of ants to 10% of the number of cities (rounded to the nearest integer) and λ to 0.05.

Table 1 contains the optimization results of GE using the three training instances. The row *Best Hits* displays the number of runs where evolved architectures were able to discover the optimal solution of the training instance. Row

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Table 1. Overview of GE evolution results (for 30 runs with 25 generations)

Instances	eil76	pr76	gr96
Best Hits	20	4	12
Mean Best	0.0014 (±0.003)	0.0079 (±0.007)	0.0043 (±0.008)
Mean Ants	0.0725 (±0.062)	0.1171 (±0.073)	0.0941 (±0.074)
Mean Branching	8.58	11.21	11.24
Mean Generations	13.1	21.2	14.0

Mean Best contains the mean of the best solutions found, while *Mean Ants* gives the mean of all solutions discovered in the last generation. These two lines also display the standard deviation (in brackets). The last two rows contain the mean branching factor at the end of the optimization and the average number of generations that GE needs to reach the best solution. The low values appearing in lines *Mean Best* and *Mean Ants* confirm that evolution consistently found architectures that were able to find the optimum or near-optimum solutions. Moreover, there are no noteworthy variations in the outcomes obtained with different training instances. The branching factor indicates the degree of convergence of an ACO pheromone matrix. For the TSP, a value of 2.0 indicates full convergence. In Table 1 the mean branching is low for all training instances, although not close to full convergence. However, a closer inspection of the evolved architectures reveals that those which are able to generate better quality solutions for the TSP training instances, have a branching factor close to 2.0. As expected, higher branching values are usually present in architectures that did not perform as well.

Results from line *Best Hits* show that the instance selected for training impacts the likelihood of evolving ACO architectures that can discover the optimal solution in the evaluation step. With eil76, 20 runs were able to learn algorithms that discover the training instance optimum, whereas in pr76 only 4 runs were successful. The reason for this effect is probably related to the spatial distribution of the cities. Although all training instances have a similar dimension, the distribution pattern is not the same. Eil76 has a uniform random distribution, pr76 has a clustered distribution and gr96 is randomly distributed but not in a uniform way. It is easier for GE to evolve architectures that perform better in a uniform-random distribution than in a clustered one. The mean of generations it takes for GE to reach the best individual also corroborates this fact.

The outcomes presented in Table 1 suggest that GE is able to learn ACO architectures that effectively solve the training instances. However, the most important topic in this research is to analyze the algorithmic structure of the evolved architectures. Two questions arise: did GE discover solutions which are replicas of standard ACO algorithms, e.g., EAS or ACS? If not, did the system converge to a particular solution or to a set of different solutions? The answer to the first question is obtained by inspecting the 90 best solutions generated by the GE engine (30 best solutions for each training instance). In this set there is not an exact copy of the standard algorithms, and only 5 individuals could be considered as similar to manually designed ACO approaches (3 for the ACS, 1

Table 2. Frequency of appearance of the grammar components in best solutions evolved for each instance and average among all training instances. Evaporation and Reinforcement are not exclusive.

Components		eil76	pr76	gr96	Avg
Tau Init	Uniform Distribution	0.60	0.42	0.25	0.42
	Min	0.05	0.08	0.25	0.13
	Max	0.05	0.08	0.00	0.04
	Ant	0.05	0.00	0.25	0.10
	EAS	0.05	0.00	0.00	0.02
	RAS	0.00	0.17	0.00	0.06
	ACS	0.10	0.17	0.00	0.09
	MMAS	0.10	0.08	0.25	0.14
Selection	Roulette Selection	0.40	0.25	0.25	0.30
	Q Selection	0.60	0.75	0.75	0.70
	with Local Trails	0.30	0.17	0.00	0.16
Evaporation	Full Evaporation	0.65	0.50	1.00	0.72
	Partial Evaporation	0.50	0.33	0.25	0.36
Reinforcement	Full	0.40	0.08	0.00	0.16
	Partial	0.45	0.42	0.25	0.37
	Rank	0.50	0.17	0.00	0.22
	Elitist	0.90	0.83	1.00	0.91
Pheromone Limits		0.45	0.92	1.00	0.79
Restart		0.50	0.50	0.75	0.58

for RAS and 1 for EAS). There is then evidence that GE tends to converge to areas in the search space of ACO algorithms containing innovative strategies.

The answer to the second question can be partially obtained in Table 2, which contains the rate of appearance of the grammar components in the best solutions evolved (parameter values are not included). The rows contain all components that appear in the best solutions and are grouped by section (e.g., selection, evaporation). Some of the components are not exclusive (e.g., reinforcement), while others are (e.g., roulette vs. Q selection). The values are normalized to the interval between 0.0 and 1.0. There is a column with results obtained with each training instance and a fourth column *Avg* containing a weighted average of the frequencies by the number of *Best Hits*. An inspection of the results reveals that evolution did not converge to a unique solution. There are some variations between instances, but the general trend is to have a few alternative components appearing in the best solutions evolved by the GE in different runs. In any case, some components are frequently used to create promising strategies (e.g., elitist reinforcement and Q selection), while others are rarely considered (e.g., elitist pheromone initialization). This outcome shows that GE is able to identify a subset of components that increase the likelihood of creating effective strategies.

4.2 Validation of the Evolved Architectures

Results from the previous section show that the structure of the training instance influences the structure of the evolved architectures. It is therefore essential to validate the learned strategies in an optimization scenario. This will confirm, not

Table 3. Optimization results of selected best strategies, with 10000 iterations for 30 runs. Rows display the best solution found (B) and the mean best fitness (M) for every training instance. Bold values indicate the overall M best value.

		ei7612	ei7618	ei7621	pr7609	pr7626	pr7627	gr9603	gr9610	gr9622
att48	B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0024	**0.0003**	0.0072	0.0028	0.0069	0.0034	0.0031	0.0021	0.0020
eil51	B	0.0000	0.0000	0.0023	0.0000	0.0000	0.0023	0.0000	0.0000	0.0000
	M	0.0067	0.0023	0.0174	0.0046	0.0091	0.0186	**0.0020**	0.0045	0.0061
berlin52	B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0043	**0.0000**	0.0104	0.0055	0.0207	0.0108	0.0057	0.0053	0.0095
kroA100	B	0.0011	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0065	**0.0000**	0.0039	0.0037	0.0075	0.0044	0.0022	0.0068	0.0048
lin105	B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0043	**0.0001**	0.0014	0.0042	0.0081	0.0060	0.0016	0.0043	0.0027
gr137	B	0.0000	0.0017	0.0083	0.0000	0.0000	0.0000	0.0016	0.0000	0.0000
	M	0.0139	0.0071	0.0203	**0.0034**	0.0047	0.0053	0.0071	0.0053	0.0049
u159	B	0.0000	0.0000	0.0124	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M	0.0034	**0.0001**	0.0262	0.0039	0.0116	0.0066	0.0012	0.0021	0.0038
d198	B	0.0059	0.0067	0.0094	0.0035	0.0032	0.0001	0.0350	0.0035	0.0053
	M	0.0211	0.0103	0.0164	0.0125	0.0111	**0.0069**	0.0432	0.0082	0.0124
pr226	B	0.0000	0.0034	0.0098	0.0000	0.0000	0.0000	0.0001	0.0001	0.0000
	M	0.0042	0.0040	0.0291	**0.0022**	0.0040	0.0049	0.0053	0.0029	0.0030
lin318	B	0.0113	0.0139	0.0833	0.0086	0.0084	0.0052	0.1417	0.0079	0.0051
	M	0.0271	0.0188	0.1034	0.0199	0.0188	0.0159	0.1589	0.0186	**0.0148**

only their effectiveness, but also their ability to scale and generalize. Specifically, we aim to: i) verify how evolved architectures behave in different TSP instances; ii) access how they perform in larger instances; iii) measure the absolute optimization performance of evolved architectures, by running them as normal ACO algorithms. To focus our analysis, we randomly selected 3 of the best evolved strategies with each training instance. Strategies are identified with an ID: the first two letters and the first two digits identify the training instance; the last two digits indicate the run (e.g., architecture ei7618 was trained with instance eil76 on run 18). Ten TSP instances were selected to access the optimization performance of the evolved architectures: att48, eil51, berlin52, kroA100, lin105, gr137, u159, d198, pr226 and lin318. The number of iterations is increased to 10000 to allow a correct optimization period for larger TSP instances.

Table 3 contains the optimization results of the selected strategies on the 10 instances (one strategy in each column). For every test instance, the table displays two rows: one with the best solutions found (B) and another with the mean best fitness (M). Bold values highlight the best M value among all evolved strategies. In general, results show that evolved strategies perform well across the instances, and thus, are able to generalize. In most cases, the automatically designed algorithms found the optimal solution. The exceptions are instances d198 and lin318, where the shortest tour was never found. Still, the normalized distances to the optimum are small (e.g., pr7627 has distances 0.0001 and 0.0052 in d198 and lin318, respectively). The mean best fitness values confirm that

Table 4. Statistical differences between architectures by applying the Wilcoxon rank sum test ($\alpha = 0.01$). A capital B highlights the strategy with the best average fitness, while a lower b indicates an architecture with an equivalent performance.

	ei7612	ei7618	ei7621	pr7609	pr7626	pr7627	gr9603	gr9610	gr9622
att48		B							
eil51		b					B		
berlin52	b	B		b					
kroA100		B							
lin105		B							
gr137				B	b	b		b	b
u159		B						b	
d198						B		b	
pr226	b			B	b			b	b
lin318						b			B

evolved strategies are robust and scale well. Instances range between 48 and 318 cities, but the M value rarely exceeds 0.02 and it does not grow excessively with the size of the instances. A closer inspection of the results reveals that, despite the overall good behavior, the performance of the evolved designs is not identical. If we recall Table 2, this variation was likely to happen. There are differences in the composition of strategies evolved with a specific instance and there are differences between strategies evolved with distinct training examples. Then, it is expectable that there are performance variations when a subset of promising strategies is selected to an optimization task.

To confirm that performance variations are not the result of random events, we applied a statistical test to the data of Table 3. In concrete, for each test instance we compared the mean best fitness of the best evolved architecture with the results obtained by each of the other strategies. Table 4 contains the results of applying the Wilcoxon rank sum test with continuity correction and confidence level $\alpha = 0.01$. The capital B highlights the best architecture for a given testing instance (e.g., gr9603 for eil51). A lower b indicates that no statistically significant difference was found between the performance of two strategies (the one in the column where the b appears and the best strategy for the instance in that line); e.g., ei7618 is equivalent to gr9603 in eil51. Finally, the white spaces indicate that significant differences were found. A combined analysis of tables 3 and 4 confirms that ei7618 has the overall best performance. It obtains the lowest mean best fitness in 5 out of 10 instances and is equivalent to the best strategy in a sixth case. However, results also show that the performance of ei7618 deteriorates as the instances grow in size, suggesting that it might not scale well. On the other extreme, ei7621 exhibits a poor behavior showing that the training instance is not the only key issue to learn effective optimization architectures. Finally, the behavior of strategy gr9610 is worth noting. It never achieves the absolute best mean performance, but, in 4 instances, it obtains results statistically equivalent to the best. In the upper-half of listing 1, we present the Lisp code for the strategy ei7618 (due to space constraints, we cannot include the code from other evolved strategies).

Listing 1. Evolved architecture ei7618 and the hand-adjusted GE-best

```
(ACO-ei7618
   (INIT (INIT-PHEROMONE-MATRIX (UNIFORM-TRAIL 1.e-5 1.0))
         (INIT-CHOICE-INFO-MATRIX 1 3))
   (REPEAT-UNTIL-TERMINATION
      (FOREACH-ANT MAKE-SOLUTION-WITH (ROULETTE-SELECTION))
      (UPDATE-TRAILS
         (DO-EVAPORATION
            (FULL-EVAPORATE 0.25)
         (DO-REINFORCE
            (RANK-REINFORCE (RANK-ANT 10))
            (ELITIST-REINFORCE 1))))
      (DAEMON-ACTIONS
         (RESTART-CHECK))))

(ACO-GE-best
   (INIT (INIT-PHEROMONE-MATRIX (UNIFORM-TRAIL 1.e-5 1.0))
         (INIT-CHOICE-INFO-MATRIX 1 3))
   (REPEAT-UNTIL-TERMINATION
      (FOREACH-ANT MAKE-SOLUTION-WITH (ROULETTE-SELECTION))
      (UPDATE-TRAILS
         (DO-EVAPORATION
            (FULL-EVAPORATE 0.45))
         (DO-REINFORCE
             (PARTIAL-REINFORCE (ALL-TIME-BEST))
             (RANK-REINFORCE (RANK-ANT 10))
             (ELITIST-REINFORCE 1)))
      (DAEMON-ACTIONS
         (UPDATE-PHEROMONE-LIMITS (MMAS-UPDATE-MIN)
                                  (MMAS-UPDATE-MAX 0.01))
         (RESTART-CHECK))))
```

4.3 Comparison with Standard ACO Algorithms

To assess the absolute optimization performance of the learned architectures, we compare the results obtained by the two best evolved strategies (ei7618 and gr9610) with those achieved by the most common ACO algorithms: AS, EAS, ACS and MMAS. We include in this study two hand-adjusted architectures, based on the best strategies obtained in the learning process: GE-avg is the

Table 5. Statistical differences between selected evolved architectures, hand-adjusted architectures and standard algorithms, by applying the Wilcoxon rank sum test ($\alpha = 0.01$). A capital B highlights the strategy with the best average fitness, while a lower b indicates an architecture with an equivalent performance.

	ei7618	gr9610	GE-avg	GE-best	AS	EAS	RAS	ACS	MMAS
att48	B								
eil51	b			B					
berlin52	B			B			B		b
kroA100	B								
lin105	b			B			b		
gr137				B			b		b
u159	b			B			b		
d198		B							
pr226		b		B			b		
lin318	b	B	b				b		

strategy obtained by selecting the most frequent components identified in column *Avg* from Table 2, whereas GE-best is a hybrid of the two best evolved architectures previously mentioned (see the bottom-half of listing 1 for the Lisp code of GE-best). The standard algorithms and the hand-adjusted strategies are applied to the selected 10 test instances in same conditions described in the previous section. The parameters settings for the standard approaches follow the recommendations from [1]. The exceptions are: colony size (we keep the 10% rule); $q0$ is set to 0.7; α and β are set to 1 and 2.

The Wilcoxon test was applied to perform a statistical comparison of the mean best fitness achieved by the different methods. Results are presented in Table 5, where the symbols B and b should be interpreted as before. Overall, learned architectures and hand-adjusted strategies inspired by the evolved ones are competitive with standard variants. Nearly all best results appear in the first 4 columns, which correspond to fully evolved or fine-tuned evolved strategies. There is just a single B in the RAS column, identifying a tie with ei7618 and GE-best. Individually, the behavior of ei7618 is remarkable, since it achieves the best or an equivalent to the best performance in 7 instances. GE-best is also effective, as it achieves the absolute best performance in 6 instances. The results obtained by gr9610 are worth noting, as it achieves the best or an equivalent to the best performance on the 3 largest test instances. These outcomes suggest that gr9610 has a good scaling ability. Finally, when compared to the other evolved strategies, the results of GE-avg are poor. This confirms that designing architectures which are a linear combination of components found in successful strategies is not a guarantee of success.

5 Conclusions

We proposed a GE framework to accomplish the automatic evolution of ACO algorithms. The grammar adopted is mostly formed by components appearing in human-designed variants, although the interpretation of individuals allows for the emergence of alternative algorithmic strategies. To the best of our knowledge, this is the first work that evolves a full-fledged ACO algorithm.

Results reveal that evolution is able to discover original architectures, different from existing ACO algorithms. Best evolved strategies exhibit a good generalization capability and are competitive with human-designed variants. The system uses a constrained grammar, which undermines the possibility of evolving strategies that strongly deviate from standard ACO structures. Nevertheless, GE discovers effective novel solutions and does not converge to the standard algorithms. This suggests that the human-designed variants regularly adopted in the optimization of the TSP might be local optima in the search space of ACO algorithms. The outcomes from the last section reveal that the hybrid architecture based on the two best evolved strategies is particularly effective. This shows that GE can also act as a supplier of promising strategies that can be later combined and/or fine-tuned.

The study presented here raises important research questions. There are several design options concerning the training phase that impact the likelihood of

discovering effective and scalable strategies. In the near future we will investigate the influence played by the training instances properties (e.g., size, structure) on the discovery of good ACO algorithms. This will help us to gain insight on how to select the most favorable training environment. Also, we will address the issue of selecting the most promising strategies from the pool of best solutions. Another important research topic is the development of less constrained grammars, which might allow the evolution of ACO architectures with a higher degree of innovation. Finally, testing the approach with other problems and doing cross-problem validation will be significant steps in the effort of development evolutionary-based Ant Systems.

References

1. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
2. Eiben, A., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation 3, 124–141 (1999)
3. Tavares, J., Pereira, F.B.: Evolving Strategies for Updating Pheromone Trails: A Case Study with the TSP. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 523–532. Springer, Heidelberg (2010)
4. Tavares, J., Pereira, F.B.: Designing Pheromone Update Strategies with Strongly Typed Genetic Programming. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 85–96. Springer, Heidelberg (2011)
5. López-Ibáñez, M., Stützle, T.: Automatic Configuration of Multi-Objective ACO Algorithms. In: Dorigo, M., Birattari, M., Di Caro, G.A., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L.M., Groß, R., Şahin, E., Sayama, H., Stützle, T. (eds.) ANTS 2010. LNCS, vol. 6234, pp. 95–106. Springer, Heidelberg (2010)
6. O'Neill, M., Ryan, C.: Grammatical Evolution. Springer, Heidelberg (2003)
7. Pappa, G.L., Freitas, A.A.: Automatically Evolving Data Mining Algorithms. Natural Computing Series, vol. XIII. Springer, Heidelberg (2010)
8. Burke, E.K., Hyde, M.R., Kendall, G.: Grammatical evolution of local search heuristics. IEEE Transactions on Evolutionary Computation (2011)
9. Botee, H., Bonabeau, E.: Evolving ant colony optimization. Advances in Complex Systems 1, 149–159 (1998)
10. White, T., Pagurek, B., Oppacher, F.: ASGA: Improving the ant system by integration with genetic algorithms. In: Proceedings of the 3rd Genetic Programming Conference, pp. 610–617. Morgan Kaufmann (1998)
11. Poli, R., Langdon, W.B., Holland, O.: Extending Particle Swarm Optimisation via Genetic Programming. In: Keijzer, M., Tettamanzi, A.G.B., Collet, P., van Hemert, J., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 291–300. Springer, Heidelberg (2005)
12. Dioşan, L., Oltean, M.: Evolving the Structure of the Particle Swarm Optimization Algorithms. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 25–36. Springer, Heidelberg (2006)
13. Runka, A.: Evolving an edge selection formula for ant colony optimization. In: Proceedings of GECCO 2009, pp. 1075–1082 (2009)
14. Tavares, J., Pereira, F.B.: Towards the development of self-ant systems. In: Proceedings of GECCO 2011. ACM (2011)

Random Sampling Technique for Overfitting Control in Genetic Programming

Ivo Gonçalves[1], Sara Silva[2,1], Joana B. Melo[3], and João M.B. Carreiras[3]

[1] ECOS/CISUC, DEI/FCTUC, University of Coimbra, Portugal
[2] INESC-ID Lisboa, IST, Technical University of Lisbon, Portugal
[3] GeoDES, Tropical Research Institute (IICT), Lisbon, Portugal
icpg@dei.uc.pt, sara@kdbio.inesc-id.pt,
joana.lx.bm@gmail.com, jmbcarreiras@iict.pt

Abstract. One of the areas of Genetic Programming (GP) that, in comparison to other Machine Learning methods, has seen fewer research efforts is that of generalization. Generalization is the ability of a solution to perform well on unseen cases. It is one of the most important goals of any Machine Learning method, although in GP only recently has this issue started to receive more attention. In this work we perform a comparative analysis of a particularly interesting configuration of the Random Sampling Technique (RST) against the Standard GP approach. Experiments are conducted on three multidimensional symbolic regression real world datasets, the first two on the pharmacokinetics domain and the third one on the forestry domain. The results show that the RST decreases overfitting on all datasets. This technique also improves testing fitness on two of the three datasets. Furthermore, it does so while producing considerably smaller and less complex solutions. We discuss the possible reasons for the good performance of the RST, as well as its possible limitations.

Keywords: Genetic programming, Overfitting, Generalization.

1 Introduction

Genetic Programming (GP) is an evolutionary computation paradigm that automatically solves problems without needing to know the structure of the solution in advance [1]. One of the areas in GP that has been recently recognized as an open issue that needs to be addressed in order for GP to realize its full potential is the one of generalization [2]. Generalization is the ability of a solution to perform well on unseen cases. Achieving good generalization is one of the most important goals of any Machine Learning (ML) method such as GP. Overfitting is said to occur when a solution performs well on the training cases but poorly on the testing cases. This indicates that the underlying relationships of the whole data were not learned, and instead a set of relationships existing only on the training cases were learned, but these have no correspondence over the whole known cases.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 218–229, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Other non-evolutionary ML methods have dedicated a far larger amount of research effort to generalization than GP, although the number of publications dealing with overfitting in GP has been increasing in the past few years. Notably, in Koza [3] most of the problems presented did not use separate training and testing data sets, so performance was never evaluated on unseen cases [4]. Part of the lack of generalization efforts can be related to another issue occurring in GP - bloat. Bloat can be defined as an excess of code growth without a corresponding improvement in fitness [5]. This phenomenon occurs in GP as in most other progressive search techniques based on discrete variable-length representations. Bloat was one of the main areas of research in GP, not only because its occurrence hindered the search progress but also because it was hypothesized, in light of theories such as Occam's razor and the Minimum Description Length, that a reduced code size could lead to better generalization ability. Researchers had a common agreement that these two issues were related and that counteracting bloat would lead to positive effects on generalization. This, however, has been recently challenged. Contributions show that bloat free GP systems can still overfit, while highly bloated solutions may generalize well [6]. This leads to the conclusion that bloat and overfitting are in most part two independent phenomena. In light of this finding, new approaches to improve GP generalization ability are in need, particularly those not based on merely biasing the search toward shorter solutions.

In this work we study the potential of a simple technique, the Random Sampling Technique (RST), to control overfitting in hard real world applications. Recently used with success on a simple benchmark problem [31], the RST is based on the idea of never using the entire training set in any given generation of the search process.

Section 2 reviews the state of the art of generalization in GP. Section 3 describes the RST and the experimental settings. Section 4 presents and discusses the results, advancing ideas for future work. Section 5 concludes.

2 State of the Art

The most common approaches to reducing overfitting in GP are those based on biasing the search toward shorter solutions. Becker and Seshadri [7] proposed adding a complexity penalty factor to the fitness function. Mahler et al. [8] explored to what extent Tarpeian bloat control affects GP generalization ability. Gagné et al. [9] tested the application of parsimony pressure. Cavaretta and Chellapilla [10] used a low-complexity-bias algorithm that uses a modification in the fitness function meant to penalize larger individuals. Zhang et al. [11] also addressed the relationship between size and generalization performance by using a fitness function with two components: fitting error and size.

More recent approaches bias the search process toward less complex solutions. In these approaches complexity is not simply defined as solution size. Vladislavleva et al. [12] proposed a complexity measure called order of nonlinearity. This measure adopts the notion of the minimal degree of the best-fit polynomial, approximating an analytical function with a certain precision. The main objective

behind the proposed complexity measure is to favor smooth and extrapolative behavior of the response surface and to discourage highly nonlinear behavior, which is unstable toward minor changes in inputs and is dangerous for extrapolation. Vanneschi et al. [13] proposed a functional complexity measure based on the classic mathematical concept of curvature. Informally, the curvature of a function can be defined as the amount by which its geometric representation deviates from being straight. This complexity measure expresses the complexity of a function by counting the number of different slopes.

Also recently, approaches based on similarities between solutions have started to appear. Uy et al. [15] proposed a Semantic Similarity based Crossover approach which is based on the Sampling Semantics Distance between two trees (or subtrees), which is calculated by choosing N random points (fitness cases) and calculating the mean absolute difference between each corresponding points on the two trees. The authors argue that the exchange of subtrees is most likely to be beneficial if the two subtrees are not too similar or too dissimilar. Vanneschi and Gustafson [16] proposed avoiding solutions similar to already known overfitted solutions. The proposed method (repGP) keeps a list of overfitting individuals (called repulsors) and prevents any new individual to enter the next generation if they are similar to any of the known repulsors.

Various other approaches were proposed. A simple and elegant idea was proposed by Da Costa and Landry [17]. The idea is to relax the training set by allowing a wider definition of the desired solution which translates into considering not only the desired output y correct but allow a more broader range to be considered, i.e. allow any output in the range [$ymin$, $ymax$]. Chan et al. [18] proposed a statistical method called Backward Elimination that works by eliminating insignificant terms in polynomials models such as those produced by GP. Nikolaev et al. [19] proposed several techniques to balance the statistical bias and variance. In the context of financial applications, Chen and Kuo [20] proposed a measure of degree of overfitting based on the extracted signal ratio. Foreman and Evett [21] proposed Canary Functions, where the idea is to measure overfitting during the run by using a validation set. When overfitting starts to occur the search process is stopped. Vanneschi et al. [22] argued that using GP with a multi-optimization approach can enhance the generalization ability of the resulting solutions. This approach uses two other criteria besides the traditional sum of errors. These are: the correlation between outputs and targets (to maximize) and the diversity of pairwise distances between outputs and targets (to minimize). Robilliard and Fonlupt [23] applied a method called Backwarding that goes back as much as needed in the evolution process until the point that overfitting is not yet very relevant. This is achieved by saving two copies of the solutions: one copy for the best solution on the training set and another copy for the best solution on the validation set. At the end of the GP process the best saved solution for the validation set is returned. Finally, in the context of the Compiling Genetic Programming System, Banzhaf et al. [24] showed the positive influence of the mutation operator in generalization ability.

Although these and a number of other works have addressed the issue of overfitting in GP, they appear as a set of isolated efforts scattered along the years and among different applications. Nevertheless, as GP matures and slowly becomes a mainstream ML approach, also the overfitting issue is slowly becoming a central research subject.

3 Experiments

This section describes the datasets used, the Random Sampling Technique and the experimental parameters used.

3.1 Datasets

Experiments are conducted on three multidimensional symbolic regression real world datasets, the first two on the pharmacokinetics domain and the third one on the forestry domain.

Toxicity. The goal of this application is to predict, in the context of a drug discovery study, the median lethal dose (LD50) of a set of candidate drug compounds on the basis of their molecular structure. LD50 refers to the amount of compound required to kill 50% of the considered test organisms (cavies). Reliably predicting this and other pharmacokinetics parameters would permit to reduce the risk of late stage research failures in drug discovery, and enable to decrease the number of experiments and cavies used in pharmacological research [25]. The LD50 dataset consists of 234 instances, where each instance is a vector of 626 molecular descriptor values identifying a drug. This dataset is freely available at `http://personal.disco.unimib.it/Vanneschi/toxicity.txt`.

Plasma Protein Binding. As in the toxicity application, also here the goal is to predict the value of a pharmacokinetics parameter of a set of candidate drug compounds on the basis of their molecular structure, this time the plasma protein binding level (%PPB). %PPB quantifies the percentage of the initial drug dose that reaches the blood circulation and binds to the proteins of plasma. This measure is fundamental for good pharmacokinetics, both because blood circulation is the major vehicle of drug distribution into human body and since only free (unbound) drugs can permeate the membranes reaching their targets [25]. This dataset consists of 131 instances, where each instance is a vector of 626 molecular descriptor values identifying a drug.

Biomass. The objective of this application is to estimate forest above-ground biomass (AGB) as a function of several metrics derived from synthetic aperture radar (SAR) data acquired by sensors on orbital platforms. Mapping and understanding the spatial distribution of forest AGB is an important and challenging task [26,27,28]. As it relates to the carbon stocks of a given ecosystem, these maps can be used to monitor forests and capture national deforestation processes, forest degradation, and the effects of conservation actions, sustainable management and enhancement of carbon stocks. The dataset is composed of 112 field measurements of forest AGB and corresponding 8 SAR metrics used to model AGB. This dataset has never been used in any GP studies.

3.2 Random Sampling Technique

The Random Sampling Technique (RST) has been previously used to improve the speed of a GP run [29], however in [30] it was used to reduce overfitting in a classification task in the context of software quality assessment. In the RST, the training set is never entirely used in the search process. Instead, at each generation, a random subset of the training data is chosen and evolution is performed taking into account the fitness of the solutions in this subset. This implies that only individuals that perform well on various different subsets will remain in the population. It is expected that, since these surviving individuals perform reasonably well on different subsets, they have captured the underlying relationships of the data instead of overfitting it. This work is a continuation of a previous work [31] that was done with the RST on a simple benchmark problem. In the mentioned work we have proposed a more flexible approach to the RST. Firstly, the size of the random subset can be defined as any percentage of the training set. Secondly, the rate at which a new random subset is chosen can be defined as either being at each N generations or as a percentage of the total number of generations. These two RST parameters are respectively labelled as Random Subset Size (RSS) and Random Subset Reset (RSR). In this extended approach they can be defined as any value, as opposed to their static nature in [30]. In this paper we build upon the previous work by exploring the RST on real world datasets with the best configuration found in that work (both RSS and RSR set to value 1, i.e. in each generation a single new random sample is chosen). Standard GP is used as the baseline for comparison.

3.3 Parameters and Statistical Tests

The experimental parameters used are provided in Table 3.3. Furthermore, crossover and mutation points are selected with uniform probability. Fitness is calculated as the Root Mean Squared Error (RMSE) between outputs and targets.

Statistical significance of the null hypothesis of no difference was determined with pairwise Kruskal-Wallis non-parametric ANOVAs at p=0.05. A non-parametric ANOVA was used because the data is not guaranteed to follow a normal distribution. For the same reason, the median was preferred over the mean in all the evolution plots shown in the next section. The median is also more robust to outliers.

4 Results and Discussion

This section presents and discusses the results achieved. For the remainder of this paper, the terms training and testing fitness are to be interpreted in the following way: training fitness is the fitness of the best individual in the training set; testing fitness is the fitness of that same individual in the testing set. For the purpose of further comparisons we have defined a simple overfitting measure. According

Table 1. GP parameters used in the experiments

Runs	30
Population	500
Generations	100
Training - Testing division	70% - 30%
Crossover operator	Standard subtree crossover, probability 0.9
Mutation operator	Standard subtree mutation, probability 0.1, new branch maximum depth 6
Tree initialization	Ramped Half-and-Half [1], maximum depth 6
Function set	+, -, *, and /, protected as in [1]
Terminal set	Input variables, no constants
Selection for reproduction	Lexicographic Parsimony Pressure [32], tournaments of size 10
Elitism	Replication rate 0.1, best individual always survives
Maximum tree depth	17

to this measure, overfitting is simply calculated as the difference between testing and training fitness. This implies that if training fitness is better than testing fitness then overfitting is occurring, i.e. the measure retrieves a positive value. Negative values are allowed and occur when testing fitness is better than training fitness. This is rather uncommon but can still happen. In the evolution plots we have chosen not to take the absolute value of the difference between both fitness values as not to lose this potentially interesting information. For the statistical tests we take the absolute values since what we want is overfitting as close to zero as possible and not to have negative overfitting. This also prevents the unwanted effect of compensation between positive and negative overfitting values. Tree size is calculated as the number of nodes of a solution. Complexity is calculated based on the notion of curvature of a function as in [13]. The evolution plots present the results based on the median of the fitness, overfitting, tree size and complexity of the best individuals at each generation over 30 runs.

4.1 Results

The fitness and overfitting plots for Standard GP and RST can be found in Figure 1. The corresponding tree size and complexity plots can be found in Figure 2. The statistical results comparing training and testing fitness, overfitting, tree size and complexity between both techniques can be found in Table 4.1. In this table, s+ indicates that the RST is statistically better than Standard GP, while s- indicates the opposite, and $\sim$ indicates that no statistically significant difference was found. As we can see in the table, the RST achieves statistically better testing fitness on LD50 and on %PPB. On AGB the difference is not statistically significant. In training fitness, Standard GP is statistically better on

Table 2. Statistical results

		LD50	**%PPB**	**AGB**
Training Fitness		s-	s-	s-
		0.000000	0.000000	0.000000
Testing Fitness		s+	s+	∼
		0.000572	0.000058	0.208871
Overfitting		s+	s+	s+
		0.000095	0.000000	0.000001
Tree Size		s+	s+	s+
		0.000000	0.000002	0.000000
Complexity		s+	s+	s+
		0.000000	0.000001	0.000000

all datasets. In overfitting the RST achieves statistically lower overfitting than Standard GP on all datasets. The same statistical significance happens for tree size and complexity also on all datasets.

Looking at the evolution plots in Figure 1 we can see, across all the datasets, a constantly widening gap between training and testing fitness in Standard GP, which means that overfitting is occurring. The widening of this gap is particularly fast on the LD50 dataset. This gap is what our overfitting measure represents. We can see that the overfitting increase is steeper on the LD50 dataset, with AGB being the second most overfitted dataset and %PPB the least overfitted of the three, although still with the referred widening gap present. RST, however, can maintain a much smaller gap between training and testing fitness. This gap is also much more constant across all generations. LD50 is the dataset where the gap is smaller and in fact very close to zero. %PPB comes next in regard to overfitting and ABG is comparatively the most overfitted of the datasets with the RST.

Comparing the training fitness values of the Standard GP with the ones of the RST, we can see that Standard GP learns the training data much faster. This was expected, since in Standard GP each generation is allowed to see many more fitness cases than RST, that sees only one. However, the testing fitness values reveal that Standard GP is merely overfitting and not actually learning the underlying relationships of the whole known instances, a fact that becomes even clearer when looking at the corresponding overfitting plots. Despite being a slow learner, RST compensates in terms of overfitting as it maintains considerably lower values than Standard GP.

Besides achieving better testing fitness in two out of three datasets, and less overfitting in all datasets, the RST does so while producing smaller and less complex solutions. We can see from the tree size and complexity plots in Figure 2 that these present a similar behavior to the overfitting plots, i.e. RST presents rather constant values while Standard GP presents a somewhat constant increase in all values (overfitting, tree size and complexity). We have not yet explored

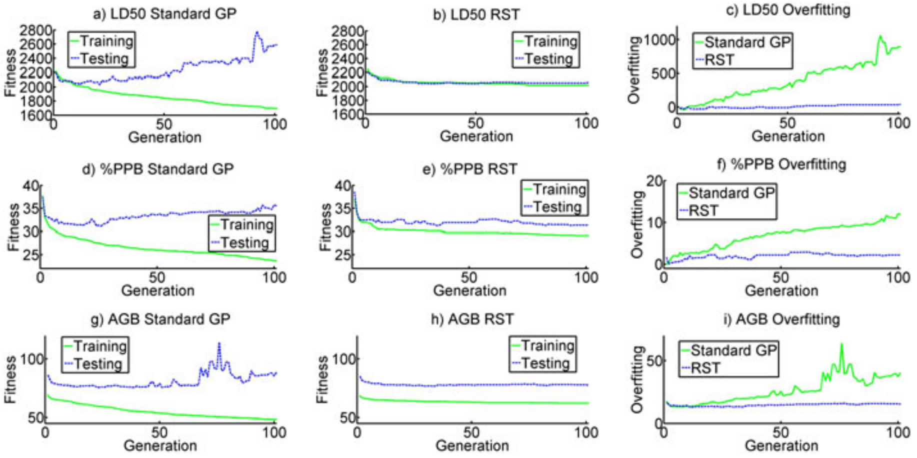


Fig. 1. Standard GP and RST fitness and overfitting evolution plots for: LD50 a) through c), %PPB d) through f) and AGB g) through i)

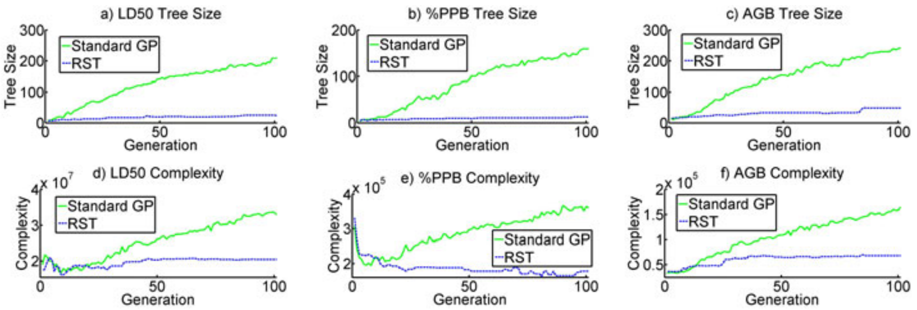


Fig. 2. Tree size and complexity evolution plots for: LD50 a) and d), %PPB b) and e) and AGB c) and f)

the relationship between these three measurements, or between these and the amount of bloat. At least one study has attempted it in the LD50 dataset [14] but did not reach solid conclusions. From the practical point of view, it is sufficient for now to state that the RST reveals clear advantages in the three real world applications studied here.

4.2 Discussion

One important point that the RST made us realize is that, in order to overfit, a solution does not necessarily need to find a set of relationships that occur across most of the fitness cases in the training set. We believe that, in most cases, the solutions that overfit are simply overfitting a few instances of the training set, but this is enough to achieve lower error (better fitness) than a solution that learns

the general trend of the training data without ever learning specific fitness cases so well. A solution that perfectly maps (i.e. has zero error) the relationship of a small number of fitness cases can have a much better fitness, thus higher chance of survival, than other solutions not as much overfitted. This can be one of the roots of the problem, since these overfitted, but low error solutions will likely remain in the population for a long time. On the other hand, RST has shown us that by using only one fitness case at the time and changing it frequently we can ultimately avoid this pitfall of the traditional approach.

Figure 3 exemplifies the abovementioned situation. In this figure we have a target and three possible solutions. In both examples solutions 2 and 3 have lower RMSE error than solution 1. However, although shifted up from the target, solution 1 represents the pattern of the target more precisely than solutions 2 or 3 and thus is more desirable to keep in the population. We can see that solutions 2 and 3 present a much higher risk of overfitting. However, in Standard GP they would be preferred over solution 1, consequently filling up the population with many similar solutions presenting the same overfitting risks. In RST, solution 1 is not necessarily discarded when compared to solutions 2 and 3, as the chosen instance for the fitness calculation in each generation can be any, and hence an instance where solutions 2 and 3 fail to fit can be chosen. This results in keeping solution 1 in the population and possibly allowing the search to find other more general solutions. In fact, a preliminary exploration of the population characteristics of both Standard GP and RST, has revealed that the genotypical diversity is generally higher with RST (with statistical significance), in particular among the best ranked individuals of the population. Additional observations have also revealed that the solutions produced by the RST tend to be smoother than the ones produced by Standard GP, much like the examples presented in Figure 3. Further investigation of these themes may help us develop better methods to control overfitting in the future.

Another issue that deserves further attention is that of the amount of search that is allowed to the RST when compared to Standard GP. One of the most interesting facts observed in the results is the evolution of the RST testing and training fitness in all datasets. These show that the RST is able to continually improve training fitness while not degrading the testing fitness. The improvement of its training fitness is much smaller than the improvement achieved by Standard GP, but the fact that the testing fitness and the overfitting values are better compensate for this fact. These results hint that the RST is indeed learning the underlying relationships of the whole known data (testing and training sets) instead of simply overfitting the training data. Even in the final generations the overfitting of the RST is close to zero. However, care must be taken when drawing conclusions, since it is not clear whether these low overfitting values can be kept if the runs are allowed to continue for more generations. We have not performed such experiments yet, but these will bring further insight on the full potential of the RST. If indeed the RST is learning the underlying relationships of the whole data, then similar overfitting values should occur in the extended runs. If not, we may have to conclude that the RST is only delaying the occurrence of overfitting,

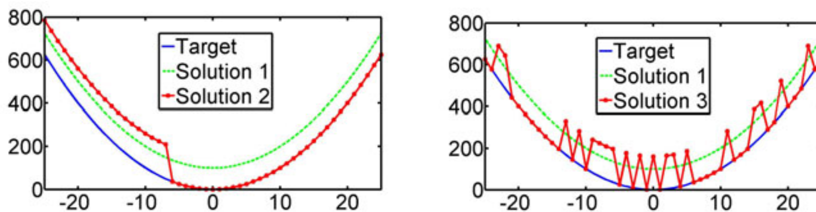


Fig. 3. A possible target and three different candidate solutions

simply because it sees very little of the training set in each generation. Note that, in a run of only 100 generations, it is not even possible for RST (with the settings used, RSS=1) to see all the instances of the training set in the LD50 dataset.

5 Conclusions

In this work we have studied the potential of the RST to control overfitting in hard real world applications. In the RST, the training set is never entirely used in the search process. Instead, at each generation, a random subset of the training data is chosen and evolution is performed taking into account the fitness of the solutions in this subset. In two multidimensional symbolic regression problem from the pharmacokinetics domain and one from the forestry domain, the RST was able to continually improve training fitness while not degrading the testing fitness, resulting in much lower overfitting values than the ones observed for Standard GP. Furthermore, the RST did so while producing considerably smaller and less complex solutions. From these facts we were able to claim that the RST, with the settings used in this work, reveals clear advantages in the three real world applications studied here.

Generalization has only recently been recognized as an important open issue of GP. Another contribution of this work is a brief summary of the published literature on this subject, which until now has been made of a series of isolated efforts scattered along the years and among different applications. While discussing the possible reasons for the good performance of the RST, as well as its possible limitations, we have advanced a few ideas for future research that we hope may help to build new methods to control overfitting in the future, thus contributing to the advancement of the state of the art of this subject.

Acknowledgments. This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds. The authors acknowledge projects EnviGP (PTDC/EIA-CCO/103363/2008, funded by FCT, Portugal) and CarboVeg GB (funded by the Ministry of Environment, Portugal), and also the Secretary of State of the Environment and Sustainable Development (SEAD, Guinea-Bissau), and Maria José Vasconcelos, Project PI, IICT, Lisbon (Portugal).

References

1. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming (With contributions by J.R. Koza) (2008), http://lulu.com, http://www.gp-field-guide.org.uk
2. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open Issues in Genetic Programming. Genetic Programming and Evolvable Machines 11, 339–363 (2010)
3. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
4. Kushchu, I.: An Evaluation of Evolutionary Generalisation in Genetic Programming. Artificial Intelligence Review 18, 3–14 (2002)
5. Silva, S., Costa, E.: Dynamic Limits for Bloat Control in Genetic Programming and a review of past and current bloat theories. Genetic Programming and Evolvable Machines 10(2), 141–179 (2009)
6. Vanneschi, L., Silva, S.: Using Operator Equalisation for Prediction of Drug Toxicity with Genetic Programming. In: Lopes, L.S., Lau, N., Mariano, P., Rocha, L.M. (eds.) EPIA 2009. LNCS, vol. 5816, pp. 65–76. Springer, Heidelberg (2009)
7. Becker, L.A., Seshadri, M.: Comprehensibility and Overfitting Avoidance in Genetic Programming for Technical Trading Rules. Technical report, Worcester Polytechnic Institute (2003)
8. Mahler, S., Robilliard, D., Fonlupt, C.: Tarpeian Bloat Control and Generalization Accuracy. In: Keijzer, M., Tettamanzi, A.G.B., Collet, P., van Hemert, J., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 203–214. Springer, Heidelberg (2005)
9. Gagné, C., Schoenauer, M., Parizeau, M., Tomassini, M.: Genetic Programming, Validation Sets, and Parsimony Pressure. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 109–120. Springer, Heidelberg (2006)
10. Cavaretta, M.J., Chellapilla, K.: Data Mining using Genetic Programming: The implications of parsimony on generalization error. In: Proceedings of the 1999 IEEE Congress on Evolutionary Computation, pp. 1330–1337. IEEE Press (1999)
11. Zhang, B.-T., Mühlenbein, H.: Balancing Accuracy and Parsimony in Genetic Programming. Evolutionary Computation 3(1), 17–38 (1995)
12. Vladislavleva, E.J., Smits, G.F., den Hertog, D.: Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. IEEE Transactions on Evolutionary Computation 13(2), 333–349 (2009)
13. Vanneschi, L., Castelli, M., Silva, S.: Measuring Bloat, Overfitting and Functional Complexity in Genetic Programming. In: Proceedings of GECCO 2010, pp. 877–884. ACM Press (2010)
14. Trujillo, L., Silva, S., Legrand, P., Vanneschi, L.: An Empirical Study of Functional Complexity as an Indicator of Overfitting in Genetic Programming. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 262–273. Springer, Heidelberg (2011)
15. Nguyen, Q.U., Nguyen, T.H., Nguyen, X.H., O'Neill, M.: Improving the Generalisation Ability of Genetic Programming with Semantic Similarity based Crossover. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 184–195. Springer, Heidelberg (2010)
16. Vanneschi, L., Gustafson, S.: Using Crossover Based Similarity Measure to Improve Genetic Programming Generalization Ability. In: Proceedings of GECCO 2009, pp. 1139–1146. ACM Press (2009)
17. Da Costa, L.E., Landry, J.-A.: Relaxed Genetic Programming. In: Proceedings of GECCO 2006, pp. 937–938. ACM Press (2006)

18. Chan, K.Y., Kwong, C.K., Chang, E.: Reducing Overfitting in Manufacturing Process Modeling using a Backward Elimination Based Genetic Programming. Applied Soft Computing 11(2), 1648–1656 (2011)
19. Nikolaev, N., de Menezes, L.M., Iba, H.: Overfitting Avoidance in Genetic Programming of Polynomials. In: Proceedings of the 2002 IEEE Congress on Evolutionary Computation, pp. 1209–1214. IEEE Press (2002)
20. Chen, S.-H., Kuo, T.-W.: Overfitting or Poor Learning: A Critique of Current Financial Applications of GP. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 34–46. Springer, Heidelberg (2003)
21. Foreman, N., Evett, M.: Preventing overfitting in GP with canary functions. In: Proceedings of GECCO 2005, pp. 1779–1780. ACM Press (2005)
22. Vanneschi, L., Rochat, D., Tomassini, M.: Multi-optimization improves genetic programming generalization ability. In: Proceedings of GECCO 2007, p. 1759. ACM Press (2007)
23. Robilliard, D., Fonlupt, C.: Backwarding: An Overfitting Control for Genetic Programming in a Remote Sensing Application. In: Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M. (eds.) EA 2001. LNCS, vol. 2310, pp. 245–254. Springer, Heidelberg (2002)
24. Banzhaf, W., Francone, F.D., Nordin, P.: The Effect of Extensive Use of the Mutation Operator on Generalization in Genetic Programming using Sparse Data Sets. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 300–309. Springer, Heidelberg (1996)
25. Archetti, F., Messina, E., Lanzeni, S., Vanneschi, L.: Genetic programming for computational pharmacokinetics in drug discovery and development. Genetic Programming and Evolvable Machines 8(4), 17–26 (2007)
26. Baccini, A., Laporte, N., Goetz, S.J., Sun, M., Dong, H.: A first map of tropical Africa's above-ground biomass derived from satellite imagery. Environmental Research Letters 3, 045011 (2008)
27. Lucas, R., Armston, J., Fairfax, R., Fensham, R., Accad, A., Carreiras, J., Kelley, J., Bunting, P., Clewley, D., Bray, S., Metcalfe, D., Dwyer, J., Bowen, M., Eyre, T., Laidlaw, M., Shimada, M.: An Evaluation of the ALOS PALSAR L-Band Backscatter-Above Ground Biomass Relationship Queensland, Australia: Impacts of Surface Moisture Condition and Vegetation Structure. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 3(4), 576–593 (2010)
28. Saatchi, S.S., Harris, N.L., Brown, S., Lefsky, M., Mitchard, E.T.A., Salas, W., Zutta, B.R., Buermann, W., Lewis, S.L., Hagen, S., Petrova, S., White, L., Silman, M., Morel, A.: Benchmark map of forest carbon stocks in tropical regions across three continents. Proceedings of the National Academy of Sciences 108(24), 9899–9904 (2011)
29. Gathercole, C., Ross, P.: Dynamic Training Subset Selection for Supervised Learning in Genetic Programming. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN 1994. LNCS, vol. 866, pp. 312–321. Springer, Heidelberg (1994)
30. Liu, Y., Khoshgoftaar, T.: Reducing Overfitting in Genetic Programming Models for Software Quality Classification. In: Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering, pp. 56–65. IEEE Press (2004)
31. Gonçalves, I., Silva, S.: Experiments on Controlling Overfitting in Genetic Programming. In: 15th Portuguese Conference on Artificial Intelligence (to appear)
32. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: Proceedings of GECCO 2002, pp. 829–836. Morgan Kaufmann (2002)

Evolutionary Operator Self-adaptation with Diverse Operators

MinHyeok Kim[1], Robert Ian (Bob) McKay[1], Dong-Kyun Kim[2], and Xuan Hoai Nguyen[3]

[1] Seoul National University, Korea
[2] University of Toronto, Canada
[3] Hanoi University, Vietnam
{rniritz,rimsnucse,dkkim1004,nxhoai}@gmail.com
http://sc.snu.ac.kr

Abstract. Operator adaptation in evolutionary computation has previously been applied to either small numbers of operators, or larger numbers of fairly similar ones. This paper focuses on adaptation in algorithms offering a diverse range of operators. We compare a number of previously-developed adaptation strategies, together with two that have been specifically designed for this situation. Probability Matching and Adaptive Pursuit methods performed reasonably well in this scenario, but a strategy combining aspects of both performed better. Multi-Arm Bandit techniques performed well when parameter settings were suitably tailored to the problem, but this tailoring was difficult, and performance was very brittle when the parameter settings were varied.

Keywords: Adaptive operator selection, Adaptive pursuit, Probability matching, Multi-armed bandit, Evolutionary algorithm.

1 Introduction

Operator parameters have been a focus of study from the earliest evolutionary algorithms. While static settings were used at first, it was quickly recognised that operator rates should vary from problem to problem, or even within a run, so adaptive mechanisms were introduced [14]. Recently, there has been renewed interest in mechanisms for adapting operator rates, with a flurry of significant advances in understanding and achievements in improved performance [12,16,3]. However these studies were generally limited either to a small number – two or three – operators, or to operators with fairly similar effects.

Meanwhile, there have been extensive studies of suitable evolutionary operators; many have been shown effective for specific problems. While there are some rules of thumb to match operators to the fitness landscape, they require knowledge that may not be available for a new problem. Operator rate adaptation offers an alternative, where the system has many operators, selecting among them based on effectiveness. This shifts the emphasis to adapting rates in parallel for a diverse range of operators, the focus of this paper. We study the performance effects of six different adaptation strategies in a Genetic Algorithm (GA) with eight diverse operators, and in Genetic Programming (GP) with seven.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 230–241, 2012.
© Springer-Verlag Berlin Heidelberg 2012

In this paper, section 2 discusses previous work on operator adaptation. Section 3 introduces some new variants and describes the experimental framework, while section 4 shows the results. In section 5, we discuss their implications, and we conclude in section 6 with the assumptions and limitations of the work, a summary of the conclusions, and pointers for further work.

2 Background

2.1 Operator Parameter Adaptation

Operator adaptation has been studied from the very early days of evolutiionary computation, beginning with Schwefel's one-fifth success rule [14]. More recently, other approaches have come to the fore: Probability Matching (PM [5,18,7]), Adaptive Pursuit (AP [17,15]) and Multi-Armed Bandits (MAB [2,4,3]).

Probability Matching. PM matches the operator probabilities to their relative reward from the environment (Eq. 1). The algorithm parameters are the number of operators K, a probability vector P giving the probability of applying each operator, a quality vector Q measuring the reward R obtained from previous use of the operator, a minimum allowable probability P_{min}, and a decay parameter α determining responsiveness to changes in reward.

$$\begin{aligned} P_a(t+1) &= P_{min} + (1 - K \cdot P_{min}) \frac{Q_a(t)}{\sum_{i=1}^{K} Q_i(t)} \\ Q(t+1) &= Q(t) + \alpha(R(t) - Q(t)) \end{aligned} \tag{1}$$

Adaptive Pursuit. AP is quite similar to PM. It differs in applying a greater probability increment to the most effective operator, correspondingly decreasing the probability of other operators (Eq. 2). It thus provides faster response to change in reward. There are two additional parameters, the maximum allowable probability P_{max}, and the learning rate for the best operator β.

$$\begin{aligned} a^* &= argmax\{Q_i, i = 1...K\} \\ P_{a^*}(t+1) &= P_{a^*}(t) + \beta(P_{max} - P_{a^*}(t)) \\ P_a(t+1) &= P_a(t) + \beta(P_{min} - P_a(t)) \qquad \text{for} \quad a \neq a^* \end{aligned} \tag{2}$$

Multi-Armed Bandits. MAB chooses one operator which maximises the function in equation 3, based on the Upper Confidence Bound (UCB) algorithm [1], which uses two terms: exploitation and exploration. The exploitation term calculates $\hat{P_{i,t}}$, the average reward of the operator up to time t, while the exploration term measures $n_{i,t}$, how often the operator is selected. Since the reward range is unknown a priori, a scaling factor C is used to balance the two terms.

$$UCB_i(t) = \hat{P_{i,t}} + C\sqrt{\frac{\log \sum_k n_{k,t}}{n_{i,t}}} \tag{3}$$

This Static MAB (S-MAB) is appropriate when reward changes slowly; however changes in the evolving population may lead to step changes in the reward. Dynamic MAB (D-MAB) uses the Page-Hinkley (PH) test [13], with parameters δ and λ, to determine when to reset the MAB log.

2.2 Evolutionary Algorithms

The structure of the search space may affect the relative performance of adaptation mechanisms; hence we use two different evolutionary systems, real-coded Genetic Algorithms (GA) with isotropic fixed-size neighbourhoods, and Genetic Programming (GP) whose neighbourhood structure varies across the search space. We use a specific form of GP, TAG-Grammar-Guided GP (TAG3P), which supports a range of single- and dual-parent operators [6].

3 Methods and Experiments

3.1 New Adaptive Mechanisms

PM and AP have previously been showed to have good performance [16,2,4,3,9], but they were not specifically designed for multiple, highly diverse operators. We introduce some variants which may be better suited to such environments.

Power Probability Matching. (PPM) is a variant of PM. PM may not work well when operator rewards are very similar, as often occurs when there are many operators. PPM amplifies the differences through exponentiation (Eq. 4):

$$P_a(t+1) = P_{min} + (1 - K \cdot P_{min}) \frac{Q_a(t)^K}{\sum_{i=1}^{K} Q_i(t)^K} \tag{4}$$

Adaptive Probability Matching. (APM) combines the algorithms of AP and PM. AP divides operators into two groups: the most effective one and the rest; it increases the rate of the former, but decreases the rates of the others equally. That is, it ignores the relative rewards of the other operators. APM follows AP in increasing the rate of the most effective operator as in AP, but it then divides the remaining operator rate amongst the other operators according to their relative reward, as in PM (Eq. 5):

$$\begin{aligned} a^* &= argmax\{Q_i, i = 1...K\} \\ P_{a^*}(t+1) &= P_{a^*}(t) + \beta(P_{max} - P_{a^*}(t)) \\ P_a(t+1) &= P_{min} + (1 - P_{a^*}(t+1) - (K-1) \cdot P_{min}) \frac{Q_a(t)}{\sum_{i=1, i \neq a}^{K} Q_i(t)} \\ &\text{for} \quad a \neq a^* \end{aligned} \tag{5}$$

Table 1. Problem Definitions and Evolutionary Parameters

Problem	**ParameterFitting**	**Quintic**	**Sextic**	**Trigonometric**
Target Function	Process Model	$x^5 - 2x^3 + x$	$x^6 - 2x^4 + x$	$\cos(2x)$
Fitness Cases	938 samples over 12 years	50 Random Points from $[-1, 1]$		20 Random Points from $[0, 2\pi]$
Minimisation Objective	RMSE of cases	MAE of cases		
Error Bound ϵ		0.01		
Success Predicate		Error $< \epsilon$ on all fitness cases		
Chromosome	15 real values	Tree 2 . . . 40 nodes		
Function Set		$+, -, \times, \div$		$+, -, \times, \div, \sin$
Terminal Set		X		$X, 1$
Runs	100	100		
Generations	100	50		
Population Size	100	1000		
Tournament Size	4	3		

3.2 Test Problems

To compare the different adaptive operator selection mechanisms, we applied them to a complex real-world parameter fitting problem using a real-coded GA, and to three symbolic regression problems [10,11] using TAG3P.

The parameter fitting problem is based on a complex time series model built by a domain expert; parameter values for the model are unknown, and the aim of the GA is to find a best fit for them [8]. Little is known a priori about the fitness landscape, and the cost of running the model makes it difficult to explore. Root Mean Square Error (RMSE) was used as the fitness metric, as is customary in this domain. Problem details and evolutionary settings for this and the following experiments are provided in table 1. The 100 runs for each treatment used the same set of 100 seeds.

The GP experiments use Koza's well-known symbolic regression quintic, sextic and trigonometric problems [10,11]. Following Koza, we used Mean Absolute Error (MAE) as the objective, with a solution being declared a success if its absolute error is less than ϵ for all data points.

3.3 Genetic Algorithm Details

Search Space. The chromosome consists of 15 genes (one for each model parameter), with acceptable ranges for each being determined by the domain expert.

Initialisation. The genes were initialised uniformly randomly over their ranges.

Operators In the experiments, we used eight genetic operators:

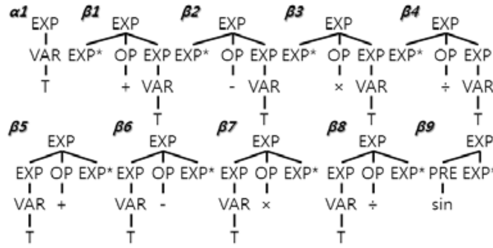


Fig. 1. TAG ElementaryTrees for Symbolic Regression Search Spaces

1. **One-point Crossover** chooses a random point in the chromosome, and exchanges the chromosome segments beyond that point.
2. **Two-point Crossover** chooses two random points in the chromosome, and exchanges the chromosome segments between them.
3. **Uniform Crossover** takes each value from one parent chromosome with probability p, and from the other with probability $1 - p$.
4. **Arithmetic Crossover** takes the arithmetic average of the parents' values.
5. **Random Crossover** randomly chooses a value between those of the parents.
6. **Reproduction Mutation** chooses two random points as in two-point crossover, and randomly initializes the gene values between them.
7. **Re-Initialisation** initialises each gene value with probability p.
8. **Range Mutation** randomly changes each gene over a range, which is specific to the corresponding parameter. If the new value lies outside the valid range, it is re-initialised.

3.4 TAG3P System Details

Search Space. TAG3P relies on a user-defined grammar specifying the search space. For these problems, we used the grammar defined by the elementary trees in figure 1. The terminal node T is substituted by $\{X\}$ for quintic and sextic, and by $\{X, 1\}$ for trigonometric. β_9 is not used in the quintic and sextic problems.

Initialisation. We sample tree sizes uniformly over the specified size range. Starting from an α tree, we randomly adjoin beta trees until the specified size is reached, then make any necessary substitutions for the terminal T.

Operators. In the experiments, we used seven TAG3P operators. Two of them are dual pairs, with opposite size biases, so that their overall effects are unbiased.

1. **Reproduction** copies the parent to the child population unchanged.
2. **One-point Crossover** chooses random points in the parent chromosomes, and exchanges the subtrees below them.
3. **Subtree Mutation** selects a random point in the chromosome, deletes the subtree below that point, and replaces it with a new subtree generated using the initialisation algorithm.

4. **Insertion and Deletion** are dual operators, applied with equal probablity, useful for fine-tuning the size. In insertion, a β tree is adjoined at a randomly-chosen node. In deletion, a β tree is removed.
5. **Duplication and Truncation** are also dual operators, applied with equal probability, but more useful for coarse adjustment. In duplication, a randomly selected subtree is copied and randomly adjoined at another location in the same individual.In truncation, a randomly selected subtree in an individual is removed.
6. **Replacement** randomly chooses two nodes in the parent which can adjoin in each other's location, and exchanges them..
7. **Relocation** disconnects a random subtree from the tree, then randomly re-adjoins it at another location. It thus has no effect on size.

3.5 Adaptive Mechanisms

We compared six adaptive mechanisms: probability matching (PM), power probability matching (PPM), adaptive pursuit (AP), adaptive probability matching (APM), static multi-armed bandit (S-MAB) and dynamic multi-armed bandit (D-MAB). Their parameters, determined through the literature and preliminary experiments, are shown in Table 2.

Reward Policy. Adaptive mechanisms rely on a reward, reflecting the effectiveness of operators in improving the population. We used a reward function based on the ratio of fitness values of children F_c and their corresponding parents F_p. Evolutionary algorithms focus resources on the elite, so our reward is calculated over the elite 30% of the children created by that operator (Eq. 6):

$$R = \frac{\sum F_p}{\sum F_c} (\text{summed over the fittest 30\% of the children}) \quad (6)$$

4 Results

The results fall naturally into three groups, which happen to share the same basic mechanisms: (PM, PPM), (AP, APM), (S-MAB, D-MAB). The first two pairs show similar behaviour on most problems, but S-MAB and D-MAB differ substantially. In all tables, the best performance is in bold; results which are significantly worse (1% confidence, Mann-Whitney rank sum test) are in italic.

Table 2. Operator Rate Parameters for Adaptive Mechanisms

	Parameter	Value		Parameter	Value
PM, AP,	Initial Rate P_{init}	$1/K$	S-MAB	δ	0.15
PPM & APM	Min. Rate P_{min}	$K/4$	&	λ	0.5
	α	0.8	D-MAB	Scale Factor C	0.5
AP & APM	Max. Rate P_{max}	$1-(K-1)\cdot P_{min}$	AP & APM	β	0.8

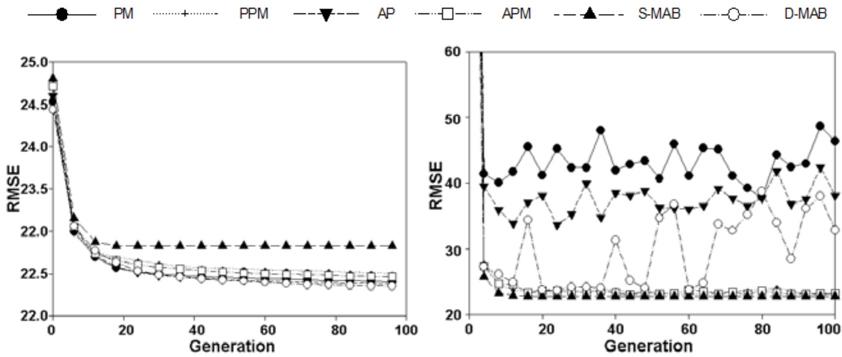


Fig. 2. RMSE Fitness on GA Parameter Fitting Problem. Left: Mean (over all runs) of the Best (in each population). Right: Median (over all runs) of the Mean (of each population).

Table 3. End of run best fitness values (mean ± standard deviation)

Method	Mean ± *SD*	Method	Mean ± *SD*
PM	*22.403 ± 0.064*	PPM	*22.375 ± 0.052*
AP	*22.499 ± 0.148*	APM	*22.464 ± 0.074*
S-MAB	*22.830 ± 0.222*	D-MAB	**22.347± 0.041**

4.1 The GA Parameter Fitting Problem

Figure 2 shows the mean value of the best fitness and the median value of the mean fitness for the GA problem.[1] Five of the six treatments show very similar performance, S-MAB performing worse. This poor performance probably results from S-MAB focusing all its effort on one-point crossover from generation 10 (see subsection 4.3), and that this operator may simply not be sufficient to escape from local optima.The median fitness is very close to the best fitness in this case, suggesting that the S-MAB populations have almost entirely converged.

Table 3 shows the mean of the end-of-run best fitness values. D-MAB is clearly the best performer, somewhat better than PPM and significantly better than the others, with S-MAB giving very much the worst performance.

4.2 TAG3P Symbolic Regression Problems

Figure 3 shows the mean best fitness by generation for the three symbolic regression problems. The mean error for the AP pair is larger than others on the Quintic and Sextic problems, while all values are similar on Trigonometric. In detail, the PM strategy gives the smallest error overall, especially on the trigonometric problem. The two MAB strategies give identical results – because unlike

[1] Because of extreme skew in fitness, the median is more informative than the mean.

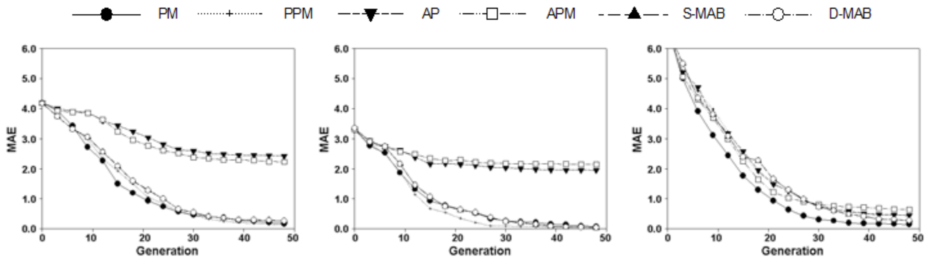


Fig. 3. Mean best fitness (MAE on sample points). Left: Quintic, centre: Sextic, right: Trigonometric.

Table 4. Proportion of runs where best individual is a hit

Method	*Quintic*	*Sextic*	*Trigonometric*
PM	72.0 %	94.0 %	76.0 %
PPM	73.0 %	**97.0** %	74.0 %
AP	**81.0** %	96.0 %	66.0 %
APM	78.0 %	96.0 %	**77.0** %
S-MAB	64.0 %	96.0 %	67.0 %
D-MAB	64.0 %	96.0 %	67.0 %

Table 5. Mean ± standard deviation of first hitting time

Method	Quintic	Sextic	Trigonometric
PM	25.63 ± 18.36	*19.27 ± 12.33*	34.46 ± 12.20
PPM	25.10 ± 17.74	**15.27** ± **9.89**	36.04 ± 11.34
AP	**24.06**± **15.74**	17.62 ± 11.65	*38.80 ± 12.37*
APM	24.29 ± 17.34	15.77 ± 11.14	**34.02** ± **11.72**
S-MAB	27.78 ± 18.79	*19.00 ± 11.60*	*38.12 ± 12.04*
D-MAB	27.78 ± 18.79	*19.00 ± 11.60*	*38.12 ± 12.04*

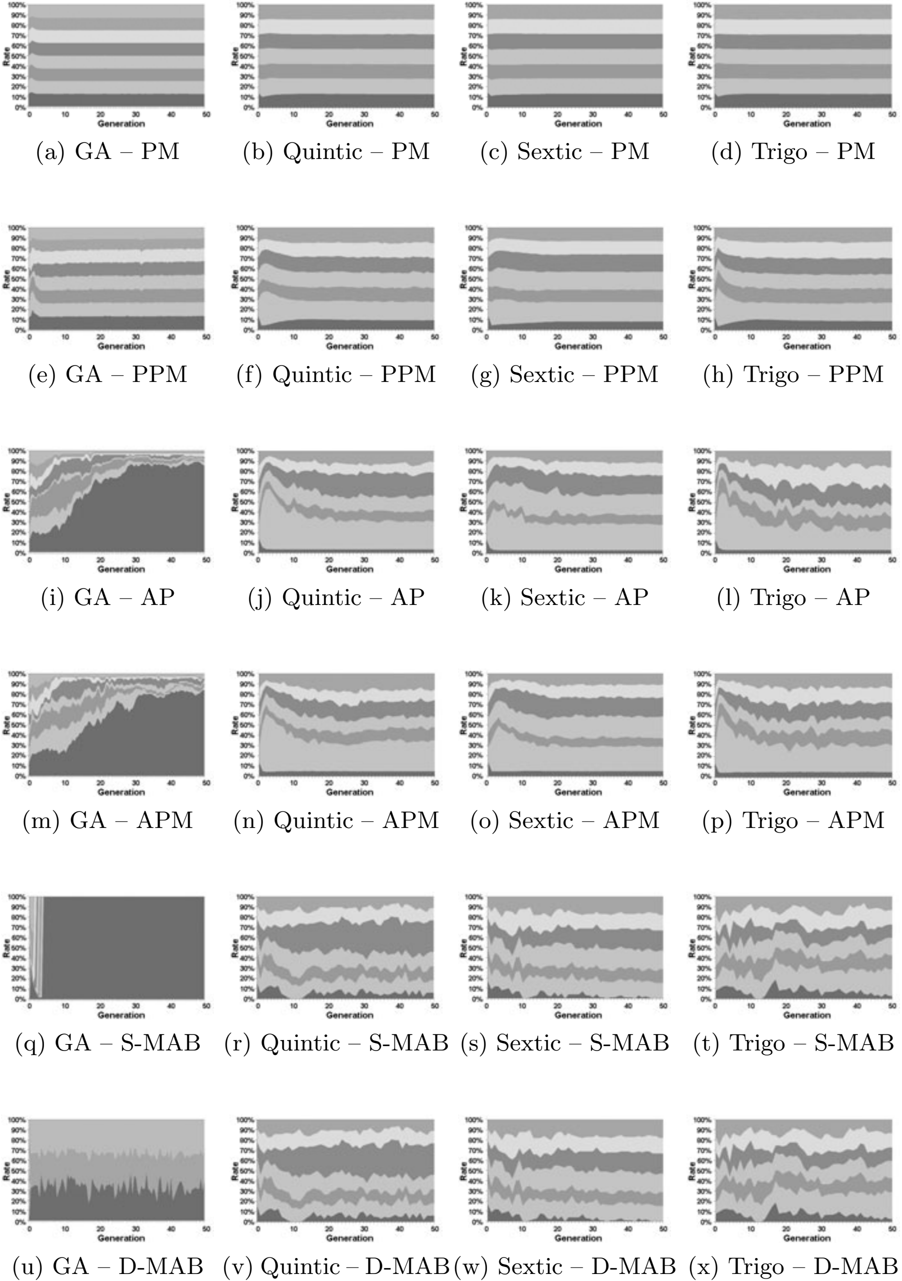
the situation in GA, the dynamic change condition was never triggered in the GP runs, so that S-MAB and D-MAB algorithms performed identically.

GP analysis emphasises hits – finding near-exact solutions. Table 4 shows this statistic. While AP and APM perform poorly in fitness, they have the best success ratios. PPM and APM show good performance in all three problems.

In addition to the success ratio, we are also interested in how quickly an algorithm finds a hit. Table 5 shows the mean and standard deviation of the first hitting time. AP, PPM and APM are respectively the best on the Quintic, Sextic and Trigonometric problems. Overall, first hitting time results closely shadow success rate, and APM is either best or near best.

4.3 Operator Application Rates

Figure 4 depicts the change in rates during evolution. With PM and PPM we don't see a great deal of variation. In PM, we see a small kick up in the rates of



(a) GA – PM (b) Quintic – PM (c) Sextic – PM (d) Trigo – PM

(e) GA – PPM (f) Quintic – PPM (g) Sextic – PPM (h) Trigo – PPM

(i) GA – AP (j) Quintic – AP (k) Sextic – AP (l) Trigo – AP

(m) GA – APM (n) Quintic – APM (o) Sextic – APM (p) Trigo – APM

(q) GA – S-MAB (r) Quintic – S-MAB (s) Sextic – S-MAB (t) Trigo – S-MAB

(u) GA – D-MAB (v) Quintic – D-MAB (w) Sextic – D-MAB (x) Trigo – D-MAB

Fig. 4. Change in Application Rates of Operators
(Bottom to Top of each subfigure: Operators in the Order from Subsections 3.3 and 3.4)
NB: GA (first column) uses a different operator set from GP (other three columns).

1-point crossover and reproduction mutation in the first few generations, but it rapidly settles back to near-uniform rates. Similarly in GP, crossover and subtree mutation increase at first, but settle back to relatively uniform rates except for a permanent drop in the rate of reproduction. All these effects are amplified, as we might expect, in PPM, but with the same overall outline.

AP and APM behave much more dynamically, but very similarly to each other. In GA, the clear trend is to increase the rate of one-point crossover at the expense of all other operators. In GP, we see the same behaviour as for PPM – transient increase in crossover and later suppression of reproduction – but substantially amplified. Later on, we see a gradual increase in rates of duplication and truncation.

S-MAB and D-MAB are even more dynamic, causing much more change in operator rates. In GA runs, both entirely eliminate many operators within a few generations – after a brief flirtation with re-initialisation and range mutation, S-MAB eliminates all operators except one-point crossover, while D-MAB retains re-initalization and range mutation. In GP, S-MAB and D-MAB showed very similar overall behaviour to APM, but with more chaotic behaviour – MAB strategies are more susceptible to amplifying random variations in operator success than the other strategies.

5 Discussion

Overall, the best performer in these experiments was APM, being fairly close to the best performance on all problems. But the differences in most cases are relatively small – any of AP, APM, PM or PPM gives reasonable performance with reasonable parameter settings.

MAB strategies were very different. Considerable effort was required to find good parameter settings for D-MAB for the GA problem (the original draft of this paper showed poor performance for both MABs strategies on all problems – until we happened to hit on exactly the right settings for D-MAB for this GA problem), and we were never able to find good parameter settings for S-MAB on the GA problem, or for either MAB strategies on the GP problems.

6 Conclusions

6.1 Summary

The PM and AP strategies extend reasonably well to a scenario with multiple diverse operators; AP generally works well, though when if fails it can fail badly (i.e. even though it had a good success rate on GP problems, the mean best fitness is poor). PPM and especially APM somewhat improve this performance, though APM shares with AP the high variation in performance between runs. However differences overall are not huge, and it remains an open question whether the additional complexity of these adaptive mechanisms, versus a simple uniform operator selection, is really justified for many-operator systems.

MAB strategies can perform very well when they are properly tuned to the problem and operators. However they are very sensitive to this tuning in the contexts we investigated here, so may be difficult to justify for these applications.

6.2 Assumptions and Limitations

The major limitation in this work was our inability to find suitable parameter settings for D-MAB in the GP experiments, or for S-MAB at all. It is possible that there are no good settings, but from our experience of the very high sensitivity of these algorithms to the parameter settings, we strongly suspect that there are good settings, we just didn't succeed in finding them. However it is clear that the good settings are problem specific, and cannot be transferred willy-nilly from one problem to another.

6.3 Further Work

We plan to further investigate parameter settings for MAB strategies in these problems, in the hope of finding more general methods for determining them. The insights resulting from this work are also helping us to design further strategies suited to evolutionary systems with multiple diverse operators.

Acknowledgments. Seoul National University Institute for Computer Technology provided research facilities for this study, which was supported by the Basic Science Research Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Project No. 2010-0012546/2011-0004338).

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2), 235–256 (2002)
2. DaCosta, L., Fialho, Á., Schoenauer, M., Sebag, M.: Adaptive operator selection with dynamic Multi-Armed bandits. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 913–920. ACM, New York (2008)
3. Fialho, Á., Da Costa, L., Schoenauer, M., Sebag, M.: Dynamic Multi-Armed Bandits and Extreme Value-Based Rewards for Adaptive Operator Selection in Evolutionary Algorithms. In: Stützle, T. (ed.) LION 3. LNCS, vol. 5851, pp. 176–190. Springer, Heidelberg (2009)
4. Fialho, Á., Schoenauer, M., Sebag, M.: Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 779–786. ACM, New York (2009)
5. Goldberg, D.: Probability matching, the magnitude of reinforcement, and classifier system bidding. Machine Learning 5(4), 407–425 (1990)
6. Hoai, N.: A Flexible Representation for Genetic Programming: Lessons from Natural Language Processing. Ph.D. thesis, University of New South Wales, Australian Defence Force Academy (2004)

7. Igel, C., Kreutz, M.: Operator adaptation in evolutionary computation and its application to structure optimization of neural networks. Neurocomputing 55, 347–361 (2003)
8. Kim, D., McKay, R.I., Haisoo, S., Yun-Geun, L., Xuan, N.X.: Ecological application of evolutionary computation: Improving water quality forecasts for the nakdong river, korea. In: World Congress on Computational Intelligence, pp. 2005–2012. IEEE Press (2010)
9. Kim, M.H., McKay, R.I(B.), Nguyen, X.H., Kim, K.: Operator Self-adaptation in Genetic Programming. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 215–226. Springer, Heidelberg (2011)
10. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
11. Koza, J.R.: Genetic Programming II Automatic Discovery of Reusable Programs. MIT Press (1994)
12. Lobo, F., Lima, C., Michalewicz, Z. (eds.): Parameter setting in evolutionary algorithms. SCI, vol. 54. Springer, Heidelberg (2007)
13. Page, E.: Continuous inspection schemes. Biometrika 41(1), 100–115 (1954)
14. Schwefel, H.: Numerical optimization of computer models. John Wiley & Sons, Inc., New York (1981)
15. Thathachar, M., Sastry, P.: A class of rapidly converging algorithms for learning automata. IEEE Transactions on Systems, Man and Cybernetics 15, 168–175 (1985)
16. Thierens, D.: Adaptive Strategies for Operator Allocation. In: Lobo, F.G., Lima, C.F., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms. SCI, vol. 54, pp. 77–90. Springer, Heidelberg (2007)
17. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, pp. 1539–1546. ACM, New York (2005)
18. Tuson, A., Ross, P.: Adapting operator settings in genetic algorithms. Evolutionary Computation 6(2), 161–184 (1998)

The Effect of Bloat on the Efficiency of Incremental Evolution of Simulated Snake-Like Robot

Ivan Tanev, Tüze Kuyucu, and Katsunori Shimohara

Information Systems Design
Doshisha University, Kyotanabe, Japan
{itanev,tkuyucu,kshimoha}@mail.doshisha.ac.jp

Abstract. We present the effect of bloat on the efficiency of incremental evolution of locomotion of simulated snake-like robot (Snakebot) situated in a challenging environment. In the proposed incremental genetic programming (IGP), the task of coevolving the locomotion gaits and sensing of the bot in a challenging environment is decomposed into two subtasks, implemented as two consecutive evolutionary stages. In the first stage we use genetic programming (GP) to evolve a pool of morphologically simple, sensorless Snakebots that move fast in a smooth, open terrain. Then, during the second stage, we use this pool to seed the initial population of Snakebots that are further subjected to coevolution of their locomotion control and sensing morphology in a challenging environment. The empirical results suggest that the bloat no immediate effect on the efficiency of the first stage of IGP. However, the bloated seed contributes to a much faster second stage of evolution. In average, the second stage with bloated seed reaches the best fitness values of the parsimony seeds about five times faster. We assume that this speedup is attributed to the neutral code that is used by IGP as an evolutionary playground to experiment with developing novel sensory abilities, without damaging the already evolved, fast locomotion of the bot.

Keywords: Incremetal genetic programming, Bloat, Neutrality.

1 Introduction

The insufficient efficiency of the genetic programming (GP), together with its non-determinism are among the most important drawbacks that still hinder the wide adoption of the evolutionary paradigm for solving challenging real-world problems. The overall efficiency of GP depends on the cumulative effect of two major, relatively independent factors: (i) the computational effort, i.e., the number of genetic programs that should be evaluated in order to achieve a given probability of success, and (ii) the computational performance, i.e., the average runtime, required to evaluate a single genetic program. Therefore, most of the efforts of researchers and practitioners in evolutionary computing (EC) community are aligned along the two orthogonal directions–improving the computational effort and computational performance of GP.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 242–253, 2012.
© Springer-Verlag Berlin Heidelberg 2012

The computational effort of GP could be improved in several ways, such as, incorporating a domain-specific knowledge into the key attributes of GP (e.g., genetic representation, genetic operations, etc.), imposing problem-specific syntax constrains (i.e., grammar) on the evolved genetic programs, employing probability-distribution models, etc. These approaches are usually intended to steer the simulated evolution towards the most promising areas in the explored (presumably rugged) fitness landscapes.

Another approach of improving computational effort of GP stems from the assumption that, the main genetic operations (crossover and mutation), due to their randomness, are often damaging the already existing building blocks of the solution. Thus, the destructive effects of these operations could be limited if they are occasionally allowed to operate on the genetic code that is irrelevant to the quality of the corresponding genetic program. Moreover, such an irrelevant, or neutral code (i.e., the code, which has no affect on the fitness), might provide the simulated evolution with a "playground" where it can experiment with developing either novel-, or better-than-existing genotypic traits without the risk of damaging the already evolved ones.

In biology, the constructive role of neutrality has been well recognized and noted to be a conducive mechanism in the evolution of many successful traits in biological organisms [1,2]. The presence of neutral genes in Nature is often associated with smoother fitness landscapes, more robust genotypes and a support mechanism for discovering new phenotypes [1,3,2]. Similarly, the neutrality has also been a topic of interest and discussion in EC. In a recent article, Galvan-lopez et al [4], provide an overview of the role of neutrality in EC. The authors conclude their work with the open issues in this area and with their opinion on the studies that would be beneficial to a better understanding of the role of neutrality in EC. Due to the complex and, to some extent, unpredictable nature of the latter, it is often difficult to achieve an in-depth theoretical analysis of the effects of mechanisms and parameters on the performance of these algorithms.

The work of Ebner [5] suggests that the neutrality induced by the "junk code" provides better performance for the evolution of genetic programs. It was also reported that the neutrality contributes to a better efficiency of evolution in Cartesian Genetic Programming (CGP) [6,7]. This conclusion remains to be a controversial one, however, as in a follow-up study Collins reported that the latter work is being flawed and that the effects of neutrality could in fact be degrading the overall performance of evolution [8]. This ongoing discussion on the possible beneficial effect of neutrality on the overall efficiency of GP is also stated by Galvan-lopez et al [4], who note that "one needs to find ways of predicting when the addition of neutrality can be beneficial in practical situations." Although neutrality has been shown to be beneficial in complex, rugged landscapes with multiple optima [9,10,11], it also has adverse effects on the evolution of simple problems with a small number of optima and a smooth landscape.

Despite the beneficial effects of neutrality on the computational effort of GP, its implications on the overall performance of GP are still highly controversial. One of the most important factors for this controversy is that neutrality often

causes a degradation of computational performance of GP. Indeed, the neutral code in GP is often associated with the resulting bloat [12], or, with a sharp increase of the size (and complexity) of genetic programs in due course of the simulated evolution. Often, the increased size of the evolved genetic programs does not correlate well with the convergence of the respective fitness values. Moreover, due to the cache memory effects, the runtime overhead of interpreting the genetic programs (represented as highly fragmented parse trees) grows faster than linear with the increase of their size. Therefore, applying a parsimony pressure in order to limit the growth of the size of genetic programs is usually seen as a natural way to alleviate the problem of bloat-induced degradation of computational performance of GP [13,14].

The objective of our work is to investigate the effects of bloat on the efficiency of GP employed for simulated incremental evolution of locomotion of sensing snake-like robot (Snakebot) in a challenging environment with obstacles. The successful bots should feature the evolved (emergent) now-how about how to clear a narrow corridor by (i) moving fast, (ii) following the walls of the corridor, (iii) overcoming a number of randomly scattered small boxes, and (iv) circumnavigating large obstacles. In the proposed incremental GP (IGP), the task of coevolving the locomotion and the sensing of Snakebot in a challenging environment is decomposed into two subtasks, implemented as two consecutive evolutionary stages. First we employ GP to evolve a pool of simple, sensorless bots that are able to move fast in a smooth, open terrain. Then, during the second stage, we use this pool to seed the initial population of the bots that are further subjected to coevolution of their locomotion control and sensing in the challenging environment. We are especially interested on how the degree of bloat (and, consequently, the associated neutrality), introduced during the fist stage of the incremental evolution, effects the overall performance of IGP.

Our choice of the application of GP is motivated by two arguments that, as we believe, are in favour of the neutrality. First, the considered problem is rather challenging, as it features a huge and highly rugged fitness landscape [15]. Shipman demonstrated that neutrality helps the discovery of multiple phenotypes, but reduces the evolutionary performance for achieving faster solutions in simpler problems [16]. Within this context, we would like to investigate if, during the second stage of IGP, the neutrality would decrease the computational effort of evolving novel traits (e.g., the sensory abilities of the bot) in addition to the already evolved locomotion of the bots. And second, as in the most of the tasks in evolutionary robotics, it is the realistic simulation of the physics of moving complex robotic artifact, rather than the parsing of the genetic programs that consumes the most of the runtime of GP. Therefore, we anticipate no major bloat-induced degradation of the computational performance of GP.

The remaining of this document is organized as follows. Section 2 introduces the morphology and the moving abilities of the Snakebot. In Section 3 we discuss the key attributes of the proposed evolutionary framework. Section 4 presents the empirical result on the effect of bloat on the efficiency of incremental evolution of the bot in a challenging environment. Section 5 draws a conclusion.

2 Sidewinding and Sensing Snake-Like Modular Robot

Snake-like robots feature potential robustness characteristics beyond the capabilities of most wheeled and legged vehicles, such as: the ability to traverse challenging terrain and insignificant performance degradation when partial damage is inflicted. Some useful features of snake-like robots include smaller size of the cross-sectional areas, stability, ability to operate in difficult terrain, good traction, and complete sealing of the internal mechanisms. Moreover, due to the modularity of their design, the snake-like robots feature high redundancy and fault tolerance [17]. Robots with such properties can be valuable for applications that involve exploration, reconnaissance, medicine and inspection. Designing a controller that can achieve optimal locomotion of a modular Snakebot is a challenging task due to the large number of degrees of freedom in the movement of segments of a Snakebot. The locomotion gait of such bots is often seen as an emergent property; observed at a higher level of consideration of complex, nonlinear, hierarchically organized systems, comprising many relatively simply-defined entities (morphological segments). In such complex systems the higher-level properties of the system and the lower-level properties of comprising entities cannot be directly induced from each other [18]. Therefore even if an effective incorporation of sensing information in fast and robust locomotion gaits might emerge from intuitively defined sensing morphology and simple motion patterns of morphological segments, neither the degree of optimality of the developed code nor the way of how to incrementally improve this code is evident to the human designer [19].

The previous research demonstrates that the control for a fast moving modular robotic organism could be automatically developed through various nature-inspired paradigms, based on models of learning and evolution. The work, presented in [17] demonstrates the use of GP for evolution of sensorless sidewinding Snakebots in various environmental conditions. Furthermore, the coevolution of active sensing and the control of the locomotion gaits was demonstrated to be achievable, albeit difficult [15]. The control of a modular snakebot with sensors for navigation through a maze with obstacles was shown to be a challenging task for canonical GP even when ADFs were used. The use of IGP was shown to be a better approach, where initially the locomotion of the snakebot in an obstacle free environment was achieved before evolving these snakebots for a second time to utilize sensors. Furthermore, the use of a Genetic Transposition inspired incremental GP, which utilizes the addition of neutral code into the genotype of the seeding individuals for the second stage of the incremental GP runs had higher success rates as well as more robust solutions [15].

In this paper we investigate the coevolution of the active sensing and locomotion control of sidewinding Snakebot in the same environment presented in [15], which features a narrow corridor with several large obstacles and many randomly placed small obstacles constituting a rugged terrain. The sensors on the Snakebot used in this paper follow the same model as proposed in [20]: each segment of the Snakebot is provided with a fixed, immobile proximity sensors (e.g., laser range finders, LRF) with evolvable initial orientation, range and timing of firing.

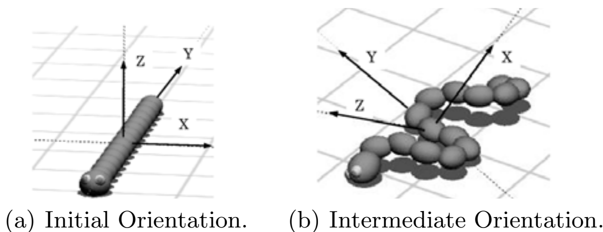
The most efficient locomotion gaits of Snakebot are not necessarily associated with the forward, rectilinear motions (and sidewinding might emerge as a fast and robust locomotion). Therefore, the eventual fusion of the readings of many sensors mounted in all the segments of the bot would provide Snakebot with the capability to perceive the features of surrounding environment along its whole body. In addition to the widening of the area of the perceived surroundings, multiple sensors offer the potential advantages of robustness to damage of some of them, dependability of the sensory information, and an ability to perceive the spatial features of the surrounding environment due to the motion parallax.

The evolution of both the morphology and the incorporation of the signals from many sensors face the challenge of dealing with the uncertain sensor readings as they move synchronously with the coupled segments of the snake. Figure 1 illustrates how the initial orientation of the axes of the internal coordination systems of the segments of a bot dramatically differs from a sample instant orientation of these axes in a moving bot. A sensor fixed to the segment of a moving Snakebot would constantly change its spatial orientation, and consequently it might alternatively perceive no signal, a signal from the ground surface or from another segment of the snake (in both cases the sensory reading should be ignored), or eventually from an obstacle. Moreover, in the targeted environment the obstacle could be either a wall (to be followed), a large box (to be circumnavigated), or a small box (to be overcome).

The large search space of the evolution of the considered Snakebot results in an intractable computational effort, and as it was demonstrated in [15], canonical GP with Automatically Defined Functions (ADF) is unable to effectively explore the emerging search space in a reasonable time frame.

3 Evolutionary Framework and the Simulation Environment

For the experiments presented in this work we employ open dynamics engine (ODE) as a simulation platform for the Snakebot. ODE is a free, industrial quality software library for simulating articulated rigid body dynamics [21]. It is fast, flexible and robust, and it has built-in collision detection. Therefore, ODE



(a) Initial Orientation. (b) Intermediate Orientation.

Fig. 1. Orientation of the axes of the internal coordination systems of the central segment at two different Snakebot positions

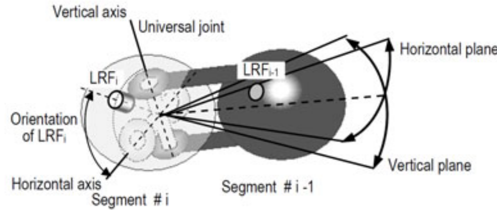


Fig. 2. Morphological segments of Snakebot are linked via universal joint. Horizontal and vertical actuators attached to the joint perform rotation of the segment #i-1 in vertical and horizontal planes respectively. A single LRF is attached to each of the segments in the plane of the axes of the universal joint.

is suitable for a realistic simulation of the physics of an entire Snakebot when applying actuating forces to its segments. The main ODE related parameters of the simulated Snakebot are same as elaborated in [17].

Snakebot is simulated in ODE as a set of 15 identical spherical morphological segments, linked together via universal joints (Figure 2). All joints feature identical angle limits and each joint has two attached actuators ("muscles"). A single LRF sensor, with a limited range is rigidly attached to each of the segments.

The functionality of the LRF can be defined by the values of the following set of parameters: (i) orientation, measured as an angle between the longitudinal axis of the sensor and the horizontal plane of the bot in its initial, standstill position of Snakebot (as depicted in Figure 1(a)), (ii) range of the sensor (in cm), and (iii) the timing of their activation, expressed as a threshold value of the turning angle of the horizontal actuator. The reading of LRF is inversely proportional to the distance between the sensor and an object (if any within the sensor's range), measured along the longitudinal axis of the LRF. In the initial standstill position of Snakebot the rotation axes of the actuators are oriented vertically (for the vertical actuator) and horizontally (horizontal actuator) and perform rotation of the joint in the horizontal and vertical planes respectively.

Considering the representation of Snakebot, the task of designing the fastest locomotion can be rephrased as developing temporal patterns of desired turning angles of horizontal and vertical actuators of each segment that result in fastest overall locomotion of Snakebot. The proposed representation of Snakebot as a homogeneous system comprising identical morphological segments is intended to significantly reduce the size of the search space of the GP.

For the evolution of the Snakebot, the genotype is represented as a triple consisting of a linear chromosome containing the encoded values of the three relevant parameters of LRF, and two parse trees corresponding to the algebraic expressions of the temporal patterns of the desired turning angles of both the horizontal and vertical actuators, respectively (Figure 3).

The Snakebot is genotypically homogeneous in that the same triple is applied for the setup of the LRF and for the control of actuators of all morphological segments. The encoding of the parameters of LRF is as elaborated in Figure 3.

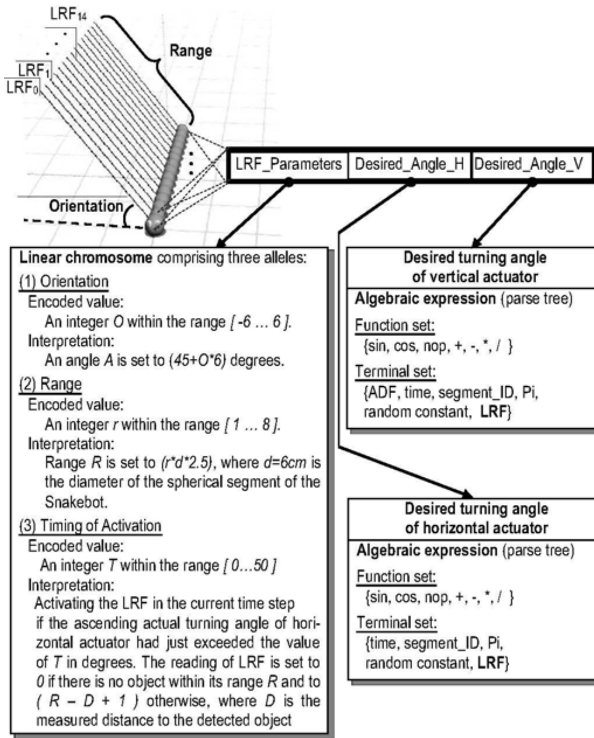


Fig. 3. Genotype of the Snakebot; represented as a triple containing the values of the parameters of LRF and two algebraic expressions of the temporal patterns of the desired turning angles of horizontal and vertical actuators, respectively. The genotype of Snakebot is homogeneous: therefore all segments feature the same triple.

The same figure also illustrates the function set and the terminal set of the GP, employed to evolve the control sequences of both actuators. Since the locomotion gaits by definition are periodical, the periodic functions sine and cosine are included in the function set of GP in addition to the basic algebraic functions. Terminal symbols include the variables time, segment_ID, an ADF, the reading of the sensor (LRF), and two constants: pi, and a random constant within the range [0, 2]. The incorporation of the terminal symbol segment_ID (a unique index of morphological segments of Snakebot) provides GP with an effective way to specialize (by phase, amplitude, frequency etc.) the genetically identical motion patterns of actuators of each of the morphological segments of the Snakebot.

The main GP (hence the EA) parameters are summarized in Table 1. We use a DOM/XML-based implementation of GP [22], with binary tournament selection and a single point crossover. The crossover point is randomly selected between the three components of the genotype (as shown in Figure 3), unless stated otherwise. The mutation randomly alters either a value of an allele in

Table 1. Main parameters of GP

Category	Value
Genotype	LRF parameters (linear chromosome) Horizontal actuator control (parse tree) Vertical actuator control(parse tree)
Population Size	200 individuals
Selection	Binary Selection ratio: 0.1 Reproduction ratio: 0.9
Elitism	4 individuals
Mutation Rate	0.01
Trial Interval	16s (400 time steps of 40ms per step)
Termination Criteria	(Fitness=120) or (Tot. No. of generations=80)

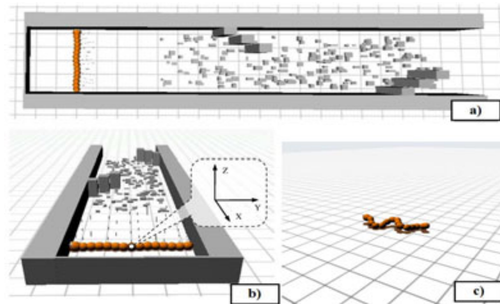


Fig. 4. The experimental scenes used

the linear chromosome representing the parameters of LRF, or a sub-tree in one of the two parse tress that correspond to the temporal patterns of the control sequences of actuators.

The fitness function is based on the average velocity of Snakebot, which is estimated from the distance travelled during the trial. The real values of the raw fitness, which are usually within the range (0, 2) are multiplied by a normalizing coefficient in order to deal with integer fitness values within the range (0, 200). A normalized fitness of 100 is equivalent to a velocity that displace the Snakebot a distance equal to twice its length.

4 Experiments

The experiments are carried out in the environments shown in Figure 4. Two stages are used for the incremental evolution of the sensing sidewinder Snakebots. In the first stage, the sidewinding Snakebots with no sensors are evolved in an obstacle free environment (Figure 4c) using random seeds. The target function for these experiments is to achieve a fitness of 120 or higher in the 16 second

evaluation period. Three experimental cases are created for the aforementioned situation, which differ only in the implementation of the fitness measure used to control bloat: *(i)Penalize bloat:* the fitness value is decreased by $\frac{1}{10}$ times the tree length, *(ii)Award bloat:* the fitness value is increased by $\frac{1}{10}$ times the tree length, *(iii)No bloat:* The fitness value is not altered with respect to the tree length. Each of these cases are executed for 40 runs and their results are shown in Figure 5.

For the second stage of IGP, there are three experimental cases as well, however for these cases identical evolutionary conditions are used i.e. all experimental cases feature the same parsimony pressure conditions, which is a penalty to the fitness of an individual by $\frac{1}{10}$ times the tree length in order to control bloat. We choose to use bloat control for all these runs, since it is the standard in our previous works. The difference in each experimental case arises in the seeds used. This way, the analysis of the results is solely based on the difference in the genetic code of the best Snakebots obtained from the different experimental cases in the first stage. In second stage, 6 of the best Snakebots from each of the experimental cases from the first stage are used to seed the populations of 3 different experimental cases and the rest of the population (194 individuals) is formed of randomly generated individuals.

4.1 Stage 1: Evolution of Fast Moving Snakebots from Random Population

The results in Figure 5 and Table 2 show that parsimony control during evolution has no effect on the performance. All three experiments demonstrate identical performance with no difference in average fitness achieved over the course of evolution and no difference in the number of successful runs. We also visually inspected the successful Snakebots from each case and could confirm that each had similar behaviour.

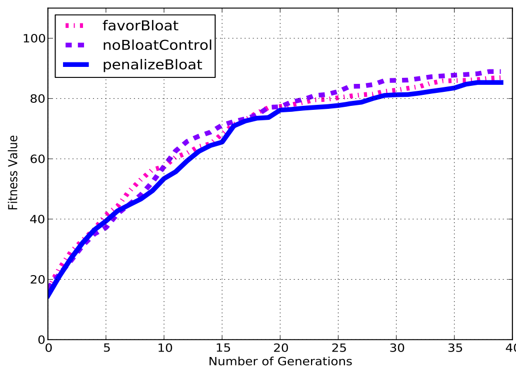


Fig. 5. The average fitness convergence plots of the results from stage 1

Table 2. Results from stage 1. The data for each experimental case is over 40 runs.

	No Bloat Control	**Reward Bloat**	**Penalize Bloat**
Median Fitness	101	100	100.5
$\#ofRuns >= 120$	4	3	4
$\#ofRuns >= 95$	24	23	21
Avg. Tree Size	105.3	258.4	82.4

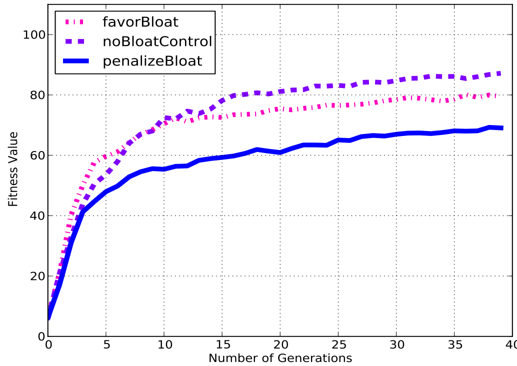


Fig. 6. The average fitness convergence plots of the results from stage 2. For the first 3 generations, the average fitness of the evolved chromosomes is same for all 3 cases. After a few generations, the runs with bloated seeds reach a higher average fitness.

We can conclude that the use of a simple parsimony pressure, as described earlier, has no implications on the overall performance of the first stage of IGP - the evolution of locomotion control for sidewinding Snakebots. In fact, the use of parsimony pressure did exactly what it is supposed to: the only significant difference among the three experimental cases was in the average tree sizes, and the use of parsimony pressure provided solutions with smallest tree sizes.

4.2 Stage 2: Seeded Evolution of Sensing Fast Moving Snakebots

The average fitness convergence of these seeded experiments is shown in Figure 6. Unlike the runs from first stage of IGP, there is a significant difference in the performance of the evolutionary runs using different seeds. The best evolutionary performance is obtained when seeds that were previously evolved with no bloat control are used, and the worst performance is observed when seeds with parsimony pressure are used. Table 3 provides more statistical information on the experimental runs from stage 2, where a clearer distinction between the runs using no bloat control and the other two can be seen from the number of successful/high scoring Snakebots. In average, the bloated seed reaches the best fitness values of the parsimony seeds about five times faster.

Table 3. Results from the seeded runs. The results are out of 40 runs for each case.

	No Bloat Control	Reward Bloat	Penalize Bloat
Median Fitness	87.3	82.5	68
$\#ofRuns >= 120$	8	0	3
$\#ofRuns >= 95$	12	6	5
Avg. Tree Size	197	177	122

5 Conclusions

In this work we studied the effect of bloat, and the associated genetic neutrality, on the efficiency of incremental evolution of simulated sensing snake-like robot in a challenging environment. As the experimental results suggest, the use of a simple parsimony pressure has no immediate effect on the efficiency of evolution of the first stage of IGP - evolution of fast moving sensorless bots in a smooth terrain. However, when, during the second stage of IGP, evolved genetic programs are reused for further development under different conditions than the first stage, the neutrality caused by the bloated genetic programs are beneficial for the more efficient evolution of the sensing abilities of the bot. We assume that this is due to the presence of a neutral code that can be used by IGP as an evolutionary playground where the novel sensory abilities could be developed without the risks of damaging the already evolved locomotion. The presented findings could be applied for the domains where the implementation of the side effects, rather than the parsing of genetic representation, is the most time-consuming aspect of fitness evaluation.

Acknowledgments. The presented research was supported (in part) by the Japan Society for the Promotion of Science (JSPS).

References

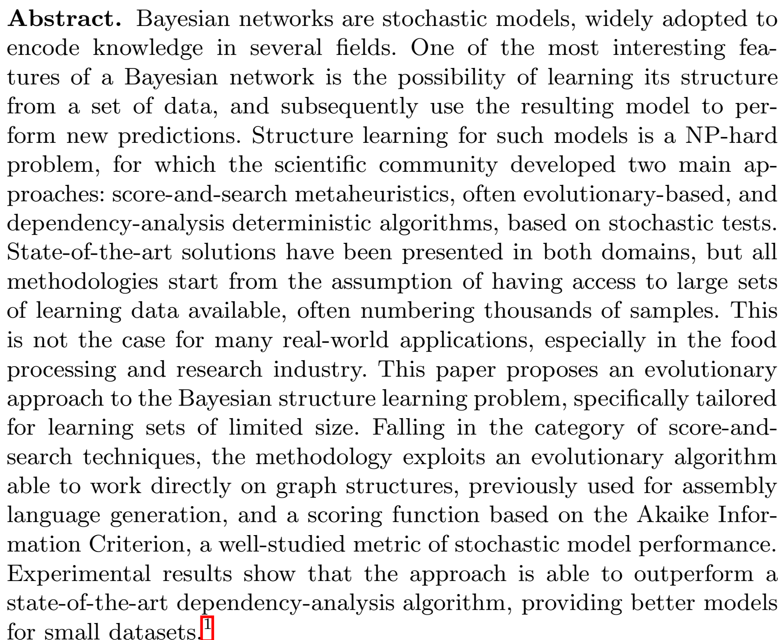
1. Huynen, M., Stadler, P., Fontana, W.: Smoothness within ruggedness: the role of neutrality in adaptation. Proceedings of the National Academy of Sciences of the United States of America 93, 397–401 (1996)
2. Wilke, C., Wang, J., Ofria, C., Lenski, R., Adami, C.: Evolution of digital organisms at high mutation rates leads to survival of the flattest. Nature 412, 331–333 (2001)
3. Wagner, A.: Robustness, evolvability, and neutrality. FEBS Letters 579(8), 1772–1778 (2005)
4. Galván-López, E., Poli, R., Kattan, A., O'Neill, M., Brabazon, A.: Neutrality in evolutionary algorithms... what do we know? Evolving Systems 2, 145–163 (2011)
5. Ebner, M.: On the search space of genetic programming and its relation to nature's search space. In: Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999, pp. 1357–1361 (1999)
6. Vassilev, V.K., Miller, J.F.: The Advantages of Landscape Neutrality in Digital Circuit Evolution. In: Miller, J.F., Thompson, A., Thompson, P., Fogarty, T.C. (eds.) ICES 2000. LNCS, vol. 1801, pp. 252–263. Springer, Heidelberg (2000)

7. Yu, T., Miller, J.F.: The role of neutral and adaptive mutation in an evolutionary search on the onemax problem. In: GECCO Late Breaking Papers 2002, pp. 512–519 (2002)
8. Collins, M.: Finding needles in haystacks is harder with neutrality. In: GECCO 2005: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1613–1618 (2005)
9. Beaudoin, W., Verel, S., Collard, P., Escazu, C.: Deceptiveness and neutrality: The nd family of fitness landscapes. In: GECCO 2006: Proceedings of the 2006 Conference on Genetic and Evolutionary Computation, pp. 507–514 (2006)
10. Doerr, B., Gnewuch, M., Hebbinghaus, N., Neumann, F.: A rigorous view on neutrality. In: IEEE Congress on Evolutionary Computation, CEC 2007, pp. 2591–2597 (September 2007)
11. Lobo, J., Miller, J.H., Fontana, W.: Neutrality in technological landscapes. In: Santa Fe Working Paper (2004)
12. Brameier, M., Banzhaf, W.: Neutral Variations Cause Bloat in Linear GP. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 286–296. Springer, Heidelberg (2003)
13. Gelly, S., Teytaud, O., Bredeche, N., Schoenauer, M.: Universal Consistency and Bloat in GP. Revue d'Intelligence Artificielle 20, 805–827 (2006)
14. Poli, R., McPhee, N.F.: Covariant parsimony pressure for genetic programming. Technical Report CES-480, Department of Computing and Electronic Systems, University of Essex, UK (2008)
15. Kuyucu, T., Tanev, I., Shimohara, K.: Incremental genetic programming via genetic transpositions for efficient coevolution of locomotion and sensing of simulated snake-like robot. In: European Conference on Artificial Life, pp. 439–446 (2011)
16. Shipman, R.: Genetic redundancy: Desirable or problematic for evolutionary adaptation. In: 4th International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA 1999), pp. 1–11 (1999)
17. Tanev, I., Ray, T., Buller, A.: Automated evolutionary design, robustness and adaptation of sidewinding locomotion of simulated snake-like robot. IEEE Transactions on Robotics 21, 632–645 (2005)
18. Morowitz, H.J.: The Emergence of Everything: How the World Became Complex. Oxford University Press (2002)
19. Koza, J., Keane, M., Yu, J., Bennett, F., Mydlowec, W.: Automatic creation of human-competitive programs and controllers by means of genetic programming. Genetic Programming and Evolvable Machines 1, 121–164 (2000)
20. Tanev, I., Shimohara, K.: Co-evolution of active sensing and locomotion gaits of simulated snake-like robot. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO 2008, pp. 257–264. ACM, New York (2008)
21. Smith, R.: Open Dynamics Engine (2004)
22. Tanev, I.T.: Dom/xml-based portable genetic representation of the morphology, behavior and communication abilities of evolvable agents. Artificial Life and Robotics 8, 52–56 (2004)

Bayesian Network Structure Learning from Limited Datasets through Graph Evolution

Alberto Paolo Tonda[1], Evelyne Lutton[2],
Romain Reuillon[1], Giovanni Squillero[3], and Pierre-Henri Wuillemin[4]

[1] Institut des Systèmes Complexes, 57-59 rue Lhomond, 75005, Paris, France
[2] INRIA Saclay-Ile-de-France, AVIZ Team
LRI - Bâtiment 650, Université Paris-Sud, 91405, Orsay Cedex, France
[3] Politecnico di Torino, DAUIN, Corso Duca degli Abruzzi 124, 10129, Torino, Italy
[4] LIP6 - Département DÉSIR, 4, place Jussieu, 75005, Paris
{alberto.tonda,romain.reuillon}@iscpif.fr, evelyne.lutton@inria.fr,
giovanni.squillero@polito.it, pierrehenri.wuillemin@lip6.fr



Abstract. Bayesian networks are stochastic models, widely adopted to encode knowledge in several fields. One of the most interesting features of a Bayesian network is the possibility of learning its structure from a set of data, and subsequently use the resulting model to perform new predictions. Structure learning for such models is a NP-hard problem, for which the scientific community developed two main approaches: score-and-search metaheuristics, often evolutionary-based, and dependency-analysis deterministic algorithms, based on stochastic tests. State-of-the-art solutions have been presented in both domains, but all methodologies start from the assumption of having access to large sets of learning data available, often numbering thousands of samples. This is not the case for many real-world applications, especially in the food processing and research industry. This paper proposes an evolutionary approach to the Bayesian structure learning problem, specifically tailored for learning sets of limited size. Falling in the category of score-and-search techniques, the methodology exploits an evolutionary algorithm able to work directly on graph structures, previously used for assembly language generation, and a scoring function based on the Akaike Information Criterion, a well-studied metric of stochastic model performance. Experimental results show that the approach is able to outperform a state-of-the-art dependency-analysis algorithm, providing better models for small datasets.[1]

Keywords: Evolutionary computation, Bayesian network structure learning, Bayesian networks, Genetic Programming, Graph representation.

[1] Acknowledgments for the funding received from the European Community's Seventh Framework Programme (FP7/2009-2013) under grant agreement DREAM n. 222654-2.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 254–265, 2012.
© Springer-Verlag Berlin Heidelberg 2012

1 Introduction

Bayesian networks are stochastic models widely used to encode knowledge in several different fields: computational biology and bioinformatics (gene regulatory networks, protein structure, gene expression analysis), medicine, document classification, information retrieval, image processing, data fusion, decision support systems, engineering, gaming and law.

A particularly interesting feature of a Bayesian network is the possibility of learning its structure from a set of data and subsequently use the obtained model to predict new results. However, the number of possible structures is superexponential in the number of variables of the model [1] and the problem of Bayesian network learning is proved to be NP-hard [2]. The machine learning community answered to this challenge with a research line dating back almost 30 years, giving birth to a class of effective deterministic algorithms that systematically determine the skeleton of the underlying graph and proceed to orient all arcs whose directionality is dictated by conditional independencies observed.

This interesting problem raised also the attention of the evolutionary computation community: several attempts at Bayesian network structure learning have been presented in recent years, ranging from cooperative coevolution [3] [4] [5] to evolutionary programming [6], to hybrid solution combining evolutionary approaches with heuristic search [7]. Both deterministic and evolutionary techniques share the assumption of the availability of dataset numbering thousands or hundreds of thousands of samples: for several real-world problems, however, this may not be the case [8]. For example, in the field of food processing and research, extremely time-consuming processes are required to get a small amount of sparse data.

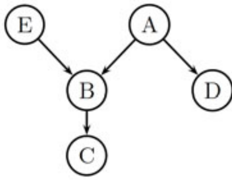
Taking inspiration from such a category of problems, this paper presents an evolutionary-based approach to Bayesian network structure learning, working with datasets of extremely reduced size. The proposed technique exploits a state-of-the-art evolutionary algorithm able to directly manipulate graph-like structures, previously used for assembly language generation, and makes use of a fitness function based on the Akaike information criterion, a metric taking into account both the accuracy and the complexity of a candidate model.

The rest of the paper is organized as follows. Section 2 briefly introduces the necessary background concepts on Bayesian networks. The proposed methodology is described in detail in section 3. Section 4 presents the case study chosen for the experimental evaluation, an established benchmark widely used in the Bayesian network learning field. Experimental results showing a comparison with a state-of-the-art dependency analysis algorithm are reported in section 5, while section 6 draws the conclusions and gives an outline for future works.

2 Background

2.1 Bayesian Networks

A Bayesian Network (BN) is a "graph-based model of a joint multivariate probability distribution that captures properties of conditional independence between



Node	Parents	Probabilities
A		$P(A{=}a_1) = 0.99$ $P(A{=}a_2) = 0.01$
B	A,E	$P(B{=}b_1 \mid A{=}a_1,E{=}e_1) = 0.5$ $P(B{=}b_2 \mid A{=}a_1,E{=}e_1) = 0.5$ $P(B{=}b_1 \mid A{=}a_1,E{=}e_2) = 0.1$ $P(B{=}b_2 \mid A{=}a_1,E{=}e_2) = 0.9$ $P(B{=}b_1 \mid A{=}a_2,E{=}e_1) = 0.4$ $P(B{=}b_2 \mid A{=}a_2,E{=}e_1) = 0.6$ $P(B{=}b_1 \mid A{=}a_2,E{=}e_2) = 0.2$ $P(B{=}b_2 \mid A{=}a_2,E{=}e_2) = 0.8$
C	B	$P(C{=}c_1 \mid B{=}b_1) = 0.3$ $P(C{=}c_2 \mid B{=}b_1) = 0.7$ $P(C{=}c_1 \mid B{=}b_2) = 0.5$ $P(C{=}c_2 \mid B{=}b_2) = 0.5$
D	A	$P(D{=}d_1 \mid A{=}a_1) = 0.8$ $P(D{=}d_2 \mid A{=}a_1) = 0.2$ $P(D{=}d_1 \mid A{=}a_2) = 0.7$ $P(D{=}d_2 \mid A{=}a_2) = 0.3$
E		$P(A{=}e_1) = 0.75$ $P(A{=}e_2) = 0.25$

Fig. 1. On the left, a directed acyclic graph. On the right, the parameters it is associated with. Together they form a Bayesian network BN whose joint probability distribution is $P(BN) = P(A)P(B|A,E)P(C|B)P(D|A)P(E)$.

variables" [9]. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. The network could thus be used to compute the probabilities of the presence of various diseases, given the symptoms.

Formally, a Bayesian network is a directed acyclic graph (DAG) whose nodes represent variables, and whose edges encode conditional dependencies between the variables. This graph is called the *structure* of the network and the nodes containing probabilistic information are called the *parameters* of the network. Figure 1 reports an example of a Bayesian network.

The set of parent nodes of a node X_i is denoted by $pa(X_i)$. In a Bayesian network, the joint probability distribution of the node values can be written as the product of the local probability distribution of each node and its parents:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|pa(X_i))$$

2.2 Akaike Information Criterion

The Akaike information criterion (AIC) is a measure of the relative goodness of fit of a statistical model [10]. It is grounded in the concept of information entropy, in effect offering a relative measure of the information lost when a given model is used to describe reality. It can be said to describe the trade-off between bias and variance in model construction, or loosely speaking, between accuracy and dimension of the model. Given a data set, several candidate models may be ranked according to their AIC values: thus, AIC can be exploited as a metric for model selection.

When dealing with Bayesian networks, AIC is expressed as a composition of the loglikelihood, a measure of how well the candidate model fits the given dataset, and a penalty tied to the dimension of the model itself. The dimensional penalty is included because, on the one hand, the loglikelihood of a Bayesian network usually grows monotonically with the number of arcs, but on the other

hand, an excessively complex network cannot be validated or even interpreted by a human expert. The loglikelihood of a model M given a dataset T is computed as

$$LL(M|T) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} log_2 \frac{N_{ijk}}{N_{ij}}$$

where n is the number of variables, r_i is the number of different values that the stochastic variable X_i can assume, q_i is the total number of possible configurations of its parent set $pa(X_i)$, N_{ijk} is the number of instances in the dataset T where the variable X_i takes its k-th value x_{ik} and the variables in $pa(X_i)$ take their j-th configuration w_{ij}, and N_{ij} is the number of instances in the dataset T where the variables in $pa(X_i)$ take their j-th configuration w_{ij}.

Taking for example the Bayesian network BN described in Figure 1, the loglikelihood of a dataset composed of one sample such as $T = (a_1, b_2, c_1, d_2, e_2)$ would be equal to

$$\begin{aligned} LL(BN|T) = log_2(P(A = a_1) \cdot P(B = b_2|A = a_1, E = e_2) \cdot \\ \cdot P(C = c_1|B = b_2) \cdot P(D = d_2|A = a_1) \cdot P(E = e_2)) = \\ = log_2(0.99 \cdot 0.9 \cdot 0.5 \cdot 0.3 \cdot 0.25) = -4.9 \end{aligned}$$

It is important to notice that datasets are usually composed by multiple samples, and that the final loglikelihood is the sum of the loglikelihoods of each sample.

Using the same formulation, the dimensional penalty of model M can be expressed as

$$|M| = \sum_{i=1}^{n} (r_i - 1) q_i$$

In the example of Figure 1, it would be thus:

$$|BN| = (1)_{penaltyA} + (4)_{penaltyB} + (2)_{penaltyC} + (2)_{penaltyD} + (1)_{penaltyE} = 10$$

In the canonical representation, the final AIC score is expressed as:

$$AIC = -2 \cdot (LL - |M|)$$

It is interesting to notice how the decomposability of the AIC makes it particularly feasible for use in a fitness function.

2.3 Bayesian Network Structure Learning

Learning the structure of a Bayesian network starting from a dataset is a NP-hard problem [2], that becomes more and more challenging as the size of the learning dataset decreases. The algorithmic approaches devised to solve this problem can be divided into two main branches: score-and-search heuristics and deterministic algorithms that rely upon statistical considerations on the learning set.

In recent years, Bayesian network structure learning gained more and more attention from the evolutionary community. Several attempts to tackle the problem has been tested, ranging from evolutionary programming [6], to hybrid approaches [7], to cooperative coevolution [3] [4].

The underlying assumption of these works is often the availability of a considerable number of samples in the dataset used for learning: under these conditions, however, a class of deterministic algorithms known as dependency analysis is able to deliver results of high quality in a fraction of the time. One of the most performing algorithms in this category is known as *Greedy Thick Thinning* (GTT)[11]. Starting from a completely connected graph, first GTT applies the well-known PC algorithm [12], that cuts arcs on the basis of conditional independence tests; then, it starts first adding and then removing arcs, scoring the network after each modification and using a set of heuristics to avoid a premature convergence. GTT implementations can be found in products such as GeNie/SMILE [13].

When the number of samples available for structure learning is small, however, dependency analysis algorithms see their performance degrade dramatically, leaving a promising open area of applicability for evolutionary techniques.

3 Proposed Methodology

The proposed approach to Bayesian network structure learning belongs to the category of score-and-search techniques: the evolutionary core is a state-of-the-art evolutionary algorithm, while the scoring metric used is a decomposition of the AIC.

3.1 μGP

μGP^3 [14] is an evolutionary algorithm tool developed by the CAD Group of Politecnico di Torino and available as a GPL software [15].

The main difference between μGP^3 and the classical Genetic Programming paradigm [16], is the encoding of individuals in tagged graphs instead of trees. More precisely, the algorithm makes use of constrained tagged graphs, that is, directed graphs where every element may own one or more tags, and that in addition have to respect a set of constraints. A tag is a name-value pair whose purpose is to convey additional information about the element it belongs to. Tags are used to add semantic information to graphs, augmenting the nodes with a number of parameters, and also to uniquely identify each element during the evolution. The constraints may affect both the information contained in the graph elements and its structure. Graphs are modified by genetic operators, such as the classical mutation and recombination, but also by different operators, as required. The activation probability and strength for every operator is an endogenous parameter. The genotype of every individual is described by one or more constrained tagged graphs, each of which is composed by one or more sections. Sections allow defining a global structure for the individuals that closely follows the structure of any candidate solution for the problem.

Constraints limit the possible productions of the evolutionary tool, and also provide them with semantic value. A user-defined XML configuration file encodes the constraints, which in turn provide the genotype-phenotype mapping for the generated individuals, describe their possible structure and define which values the existing parameters (if any) can take.

Individuals' fitness is computed by means of an external evaluator: this is usually a script that runs a simulation using the individual as input and collects the results, but may be any program able to provide the evolutionary core with proper feedback. The fitness of an individual is represented by a sequence of floating point numbers optionally followed by a comment string. This is currently used in a prioritized fashion: one fitness A is considered greater than another fitness B if the nth component of A is greater than the n-th component of B and all previous components (if any) are equal; if all components are equal then the two fitnesses are considered equal.

Given the versatility and the ease of configuration of the tool, it is not surprising that several successful applications of μGP^3 appear in literature, mainly in the area of assembly language generation [17], but also in very different fields, such as software verification [18].

The ability of evolving directed graphs, with genetic operators already designed to work on them, is particularly promising when dealing with problems where the individual is in fact a graph, as in the case of Bayesian network structure learning. A specific type of parameter, `innerBackwardLabel`, used in assembly generation for jumps to previous lines of the code, is now exploited to describe an oriented arc coming from a previous node. Other features of the algorithm are of particular use in this situation: a simple self-adapting mechanism avoids the need of setting fixed values for the genetic operators, increasing the activation probability for operators that constantly produce good offspring. A map of the individuals at genotype-level is also used to remove possible clones before the evaluation step.

The possibility of choosing the categories of genetic operators to apply is also advantageous: for this experience, two kinds of `crossover` and three `mutations` have been selected, as shown in Table 1. The difference between the crossovers is in the number of points of cut (one-point and two-point). The chosen mutations, operating on `innerBackwardLabel` type parameters, can collectively perform the addition, removal or variation of directed arcs in the graph. It is interesting to notice how the same mutation operators, applied to parameters such as floating point numbers, behave in a different fashion, changing the real-valued parameter according to a Gaussian distribution. Figure 2 shows an example of the effect of the `onePointCrossover` genetic operator on two individuals containing directed arcs.

3.2 Individual Encoding

A candidate solution to the Bayesian network structure learning problem is a directed acyclic graph. Most evolutionary approaches take into account the possibility of generating loops, by either using repairing operators or discarding

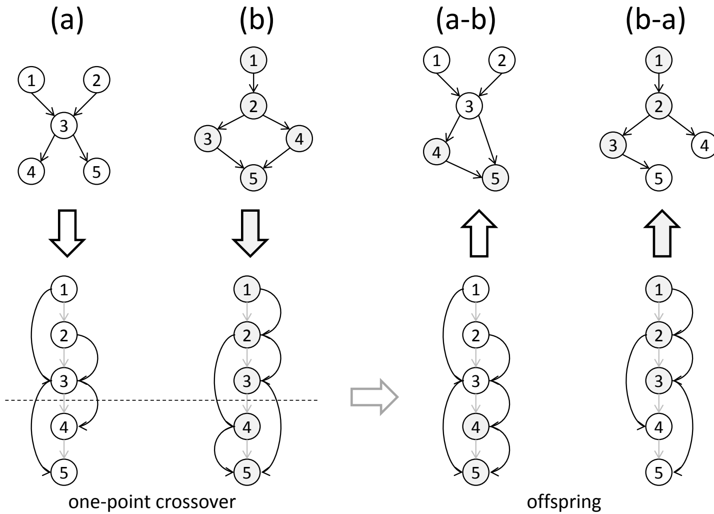


Fig. 2. Example of `onePointCrossover` application in μGP^3. When the genotype is of fixed length, the directed arcs are reattached to corresponding parts of the new individuals.

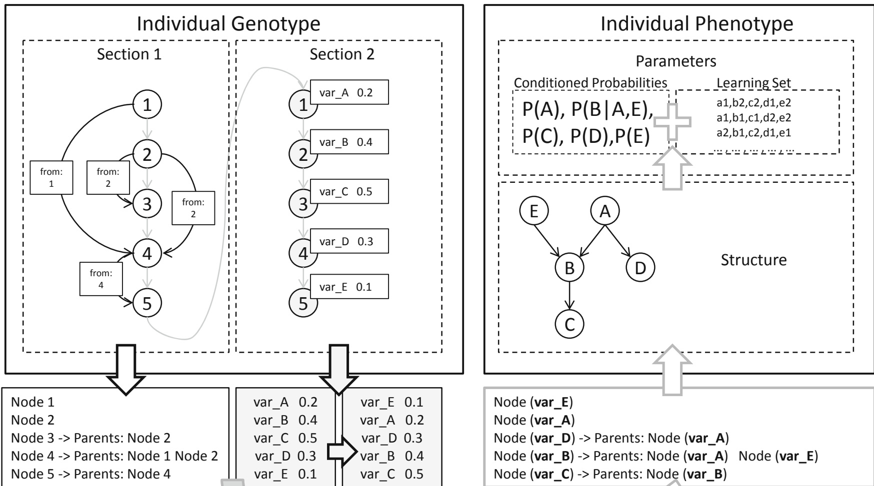


Fig. 3. Example of individual encoding in the proposed approach. The first part of the genome (Section 1) describes a directed acyclic graph, correct by construction. The second part (Section 2) specifies the mapping of the variables in the dataset on the nodes of the graph. The resulting phenotype is a graph with the structure of the first part of the genotype and the node ordering of the second. The parameters of the resulting Bayesian network are computed on the learning set, starting from the conditional probabilities derived from the structure.

graphs containing cycles. Even detecting the presence of a cycle, however, is non-trivial and computationally expensive.

Thanks to the options available in μGP^3, it is in fact possible to set the evolutionary algorithm to always generate valid structures. The nodes of the graph, in the individual description, will appear in a given order: it is sufficient to constraint the generation of directed arcs from one node to nodes that only appear after it in the individual description, and loops are automatically avoided by construction.

The mapping from the variables that appear in the learning dataset to the directed acyclic graph is encoded in the second part of the individual. Each variable is associated with a floating point weight ranging from 0 to 1: when the individual is evaluated, the variables are sorted with their weight and are subsequently mapped in-order to the graph structure described in the first part. An example of individual is reported in Figure 3.

The maximum number of parents for a single node is set to 10, as it is common for most search-and-score metaheuristics [3].

3.3 Fitness Function

Preliminary experiments with the canonical AIC scoring show a tendency for the algorithm to explore prevalently local optima of very simple structures: probably, in the AIC fitness landscape, the slope towards low complexity is much steeper than the one towards good loglikelihood values.

To avoid premature convergences, the fitness function used in the experiments is thus a hierarchical decomposition of the AIC. First, the loglikelihood of the model with respect to the learning dataset is considered; if two candidate models have the same loglikelihood, they are then compared on the penalty tied to their respective dimension.

4 Case Study

Over the course of time, the Bayesian network structure learning community developed a vast number of benchmarks, from simple to relatively complex. To assess the proposed approach, the Alarm network [19] is selected. Alarm was constructed for monitoring patients in intensive care, and it is effectively used. It features 37 nodes and 46 edges, and its overview can be found for example at `http://www.norsys.com/netlib/alarm.htm`

5 Experimental Results and Discussion

Experts usually agree that a comprehensive learning set for a Bayesian network should include at least 10-15 samples for each parameter. In the case of Alarm, this would mean the availability of thousands of samples: since the proposed approach is aimed at working with limited information, datasets smaller by an order of magnitude are selected.

Table 1. Parameters used for μGP^3 in all the performed experiments. λ is the number of genetic operators applied at each step. The `stagnation` stop condition forces the end of the execution if the best individual in the population does not change for a specified number of generations. For further details on the parameters, see [14].

Parameter	Value
μ	1,000
λ	1,000
Selection	`tournamentSelection` with τ = (1,4)
Diversity preservation	`fitnessHole` (0.5)
Stop condition	`stagnation` (100 generations)
Genetic operators	`singlePointCrossover`, `twoPointCrossover`, `alteratioMutation`, `replacementMutation`, `singleParameterAlteratioMutation`

Thus, to assess the validity of the methodology, three learning sets of 100 samples each are generated starting from the Alarm network description available in literature [19]. `alarm-100-a` is produced using the standard methodology of constrained probability generation; `alarm-100-b` contains 100 samples randomly taken from a dataset of original size 5,000; and `alarm-100-c` contains 100 samples randomly selected from a dataset of original size 10,000. It is important to note that, albeit randomly created, each 100-samples dataset contains all possible values for each variable in the Alarm network. In principle, all the datasets should have the same probability of being representative of the original network.

μGP^3 is set with the parameters reported in Table 1. Note that, since the algorithm is self-adapting, it is not necessary to initially select an activation probability for the genetic operators. The strenght of the genetic operators is also self-adapted during the evolution, with an initial value of $\sigma = 0.9$.

For each learning set (`alarm-100-a`, `alarm-100-b`, `alarm-100-c`), 20 runs of the proposed approach are executed, along with a run of *Greedy Thick Thinning*. Since the latter is deterministic, it reports the same output even for repeated runs.

Results are reported in Table 2. The first four lines show the performance of the true structure of the Alarm network, for comparison. It is interesting to notice the significant difference in loglikelihood values between the same structure, when the parameters are learned from different datasets (lines 2-4). Even knowing the true structure of the network, a dataset of considerable size would be needed to learn the true values for all the parameters. Learning algorithms of all categories can only compute an approximation of the parameters, that degrades with the reduction in size of the learning set. The grayed cells show the dataset the method is trained on.

The first evidence is that all solutions found by the proposed approach have lower dimensional penalties, much closer to the true structure than the ones delivered by GTT. While the loglikelihood values seem to favor GTT, it becomes evident that the greedy algorithm overfits the learning data: the networks delivered by GTT show loglikelihoods that are higher than the corresponding values of the original structure with parameters learned from the same dataset.

Table 2. Results of the experiments. While the logliklihood of solutions obtained with *Greedy Thick Thinning* is generally higher, it shows more overfitting when compared to the loglikelihood of the true structure with parameters trained on the same dataset, due to the higher dimension of the networks found.

Method	Dimensional Penalty		Loglikelihood alarm-100-a		Loglikelihood alarm-100-b		Loglikelihood alarm-100-c	
	Avg	StDev	Avg	StDev	Avg	StDev	Avg	StDev
True network	509.0	-	-1,571.90	-	-1,476.43	-	-1,470.95	-
True structure (a)	509.0	-	-1,651.05	-	-1,693.29	-	-1,672.12	-
True structure (b)	509.0	-	-1,807.32	-	-1,567.74	-	-1,688.81	-
True structure (c)	509.0	-	-1,809.77	-	-1,720.54	-	-1,569.26	-
GTT-100a	796.0	-	-1,641.61	-	-1,808.63	-	-1,801.35	-
GTT-100b	1,226.0	-	-1,979.28	-	-1,494.18	-	-1,825.90	-
GTT-100c	1,238.0	-	-1,918.89	-	-1,869.84	-	-1,498.90	-
μgp-100a	763.39	42.54	-1,644.68	30.39	-1,861.99	34.06	-1,848.50	31.06
μgp-100b	702.29	95.18	-1,981.79	27.47	-1,529.00	18.94	-1,850.80	23.89
μgp-100c	669.00	96.21	-1,986.72	33.40	-1,873.79	26.82	-1,554.09	10.68

When dealing with such a small amount of data, all stochastic considerations that are at the base of the class of deterministic algorithms GTT belongs to, simply do not hold anymore: even relatively effective statistics tests yield wrong results. In such a difficult situation, an effective score-and-search metaheuristic can provide better results.

For a final remark on computational times, as anticipated in Subsection 2.3, a run of GTT lasts a few seconds. A single generation of the proposed approach, on the same machine, takes between 1 and 2 minutes to complete, with the global time for the whole process amounting to hours. In less than 5% of the runs, the algorithm is also restarted, after delving deep into parts of the fitness landscape with high dimension, thus increasing the evaluation time over an acceptable threshold. Since the main focus of the present work is not on efficiency, it must be noted that previous computations of conditional probability statements are not stored and reused; the possibility of parallel evaluations of individuals in the same generation is also not exploited.

6 Conclusions

Bayesian network structure learning from a dataset is a complex problem, especially when the learning set is small: but in many real-world problems, particularly in the food processing and research industry, it is possible to have access to limited-size datasets only.

This paper presents an approach to Bayesian network structure learning from datasets of limited size. The methodology is based on an evolutionary algorithm able to evolve directed graphs, that can be easily set to always produce acyclic graphs by construction, avoiding the repairing of non-valid structures.

The proposed methodology is assessed on a well-known benchmark, the Alarm network, and compared against a state-of-the-art dependency analysis algorithm, known as *Greedy Thick Thinning*. The experimental results show that the approach is effective, obtaining simpler structures, for which interpretation and validation by humans are more feasible. The proposed approach also avoids the likelihood overfitting on the learning set of the greedy algorithm.

Future works will compare the proposed approach with other evolutionary solutions in literature on small datasets. The effect of different metrics or combination of metrics, such as Maximum Description Length or Bayesian Information Criterion, on the scoring of the candidate solutions will be also explored, along with the possibility of using cooperative coevolution to split the original task in sub-problems.

References

1. Robinson, R.: Counting unlabeled acyclic digraphs. In: Little, C. (ed.) Combinatorial Mathematics V. Lecture Notes in Mathematics, vol. 622, pp. 28–43. Springer, Heidelberg (1977), doi:10.1007/BFb0069178
2. Chickering, D.M., Geiger, D., Heckerman, D.: Learning bayesian networks is np-hard. Technical Report MSR-TR-94-17, Microsoft Research, Redmond, WA, USA (November 1994)
3. Carvalho, A.: A cooperative coevolutionary genetic algorithm for learning bayesian network structures. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1131–1138. ACM, New York (2011)
4. Wong, M.L., Lee, S.Y., Leung, K.S.: Data mining of bayesian networks using cooperative coevolution. Decis. Support Syst. 38, 451–472 (2004)
5. Barriere, O., Lutton, E., Wuillemin, P.H.: Bayesian network structure learning using cooperative coevolution. In: Genetic and Evolutionary Computation Conference, GECCO 2009 (2009)
6. Wong, M.L., Lam, W., Leung, K.S.: Using evolutionary programming and minimum description length principle for data mining of bayesian networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 21(2), 174–178 (1999)
7. Fournier, F., Wu, Y., McCall, J., Petrovski, A., Barclay, P.: Application of evolutionary algorithms to learning evolved bayesian network models of rig operations in the gulf of mexico. In: 2010 UK Workshop on Computational Intelligence (UKCI), pp. 1–6 (September 2010)
8. Barrière, O., Lutton, E., Baudrit, C., Sicard, M., Pinaud, B., Perrot, N.: Modeling Human Expertise on a Cheese Ripening Industrial Process Using GP. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 859–868. Springer, Heidelberg (2008)
9. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using bayesian networks to analyze expression data. In: Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, RECOMB 2000, pp. 127–135. ACM, New York (2000)
10. Akaike, H.: A new look at the statistical model identification. IEEE Transactions on Automatic Control 19(6), 716–723 (1974)
11. Cheng, J., Bell, D.A., Liu, W.: An algorithm for bayesian belief network construction from data. In: Proceedings of AI & STAT 1997, pp. 83–90 (1997)

12. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search, 2nd edn. MIT Press Books, vol. 1. The MIT Press (2001)
13. Druzdzel, M.J.: SMILE: Structural modeling, inference, and learning engine and GeNIe: A development environment for graphical decision-theoretic models, pp. 902–903. American Association for Artificial Intelligence (1999)
14. Sanchez, E., Schillaci, M., Squillero, G.: Evolutionary Optimization: the uGP toolkit. Springer, Heidelberg (2011)
15. SourceForge: Host of μgp3, http://sourceforge.net/projects/ugp3
16. Koza, J., Poli, R.: Genetic programming. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies, pp. 127–164. Springer, US (2005), doi:10.1007/0-387-28356-0_5
17. Squillero, G.: Microgp - an evolutionary assembly program generator. Genetic Programming and Evolvable Machines 6, 247–263 (2005)
18. Gandini, S., Ruzzarin, W., Sanchez, E., Squillero, G., Tonda, A.: A framework for automated detection of power-related software errors in industrial verification processes. J. Electron. Test. 26, 689–697 (2010)
19. Beinlich, I.A., Suermondt, H.J., Chavez, R.M., Cooper, G.F.: The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In: Second European Conference on Artificial Intelligence in Medicine, London, Great Britain, vol. 38, pp. 247–256. Springer, Berlin (1989)

Efficient Phenotype Evaluation in Cartesian Genetic Programming

Zdeněk Vašíček and Karel Slaný

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence, Brno, Czech Republic
{vasicek,slany}@fit.vutbr.cz

Abstract. This paper describes an efficient acceleration technique designed to speedup the evaluation of candidate solutions in Cartesian Genetic Programming (CGP). The method is based on translation of the CGP phenotype to a binary machine code that is consequently executed. The key feature of the presented approach is that the introduction of the translation mechanism into common fitness evaluation procedure requires only marginal knowledge of target CPU instruction set. The proposed acceleration technique is evaluated using a symbolic regression problem in floating point domain. It is shown that for a cost of small changes in a common CGP implementation, a significant speedup can be obtained even on a common desktop CPU. The accelerated version of CGP implementation accompanied with performance analysis is available for free download from http://www.fit.vutbr.cz/~vasicek/cgp

Keywords: Cartesian genetic programming, Fitness evaluation, Acceleration, Symbolic regression.

1 Introduction

During the last two decades the evolutionary computing community has demonstrated the evolutionary algorithms can deliver very efficient and sometimes also patentable implementations of various design problems [1,2]. For example, John Koza, the pioneer of the field, dealing primarily with the evolutionary design of analog circuits, has reported tens of human-competitive results in various areas of science and technology [3]. Although the evolutionary optimization and synthesis has been shown to be a promising method, there exist problems that make this approach problematic in several applications. Among others, the runtime needed to evaluate a fitness function (i.e. calculate a fitness value for a given candidate solution) represents a serious issue [4]. As complex candidate solutions usually require evaluating a huge number of training vectors the evaluation represents the main bottleneck of the whole evolutionary system.

Many techniques for accelerating the evaluation of fitness function have been proposed including the optimization at the algorithmic level (e.g. better representation for trees in GP [5]), cluster-based accelerators [3], problem-specific hardware accelerators implemented in reconfigurable programmable devices [6,7,8]

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 266–278, 2012.
© Springer-Verlag Berlin Heidelberg 2012

and implementations based on general purpose Graphic Processing Unit (GPU) architectures [9,10]. A common feature all of these approaches is the presence of a particular degree of parallelism implemented at different levels.

Recently, GPUs that are available in common desktop computers have been used to parallelize the fitness function evaluation [11,12,10]. According to the published results, the hardware-based accelerators together with GPU-based accelerators provide interesting speedup [13]. Although a lot of care is taken to optimize these hardware-based accelerators, nearly no effort is given to optimize the fitness evaluation procedure on common CPUs. The researchers usually overlook the fact, that modern CPUs implement a very high degree of parallelism directly at the instruction level. The general meaning is that it is very difficult to optimize the EAs on CPU and moreover, that the time-consuming optimization will not pay off in the end (i.e. the speedup will be only marginal). On the other hand, common desktop CPUs are equipped with SIMD instruction sets with continuously increasing number of data bits that can be processed in parallel. While the Multimedia Extension (MMX) instruction set introduced in 1996 uses 64-bit operands, contemporary instruction sets such as Advanced Vector Extension allow processing 256-bit data types. If we look at the current situation, every 64-bit desktop CPU, such as the AMD64 architecture, mandatory implements Streaming Extensions instruction sets such as SSE/SSE2 allowing delegation of a substantial part of generic ALU and FPU workload into specialized and highly optimized SIMD units.

According to Poli, to go beyond the speedup provided by the language, some drastic changes are required [14]. In this paper, we would like to show that this statement is not necessarily true. The goal of this paper is to introduce semi-automatic approach addressing the problem of speeding up the fitness function evaluation on commonly available CPUs. In particular, a system for acceleration of evaluations of candidate solutions in Cartesian Genetic Programming is presented. The method described in this paper is illustrated on the AMD64 CPU architecture, but it is applicable on any type of CPU present in current PCs. According to our best knowledge, there exists only one paper that addresses the similar problem. In [15] a genome compiler addressing the problem of acceleration of s-expression in GP has been proposed. In this paper, an approach suitable for linear genetic programming and CGP is presented.

The paper is organized as follows. Section 2 deals with the Cartesian Genetic Programming and the existing methods used for evaluation of candidate solutions. Proposed approach dealing with the acceleration of evaluation in CGP is introduced in Section 3. Results of experiments are presented in Section 4 and discussed in Section 5. Conclusions are given in Section 6.

2 Cartesian Genetic Programming

Cartesian genetic programming [16], introduced by Julian Miller and Peter Thomson in 2000, is a variant of genetic programming where the genotype is represented as a list of integers that are mapped to directed oriented graphs.

CGP encodes a candidate solution (typically a circuit or a program) using an array consisting of $C \times R$ configurable nodes. The C determines the number of columns whereas R determines the number of rows. Each programmable node has fixed number of inputs E_I and outputs E_O and can implement one of F predefined primitive functions; in most cases $E_I = 2$ and $E_O = 1$. The main feature of CGP is that all the parameters including the number of programmable nodes, node inputs and outputs and program inputs, P_I, and program outputs, P_O, are fixed. Each node input can be connected either to the output of a node placed in the previous L columns or to one of the program inputs. The parameter L (referred to as l-back parameter) defines the level of connectivity and thus reduces or extends the search space. For example, if L=1 only neighboring columns may be connected. Because of the complicated evaluation, feedback is not allowed in the standard version of CGP.

2.1 Fitness Function Evaluation

In order to evaluate the fitness function, the response for each training vector has to be calculated. This step involves the interpretation of a CGP genotype for each vector. One of the key features of CGP encoding is that it can directly be used as an intermediate code that is processed by an interpreter. Two types of interpreters are usually utilized. The interpreter based on recursion and linear interpreter.

The interpreter based on the recursion works as follows. The encoded graph structure is executed by recursion, starting from the output nodes down through the functions, to the input nodes [9]. In this way, the unconnected nodes are not processed and do not affect the performance of the evaluation. The calculated values are stored in local stacks and propagated upwards.

The linear interpreter works in the opposite direction. The execution of the encoded graph starts from the first node and continues according to the increasing node index. This scheme represents the most efficient implementation as it does not introduce any overhead due to function calling that have to manipulate with stack. In contrast with the recursive approach, all the output values are calculated in one pass. However, all the nodes are evaluated even if they are not connected. In order to improve the performance, a simple preprocessing step that marks the utilized nodes only can be introduced. Only the marked nodes are subsequently evaluated.

Both of these interpreters are applicable for absolute [16] as well as relative CGP encoding [9].

2.2 Common Linear CGP Interpreter

Let us assume that the goal is to evaluate a candidate program having P_I primary inputs that is encoded using $C \times R$ nodes. To simulate a candidate program, a temporary array consisting of $P_I + CR$ data items is needed. In fact, this array stores the calculated output value for each node as well as the values at the

primary inputs. Thus, it can be directly addressed by the indices stored in the chromosome.

The evaluation works as follows. Firstly, the first P_I items are initialized using the input values obtained from training data. Secondly, the `evaluateCGP` method is executed. The structure of the linear CGP interpreter for $E_I = 2$ and $E_O = 1$ that calculates the response for a single training vector situated in *data*[0] ... *data*[P_I-1] looks as follows.

```
void evaluateCGP(int *chromosome, datatype *data) {
  int nodeidx = PI;
  foreach(node, chromozome) {
     if (!node_used(nodeidx)) { nodeidx++; continue; }
     switch (node.func) {
        case 0x0:
          F0(data, node.in1, node.in2, nodeidx++); break;
        case 0x1:
          F1(data, node.in1, node.in2, nodeidx++); break;
        ...
     }
  }
}
```

The interpreter consists of a loop that calculates the response for each CGP node according to the genes of the CGP chromosome. CGP encodes a candidate solution using a triplet consisting of three integers. The first two integers (denoted as 'node.in1' and 'node.in2') determine the indices of nodes connected to currently evaluating node; the last integer (denoted as 'node.func') determines the function of the currently evaluating node.

In order to use the CGP, a function set has to be defined. The described approach enables to use the primitive inline functions (such as addition, subtraction, etc.) as well as complex functions by changing the macro definitions F0, F1, etc. In some cases it is also suitable to replace each node function by an indirect function call driven by an array of pointers to called functions which allows a dynamic change of the function set during the code execution by rewriting the content of the array.

The example of a fitness function required for guiding the search strategy is shown in the following procedure. The fitness value is calculated as the sum of absolute differences between obtained response and desired response.

```
datatype fitnessCGP(int  *chromosome) {
  datatype *data[PI + C*R];
  datatype fitness = 0
  for (int i=0;i<vectors;i++) {
     copydata(data, training_input[i], PI);
     evaluateCGP(chromosome, data);
     fit += abs(data[chromosome[3*C*R]] - training_desired[i]);
  }
  return fit;
}
```

The fitness function calls the `evaluateCGP` procedure which, given the calculated outputs by CGP and a candidate solution, evaluates the response for a single input vector (i.e. a single fitness case stored in the array denoted as 'training_input'). Then, according to the information about the output connections stored in the chromosome, the fitness value of a genotype for the utilized input vector is calculated. These steps are repeated until the last training vector is evaluated.

It is clear that the time needed to calculate the fitness value t_{eval} increases with the increasing number of training vectors and increasing number of evaluated nodes, i.e. $t_{eval} \propto N_v \cdot R \cdot C$ where N_v denotes the number of training vectors. While in case of symbolic regression problem N_v can be chosen by designer, $N_v = 2^{P_I}$ fitness cases have to be evaluated in a typical digital-circuit evolution.

3 Proposed Method

In order to improve the performance of the CGP system, it is important to simulate candidate solutions effectively. Intuitively, the performance of the CGP-based evolutionary system will be significantly improved if the interpreter is avoided and replaced by a native machine code that directly calculates the fitness value. The straightforward approach is to use a compiler and compile the procedure that calculates response for a single training vector into a high-level language such as C. This approach was used on GPUs by Harding in [9,10] where the generated C description supplemented with the evaluation procedure is compiled with a common C compiler and is run just like any other program. However, the transformation of the phenotype into C source code and the subsequent compilation represent the main bottleneck. In order to improve the overall performance, Harding uses a cluster of GPUs and CPUs to overlap the time needed to generate and compile C description with evaluation.

To maximize the overall performance, we propose a method that is able to efficiently compile the CGP genotype to an efficient binary machine code without necessity to call the external C compiler. Moreover, as it will be shown, the proposed method does not require deep knowledge of the target CPU instruction set. The CGP chromosome is compiled once when it is necessary to evaluate the fitness function. The process of fitness function calculation consists of two phases. In the first phase, the CGP chromosome is compiled or more precisely translated to machine code that resides in the application's address space. The process of translation exhibits linear time complexity with respect to the number of CGP nodes. The second phase involves the calculation of response for the training vectors and fitness value. This procedure executes the obtained machine code for each training vector.

3.1 Data Execution Prevention Handling

Although common computers store both data and instructions in the same memory, modern operating systems implement several security mechanisms that prevent an application from executing code from a non-executable memory region.

These techniques are known as data execution prevention (DEP) and are available on all operating systems including various favors of Unix, Mac OS as well as Windows.

Since the proposed method is based on the runtime compilation, it relies on the presence of executable code in data memory space. In order to avoid the raising of exception caused by DEP, the application has to request the hosting OS to mark the desired memory pages as executable. For example, this can be done by calling the `mprotect()` system function in POSIX systems.

3.2 Machine Code Generation

The code given in the following listing illustrates the principle of the proposed method.

```
unsigned char *code[CODE_SIZE] \
  __attribute__ ((aligned (4096)));

function initializeCGP() {
  mprotect(code, CODE_SIZE, PROT_READ|PROT_WRITE|PROT_EXEC);
}

function compileCGP(int *chromozome) {
  unsigned char *instr_ptr = code;
  /* preserve all used registers on stack excluding rax */
  *instr_ptr++ = 0x57; // push rdi
  ...
  /* translate CGP code */
  ...
  /* restore stored registers */
  ...
  *instr_ptr++ = 0x5F; // pop rdi
  /* return from function */
  *instr_ptr++ = 0xC3; // ret
}

function evaluateCGP() {
  typedef unsigned long (* func)(void);
  ((func) code)();
}
```

The code consists of three procedures related to the initialization, compilation and evaluation. During the initialization, several pages that are utilized for generated machine code are allocated in data space and marked as executable.

The process of CGP genotype compilation contains three sections. Since the generated code must take care of the C function call conventions (i.e. preserving register values, stack management and return value passing conventions), the code that preserves the utilized registers is generated in the first section. Then, the CGP chromosome is interpreted and corresponding machine code is generated. The code that translates CGP chromosome to the machine code looks nearly the same as the structure of linear CGP interpreter described in Section 2.2. The only difference is that instead of evaluation of each node, corresponding machine code is generated. Finally, the instructions restoring the values of previously stored registers are generated.

The evaluation procedure `evaluateCGP` consists of a simple call. This procedure represent a direct replacement of the procedure presented in Section 2.2. Note that the phenotype is translated only once, when the first training vector ought to be evaluated.

3.3 Translating Primitive Functions into Machine Code

Suppose one needs to generate machine code for a given primitive function. The function may be specified in C in the following way:

```
void add(int *data, int in1, int in2, int out) {
    data[out] = data[in1] + data[in2];
}
```

This function represents a basic building block of CGP interpreter that performs an action on an array of intermediate values using indexes which have been determined from a corresponding CGP node. Following the procedure *evaluateCGP* described in Section 2.2, the given primitive function can be utilized as follows:

```
#define F0(data, in1, in2, out) add(data, in1, in2, out)
```

The straightforward approach for obtaining a corresponding machine code would be to implement such a code in assembler (i.e. according to the binary code of the instruction, which can be found in the processor documentation, convert the given code into binary machine code). However this represents a tedious and time consuming manual process that may be discouraging.

In this paper, we present a more convenient approach. The main idea is to utilize a common compiler supporting in-line assembly statements (e.g. standard GCC compiler) to get the corresponding machine code automatically. Assume that the goal is to obtain a machine code for the given primitive function. One can utilize the assembly statements that tell the compiler which registers are used for passing given parameters. Let us suppose, for example, that register `r8` will be used for value of variable `in1`, register `r9` for value of variable `in2`, register `r10` for output index `out` and index register `rdi` for pointer to the temporary array `data`. Then, the following code can be used to generate the machine code that evaluates the primitive function.

```
void add(void) {
  register int *data asm("rdi");
  register long  in1 asm("r8"),  in2 asm("r9"), out asm("r10");
  data[out] = data[in1] + data[in2];
}
```

If we compile the given source code with debug information, the standard object dump tool can be used to display the corresponding binary machine code together with the human readable instructions. The generated code is depicted in the following listing.

```
0000000000000000 <add>:
   0:   42 8b 04 8f     mov    (%rdi,%r9,4),%eax
   4:   42 03 04 87     add    (%rdi,%r8,4),%eax
   8:   42 89 04 97     mov    %eax,(%rdi,%r10,4)
   c:   c3              retq
   d:   0f 1f 00        nopl   (%rax)
```

The generated machine code can be directly applied as it contains the required registers. These steps may be done by hand or automatically by a simple script.

The advantage of this approach is that it enables to translate simple as well as complex functions. Another advantage is that the user can enable the compiler optimizations that can produce even better code without affecting source or target registers.

3.4 Translating CGP Phenotype into Machine Code

The translation procedure iterates over the nodes. For each node contributing to the phenotype, a corresponding machine code is generated. The generated machine code firstly passes all necessary arguments to the primitive function machine code obtained according to the instructions in Section 3.3.

The similar approach can be utilized to prepare the code which passes the arguments to the primitive function code. For example, the following assembly statements can be used to obtain all the necessary assignments.

```
    register int *data asm("rdi") = (int *) 0x1000000000000001;
    register long in1 asm("r8") = 0x1000000000000002,
    in2 asm("r9") = 0x1000000000000003,
    out asm("r10") = 0x1000000000000004;
```

Then by overwriting the constant values with desired index values the machine code may be adjusted according to the requirements of the evaluated node.

3.5 Machine Code Vectorization

To exploit the performance of modern CPUs as much as possible, additional technique can be applied. The evaluation of training vectors can be parallelized using a SIMD instruction set offered by the target architecture. The use of SIMD instructions introduces another level of code optimization resulting in even higher speedup. The same approach proposed in previous paragraph can be utilized to obtain vectorized machine code. The only difference is that a suitable compiler intrinsics have to be used. The following GCC SSE/SSE2 shows an example which allows parallelization of four integer additions.

```
void add_vec(void) {
  register int *data asm("rdi");
  register long in1 asm("r8"), in2 asm("r9"), out asm("r10");
  *(__m128i *)&data[out] =
       _mm_add_epi32(*(__m128i *)&data[in1], *(__m128i *)&data[in2]);
}
```

4 Experimental Setup

In order to evaluate the speedup of the proposed approach, a symbolic regression problem in floating point domain was chosen. Note that the problem to be used is not important in this evaluation, because the time of evaluation depends mainly on the number of training vectors, number of evaluated nodes and complexity of primitive functions. The goal of this evaluation is to show, that it is worth to modify the common interpreted evaluation procedure and introduce the machine code compilation; i.e. the absolute values of speedup are not important.

We have investigated several factors affecting the performance including the size of training set and influence of floating point precision (i.e. single and double precision). The training set S contains 1000, 10000, and 100000 training vectors. Each training vector consists of one input value and one desired output value, both floating point numbers. The fitness function is calculated as an absolute error

$$fitness = \sum_{i=1}^{|S|} |c(S_i) - r(S_i)|$$

where $c(S_i)$ denotes calculated response for i-th training vector and $r(S_i)$ represents desired response for i-th training vector.

The following experimental setup was utilized. Standard CGP strategy with the population consisting of 8 individuals was used. For all experiments, the runtime needed for execution (i.e. evaluation) of 1000 generations was measured. Each CGP node can implement one of the six primitive functions $F = \{x+y, x-y, x*y, (x+y)/2, \min(x,y), \max(x,y)\}$. A linear CGP structure consisting of one row ($R = 1$) and 25, 50 and 1000 columns (C) was used. The l-back parameter was set to the maximum value, i.e. $l = C$. The experiments were performed on the Intel Core2 CPU E8400 running at 3.00 GHz In order to evaluate the impact of various parameters, we have arranged a set of three experiments.

Firstly, we have investigated the speedup of the proposed phenotype evaluation approach based on machine code translation. The generated machine code contains SSE/SSE2 SIMD instruction calls operating with 128-bit vectors which may process four or two floating point numbers at once, depending on their precision.

Then, the impact of removing unused CGP nodes (i.e. the nodes that does not contribute to the resulting phenotype) before phenotype evaluation is also investigated. In the first scenario, the unused nodes are removed; in the second scenario, all CGP nodes are evaluated (compiled to machine code) regardless of whether they contribute to the resulting value or not.

Finally, the impact of the processed data type on the speed of the evolution was measured. Two common data types were used – single precision 32-bit floating point and double precision 64-bit floating point numbers.

Table 1. Average time needed to evaluate 1000 generations (shown in seconds) for the single precision floating point data type

		without removal of unused nodes			with removal of unused nodes		
node count	training vectors	interpreted version	proposed version	speedup	interpreted version	proposed version	speedup
25	100	0.4 ± 0.01	0.02 ± 0.00	20.6 ± 2.03	0.1 ± 0.04	0.01 ± 0.00	12.7 ± 2.85
25	1000	4.2 ± 0.08	0.11 ± 0.01	38.3 ± 3.32	1.5 ± 0.65	0.09 ± 0.02	17.0 ± 5.15
25	10000	42.3 ± 0.81	0.98 ± 0.11	43.7 ± 4.80	13.1 ± 6.31	0.79 ± 0.22	15.9 ± 4.91
50	100	0.9 ± 0.01	0.04 ± 0.00	21.0 ± 2.04	0.2 ± 0.07	0.02 ± 0.00	8.3 ± 2.51
50	1000	8.8 ± 0.07	0.16 ± 0.01	54.5 ± 1.71	1.6 ± 0.70	0.09 ± 0.01	16.4 ± 5.12
50	10000	87.4 ± 0.38	1.31 ± 0.05	66.9 ± 2.50	13.0 ± 5.25	0.73 ± 0.10	17.2 ± 5.30
100	100	1.8 ± 0.12	0.09 ± 0.00	20.1 ± 1.32	0.3 ± 0.13	0.04 ± 0.01	6.3 ± 2.67
100	1000	17.3 ± 0.23	0.27 ± 0.01	64.8 ± 1.71	2.1 ± 1.18	0.11 ± 0.02	17.6 ± 6.81
100	10000	172.2 ± 1.25	2.11 ± 0.11	81.8 ± 4.41	20.3 ± 9.29	0.93 ± 0.28	21.0 ± 4.59
1000	100	18.2 ± 0.10	0.83 ± 0.01	21.7 ± 0.20	0.6 ± 0.22	0.27 ± 0.01	2.4 ± 0.74
1000	1000	178.5 ± 0.32	2.15 ± 0.01	83.1 ± 0.62	4.5 ± 2.29	0.36 ± 0.03	11.7 ± 5.19
1000	10000	1780.1 ± 3.93	15.23 ± 0.07	117.0 ± 0.33	38.6 ± 21.39	1.30 ± 0.29	27.5 ± 10.40

5 Experimental Results and Discussion

The experimental results for the single precision floating point numbers and double precision floating point numbers are summarized in Table 1 and Table 2. The runtimes were calculated as average from 50 independent runs (standard deviations aslo given). According to the obtained results, the proposed approach exhibits a significant speedup for all test cases.

Let us look at the first scenario where all nodes are evaluated (the column denoted as ‘without removal of unused nodes‘). We can identify that for a given size of CGP instance, the speedup increases with the increasing number of training vectors. This behavior is caused by the utilization of caches and the locality of training data. It can be also seen that for interpreted evaluation and a certain number of CGP nodes, the evaluation time increases linearly with the increasing number of training vectors as it has been expected. This trend is also observable for a given number of training vectors and increasing number of node counts.

On the other hand, we do not observe such correlation for data collected from runs where the unused nodes were omitted from the evaluation and where the number of nodes is relatively small. The problem is that the total number of utilized and evaluated nodes is relatively small thus the overhead introduced by the translation to machine code probably affects the overall performance. Nevertheless, a significant speedup has been achieved in both cases. In this scenario, the runtimes also exhibits large deviances. This behavior is also expectable because the number of utilized nodes is determined by the initial seed (which is generated randomly) and noticeably fluctuates during the evolution.

While the interpreter-based evaluation exhibits similar runtimes regardless of the utilized data type (i.e. float or double), the proposed evaluation method

Table 2. Average time needed to evaluate 1000 generations (shown in seconds) for the double precision floating point data type

		without removal of unused nodes			with removal of unused nodes		
node count	training vectors	interpreted version	proposed version	speedup	interpreted version	proposed version	speedup
25	100	0.4 ± 0.01	0.03 ± 0.00	13.6 ± 1.28	0.2 ± 0.07	0.02 ± 0.00	6.6 ± 2.43
25	1000	4.3 ± 0.19	0.19 ± 0.03	22.4 ± 2.88	1.5 ± 0.71	0.16 ± 0.04	9.0 ± 2.39
25	10000	42.3 ± 1.14	1.81 ± 0.24	23.7 ± 2.42	13.5 ± 6.38	1.46 ± 0.39	8.7 ± 2.34
50	100	0.9 ± 0.02	0.06 ± 0.00	15.0 ± 0.25	0.2 ± 0.07	0.03 ± 0.00	6.6 ± 1.99
50	1000	8.9 ± 0.10	0.28 ± 0.01	32.0 ± 1.41	1.9 ± 0.67	0.16 ± 0.02	11.1 ± 2.84
50	10000	88.8 ± 1.18	2.48 ± 0.12	35.9 ± 1.84	15.8 ± 7.21	1.47 ± 0.31	10.3 ± 3.02
100	100	1.8 ± 0.01	0.11 ± 0.00	16.1 ± 0.08	0.2 ± 0.09	0.04 ± 0.01	4.9 ± 1.78
100	1000	17.7 ± 1.11	0.46 ± 0.01	38.5 ± 2.49	2.0 ± 0.87	0.18 ± 0.03	10.4 ± 3.51
100	10000	172.9 ± 1.01	3.98 ± 0.12	43.5 ± 1.25	17.7 ± 8.36	1.56 ± 0.35	10.7 ± 3.78
1000	100	18.5 ± 1.15	1.03 ± 0.01	18.1 ± 1.18	0.6 ± 0.27	0.27 ± 0.02	2.2 ± 0.84
1000	1000	177.4 ± 1.11	3.65 ± 0.01	48.6 ± 0.40	3.7 ± 2.43	0.45 ± 0.06	7.8 ± 4.13
1000	10000	1775.7 ± 9.68	29.90 ± 0.12	59.4 ± 0.44	31.7 ± 19.19	2.17 ± 0.56	13.3 ± 5.60

utilizing the machine code exhibits two times larger runtime when dealing with the double data type in comparison with the float data type. These results confirm our expectations since the utilized SSE/SSE2 instructions can process four floats or two doubles in a single instruction call.

6 Conclusion

In this paper, we have introduced a framework that shows how to improve a classic interpreter-based phenotype evaluation in linear GP. The key feature of the proposed method is embedded translation of the genotype into machine code which is subsequently executed on the CPU. No external compiler is used to translate chromosomes into executable code. The described method introduces significantly lower overhead than the methods based on external compiler invocation proposed for GPUs. The executing of the generated machine code introduces no additional overhead as it is executed directly similarly to a function call. This stands in contrast with the GPU-based accelerators that introduce a very large overhead.

Another feature of the proposed method is that it requires only marginal knowledge of target CPU instruction set. The process of acquiring a partial machine code can be done mechanically and automatically.

Even if the proposed technique introduces some overhead when translating the phenotype into the executable machine code, it exhibits a significant speedup. As it has been demonstrated, the large problem instance the large speedup due to the data locatity and utilization of CPU's caching mechanism.

Note that the achieved speedup may be even larger when also the evaluation procedure is compiled to machine code. Moreover, assuming we have a regression problem that utilizes small data types such as char or short, the evaluation

procedure can benefit from the wide SIMD packed types. Thus, several operations can be executed using a single instruction. The typical example of such application is the evolutionary design of image filters.

Acknowledgments. This work was supported by the Czech science foundation project P103/10/1517, the research programme MSM 0021630528, the BUT project FIT-S-11-1 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

1. Koza, J.R.: Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines 11(3-4), 251–284 (2010)
2. Miller, J., Job, D., Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines 1(1), 8–35 (2000)
3. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
4. Haddow, P., Tyrrell, A.: Challenges of evolvable hardware: past, present and the path to a promising future. Genetic Programming and Evolvable Machines 12, 183–215 (2011)
5. Handley, S.: On the use of a directed acyclic graph to represent a population of computer programs. In: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, vol. 1, pp. 154–159 (1994)
6. Sekanina, L., Friedl, S.: An evolvable combinational unit for FPGAs. Computing and Informatics 23(5), 461–486 (2004)
7. Glette, K., Torresen, J.: A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device. In: Moreno, J.M., Madrenas, J., Cosp, J. (eds.) ICES 2005. LNCS, vol. 3637, pp. 66–75. Springer, Heidelberg (2005)
8. Vasicek, Z., Sekanina, L.: Hardware Accelerators for Cartesian Genetic Programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 230–241. Springer, Heidelberg (2008)
9. Harding, S.: Evolution of image filters on graphics processor units using cartesian genetic programming. In: 2008 IEEE World Congress on Computational Intelligence, Hong Kong, pp. 1921–1928. IEEE Computational Intelligence Society, IEEE Press (2008)
10. Harding, S., Banzhaf, W.: Implementing cartesian genetic programming classifiers on graphics processing units using GPU.NET. In: GECCO 2011: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 463–470. ACM, New York (2011)
11. Chitty, D.M.: A data parallel approach to genetic programming using programmable graphics hardware. In: GECCO 2007: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1566–1573. ACM Press, London (2007)
12. Harding, S., Banzhaf, W.: Fast Genetic Programming on GPUs. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 90–101. Springer, Heidelberg (2007)

13. Vasicek, Z., Sekanina, L.: Hardware accelerator of cartesian genetic programming with multiple fitness units. Computing and Informatics 29(7), 1359–1371 (2010)
14. Poli, R., Langdon, W.B.: Sub-machine-code genetic programming. In: Advances in Genetic Programming, ch. 13, vol. 3, pp. 301–323. MIT Press (1998)
15. Fukunaga, A., Stechert, A., Mutz, D.: A genome compiler for high performance genetic programming, pp. 86–94. University of Wisconsin, Morgan Kaufmann (1998)
16. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)

Author Index

Adamatzky, Andrew 37
Agapitos, Alexandros 109

Brabazon, Anthony 85, 134
Bull, Larry 37

Carreiras, João M.B. 218
Castle, Tom 1
Clarke, Tim 170
Correia, João 73
Cotillon, Alban 13

Dracopoulos, Dimitris C. 25

Effraimidis, Dimitrios 25

Gonçalves, Ivo 218

Hemberg, Erik 85
Howard, Gerard David 37
Hyde, Matthew R. 158

Jackson, David 49
Johnson, Colin G. 1
Johnston, Mark 121
Jurdak, Raja 13

Kim, Dong-Kyun 230
Kim, MinHyeok 230
Krawiec, Krzysztof 61
Kuyucu, Tüze 242

Lutton, Evelyne 254

Machado, Penousal 73
Matyáš, Vashek 194
McKay, Robert Ian (Bob) 230
Melo, Joana B. 218
Miller, Julian F. 170
Murphy, Eoin 85

Neshatian, Kourosh 97
Nguyen, Quang Uy 109
Nguyen, Su 121
Nguyen, Xuan Hoai 109, 230
Nicolau, Miguel 85, 134

Olmo, Juan Luis 146
O'Neill, Michael 85, 109, 134
Osborne, Bruce 134
Özcan, Ender 158

Parkes, Andrew J. 158
Pereira, Francisco B. 206

Reuillon, Romain 254
Romero, José Raúl 146
Romero, Juan 73

Saunders, Matthew 134
Seaton, Tom 170
Sekanina, Lukáš 182, 194
Shimohara, Katsunori 242
Šikulová, Michaela 182
Silva, Sara 218
Slaný, Karel 266
Smolka, Tobiáš 194
Squillero, Giovanni 254
Švenda, Petr 194

Tan, Kay Chen 121
Tanev, Ivan 242
Tavares, Jorge 206
Tonda, Alberto Paolo 254

Valencia, Philip 13
Vašíček, Zdeněk 266
Ventura, Sebastián 146

Wuillemin, Pierre-Henri 254

Zhang, Mengjie 97, 121