Carlos A. Coello Coello Vincenzo Cutello
Kalyanmoy Deb Stephanie Forrest
Giuseppe Nicosia Mario Pavone (Eds.)

LNCS 7491

Parallel Problem Solving from Nature - PPSN XII

12th International Conference
Taormina, Italy, September 2012
Proceedings, Part I

Springer

Lecture Notes in Computer Science 7491

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

Carlos A. Coello Coello Vincenzo Cutello
Kalyanmoy Deb Stephanie Forrest
Giuseppe Nicosia Mario Pavone (Eds.)

Parallel Problem Solving from Nature - PPSN XII

12th International Conference
Taormina, Italy, September 1-5, 2012
Proceedings, Part I

Springer

Volume Editors

Carlos A. Coello Coello
CINVESTAV-IPN, Mexico City, Mexico
E-mail: ccoello@cs.cinvestav.mx

Vincenzo Cutello
Giuseppe Nicosia
Mario Pavone
University of Catania, Italy
E-mail:{cutello, nicosia, mpavone}@dmi.unict.it

Kalyanmoy Deb
Indian Institute of Technology, Kanpur, India
E-mail: deb@iitk.ac.in

Stephanie Forrest
University of New Mexico, Albuquerque, NM, USA
E-mail: forrest@cs.unm.edu

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-32936-4 e-ISBN 978-3-642-32937-1
DOI 10.1007/978-3-642-32937-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012944753

CR Subject Classification (1998): J.3, I.2, F.1, F.2, I.4-5, G.2

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

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

This LNCS volume contains the proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN 2012). This biennial event constitutes one of the most important and highly regarded international conferences in evolutionary computation and bio-inspired metaheuristics. Continuing with a tradition that started in Dortmund, in 1990, PPSN 2012 was held during September 1–5, 2012 in Taormina, Sicily, Italy.

PPSN 2012 received 226 submissions from 44 countries. After an extensive peer-review process involving more than 230 reviewers, the Program Committee Chairs went through all the reports and ranked the papers according to the reviewers' comments. Each paper was evaluated by at least four reviewers. The top 105 manuscripts were finally selected for inclusion in this LNCS volume and for presentation at the conference. This represents an acceptance rate of 46%, which guarantees that PPSN will continue to be one of the most respected conferences for researchers working in natural computing around the world.

PPSN 2012 featured four distinguished keynote speakers: Angelo Cangelosi (University of Plymouth, UK), Natalio Krasnogor (University of Nottingham, UK), Panos M. Pardalos (University of Florida, USA), and Leslie G. Valiant (Harvard University, USA).

The meeting began with six workshops: "Evolving Predictive Systems" (Bogdan Gabrys and Athanasios Tsakonas), "Joint Workshop on Automated Selection and Tuning of Algorithms" Part A: Continuous Search Spaces—Focus on Algorithm Selection (Heike Trautmann, Mike Preuss, Olaf Mersmann, and Bernd Bischl), Part B: Discrete Search Spaces – Focus on Parameter Selection (Andrew Parkes and Ender Özcan), "Theoretical Aspects of Evolutionary Multiobjective Optimization: Interactive Problem Solving Sessions and New Results" (Dimo Brockhoff and Günter Rudolph), "Modeling Biological Systems" (Julia Handl, Joshua Knowles, and Yaochu Jin), and "Parallel Techniques in Search, Optimization, and Learning" (Enrique Alba and Francisco Luna). The workshops offered and ideal opportunity for the conference members to explore specific topics in evolutionary computation, bio-inspired computing, and metaheuristics in an informal and friendly setting.

PPSN 2012 also included eight tutorials: "Introduction to Bioinformatics" (Jaume Bacardit, University of Nottingham, UK), "Evolutionary Multi-Objective Optimization" (Jürgen Branke, University of Warwick, UK), "Implementing Artificial Evolution on GPGPU-Based Computing Eco-Systems with the EASEA-CLOUD Massively Parallel Platform" (Pierre Collet, Strasbourg University, France), "Programming by Optimization—A New Paradigm for Developing High-Performance Software" (Holger H. Hoos, University of British Columbia, Canada), "Computational Intelligence and Games" (Pier Luca Lanzi, Polytechnic of Milan, Italy), "Ant Colony Optimization" (Vittorio Maniezzo, University of Bologna,

Italy), "Complex Systems Science in Its Thirties" (Roberto Serra, University of Modena and Reggio Emilia, Italy), and "Expressive Genetic Programming" (Lee Spector, Hampshire College, USA).

We wish to express our gratitude to the authors who submitted their papers to PPSN 2012 and to the Program Committee members and external reviewers who provided thorough evaluations of all these submissions. We also express our profound thanks to Marisa Lappano Anile, Claudio Angione, Jole Costanza, Giovanni Carapezza, Giovanni Murabito, and all the members of the Organizing Committee for their substantial efforts in preparing for and running the meeting. Thanks to all the keynote and tutorial speakers for their participation, which greatly enhanced the quality of this conference. Finally, we also express our gratitude to all the organizations that provided financial support for this event.

September 2012

Carlos Coello Coello
Vincenzo Cutello
Kalyanmoy Deb
Stephanie Forrest
Giuseppe Nicosia
Mario Pavone

Organization

PPSN 2012 was organized and hosted by the Optimization and BioComputing Group of the Department of Mathematics and Computer Science, University of Catania, Italy. The University of Catania is the 29th oldest university in the world. Its establishment dates back to 1434.

Conference Committee

General Chairs

Vincenzo Cutello	University of Catania, Italy
Mario Pavone	University of Catania, Italy

Honorary Chair

Hans-Paul Schwefel	Technische Universität Dortmund, Germany

Program Chairs

Carlos A. Coello Coello	CINVESTAV-IPN, Mexico
Kalyanmoy Deb	Indian Institute of Technology, India
Stephanie Forrest	University of New Mexico, USA
Giuseppe Nicosia	University of Catania, Italy

Tutorial Chairs

Giuseppe Narzisi	Cold Spring Harbor Laboratory, USA
Germán Terrazas Angulo	University of Nottingham, UK

Workshop Chair

Alberto Moraglio	University of Birmingham, UK

E-Publicity Chairs

Heder Bernardino Soares	LNCC, Brazil
Fernando Esponda	Instituto Tecnologico Autonomo de Mexico, Mexico

Financial Manager

Marisa Lappano Anile	University of Catania, Italy

Local Organization

Giovanni Carapezza	University of Catania, Italy
Piero Consoli	University of Catania, Italy
Jole Costanza	University of Catania, Italy

Matteo De Felice	ENEA, Italy
Luigi Malagó	Politecnico di Milano, Italy
Giovanni Murabito	University of Catania, Italy
Annalisa Occhipinti	University of Catania, Italy
Elisa Pappalardo	University of Catania, Italy
Giovanni Stracquadanio	Johns Hopkins University, USA
Renato Umeton	University of Rome "La Sapienza", Italy

Steering Committee

Carlos Cotta	Universidad de Málaga, Spain
David W. Corne	Heriot-Watt University Edinburgh, UK
Kenneth A. De Jong	George Mason University, USA
Agoston E. Eiben	Vrije Universiteit Amsterdam, The Netherlands
Juan Julián Merelo Guervós	Universidad de Granada, Spain
Günter Rudolph	Technische Universität Dortmund, Germany
Thomas P. Runarsson	University of Iceland, Iceland
Robert Schaefer	University of Krakow, Poland
Marc Schoenauer	Université Paris Sud, France
Xin Yao	University of Birmingham, UK

Workshops

Evolving Predictive Systems
Bogdan Gabrys and Athanasios Tsakonas

Workshop on Automated Selection and Tuning of Algorithms
Part A: Continuous Search Spaces – Focus on Algorithm Selection
Heike Trautmann, Mike Preuss, Olaf Mersmann, and Bernd Bischl

Workshop on Automated Selection and Tuning of Algorithms
Part B: Discrete Search Spaces – Focus on Parameter Selection
Andrew Parkes and Ender Özcan

Theoretical Aspects of Evolutionary Multiobjective Optimization:
Interactive Problem Solving Sessions and New Results
Dimo Brockhoff and Günter Rudolph

Modeling Biological Systems Workshop
Julia Handl, Joshua Knowles, and Yaochu Jin

Parallel Techniques in Search, Optimization, and Learning
Enrique Alba and Francisco Luna

Tutorials

Introduction to Bioinformatics
Jaume Bacardit

Evolutionary Multi-Objective Optimization
Jürgen Branke

Implementing Artificial Evolution on GPGPU-Based Computing Eco-Systems with the EASEA-CLOUD Massively Parallel Platform
Pierre Collet

Programming by Optimization—A New Paradigm for Developing High-Performance Software
Holger H. Hoos

Computational Intelligence and Games
Pier Luca Lanzi

Ant Colony Optimization
Vittorio Maniezzo

Complex Systems Science in Its Thirties
Roberto Serra

Expressive Genetic Programming
Lee Spector

Keynote Speakers

Angelo Cangelosi	University of Plymouth, UK
Natalio Krasnogor	University of Nottingham, UK
Panos M. Pardalos	University of Florida, USA
Leslie G. Valiant	Harvard University, USA

Program Committee

Enrique Alba	Wolfgang Banzhaf	Hans-Georg Beyer
Youhei Akimoto	Helio Jose Barbosa	Mauro Birattari
Jaroslaw Arabas	Thomas Bartz-Beielstein	Christian Blum
Paolo Arena	Simone Bassis	Yossi Borenstein
Dirk Arnold	Roberto Battiti	Peter Bosman
Anne Auger	Gerardo Beni	Pascal Bouvry
Dogan Aydin	Heder S. Bernandino	Anthony Brabazon
Jaume Bacardit	Adam Berry	Jürgen Branke

Dimo Brockhoff
Will Browne
Larry Bull
Tadeusz Burczynski
Edmund Burke
Stefano Cagnoni
Erick Cantú-Paz
Luigi Cardamone
Uday Chakraborty
Kay Chen Tan
Tianshi Chen
Ying-ping Chen
Miroslav Chlebik
Sung-Bae Cho
Siang-Yew Chong
Carlos Coello Coello
David Corne
Ernesto Costa
Carlos Cotta
Peter Cowling
Matteo De Felice
Kalyanmoy Deb
Kenneth A. De Jong
Antonio Della Cioppa
Gianni Di Caro
Luca Di Gaspero
Federico Divina
Marco Dorigo
Benjamin Doerr
RafałDrężewski
Jérémie Dubois-Lacoste
Gusz Eiben
Aniko Ekart
Talbi El-Ghazali
Michael Emmerich
Aniko Ekart
Anton Ereemev
Anna I Esparcia-Alcázar
José Figueira
Steffen Finck
Carlos M. Fonseca
Giuditta Franco
Tobias Friedrich
Marcus Gallagher
Jonathan M. Garibaldi
Mario Giacobini
Adam Ghandar
Tobias Glasmachers
Faustino Gomez
Maoguo Gong
Salvatore Greco
Roderich Groß
Steven Gustafson
Walter Gutjahr
Pauline Haddow
Hisashi Handa
Nikolaus Hansen
Julia Handl
Jin-Kao Hao
Emma Hart
Verena Heidrich-Meisner
Philip Hingston
Andrew Hone
Matthew Hyde
Christian Igel
Pedro Isasi Viñuela
Hisao Ishibuchi
Christian Jacob
Thomas Jansen
Licheng Jiao
Yaochu Jin
Bryant A. Julstrom
Devis Karaboga
George Karakostas
Andy Keane
Graham Kendall
Joshua Knowles
Timo Koetzing
Krzysztof Krawiec
Halina Kwasnicka
Dario Landa-Silva
Pier Luca Lanzi
Jörg Lassig
Sanja Lazarova-Molnar
Per Kristian Lehre
Peter Lewis
Xiaodong Li
Tianjun Liao
Giosué Lo Bosco
Fernando Lobo
Daniele Loiacono
Manuel López-Ibáñez
Jose A. Lozano
Simon Lucas
Evelyne Lutton
Luigi Malagó
Jacek Mandziuk
Vittorio Maniezzo
Angelo Marcelli
Elena Marchiori
Benedetto Matarazzo
Matteo Matteucci
Giancarlo Mauri
Barry McCollum
Alexander Melkozerov
Juan Julián Merelo Guervós
Olaf Mersmann
Silja Meyer-Nieberg
Zbigniew Michalewicz
Martin Middendorf
Kaisa Miettinen
Orazio Miglino
Julian Miller
Sara Montagna
Orazio Miglino
Marco A. Montes de Oca
Alberto Moraglio
Alison A. Motsinger-Reif
Christian Müller
Giuseppe Narzisi
Boris Naujoks
Ferrante Neri
Frank Neumann
Una-May O'Reilly
Gabriella Ochoa
Gisele Pappa
Elisa Pappalardo
Luis Paquete
Andrew Parkes
Marco Pavone
Martin Pelikan
David Pelta
Clara Pizzuti
Silvia Poles

Petr Posík
Mike Preuss
Christian Prins
Adam Pruegel-Bennett
Günther Raidl
Vitorino Ramos
William Rand
Khaled Rasheed
Mauricio Resende
Katya Rodriguez
Eduardo A. Rodríguez Tello
Philipp Rohlfshagen
Andrea Roli
Günter Rudolph
Thomas Runarsson
Thomas A. Runkler
Conor Ryan
Erol Sahin
Michael Sampels
Ivo Sbalzarini
Robert Schaefer
Andrea Schaerf
Marc Schoenauer
Oliver Schütze
Michele Sebag
Bernhard Sendhoff
Roberto Serra
Marc Sevaux
Jonathan Shapiro
Moshe Sipper
Roman Slowinski
Christine Solnon
Terence Soule
Dipti Srinivasan
Catalin Stoean
Giovanni Stracquadanio
Thomas Stützle
Dirk Sudholt
Ponnuthurai Suganthan
Jerry Swan
Daniel Tauritz
Jorge Tavares
Andrea G.B. Tettamanzi
Madeleine Theile
Lothar Thiele
Dirk Thierens
Jon Timmis
Jerzy Tiuryn
Julian Togelius
Marco Tomassini
Heike Trautmann
Vito Trianni
Bianca Truthe
Elio Tuci
Andrew M. Tyrrell
Renato Umeton
Leonardo Vanneschi
Sebastien Verel
Carlos Martín Vide
Verel Carlos
Markus Wagner
Lipo Wang
Darrel Whitley
R. Paul Wiegand
Carola Winzen
Carsten Witt
Man-Leung Wong
John Woodward
Ning Xiong
Xin Yao
Gary Yen
Tina Yu
Yang Yu
Christine Zarges
Ivan Zelinka
Qingfu Zhang
Eckart Zitzler

Sponsor

ESTECO
IBM Italy
SolveIT Software Pty Ltd

Patronage

Angelo Marcello Anile Association
IET - The Institute of Engineering and Technology
Tao Science Research Center, Italy
UNINFO
University of Catania, Italy

Table of Contents – Part I

Theory of Evolutionary Computation

Machine Learning, Classifier Systems, Image Processing

Experimental Analysis, Encoding, EDA, GP

Applications (I)

Table of Contents – Part II

Multiobjective Optimization

Swarm Intelligence, Collective Behaviour, Coevolution and Robotics

Memetic Algorithms, Hybridized Techniques, Meta and Hyperheuristics

Applications (II)

Convergence of the IGO-Flow of Isotropic Gaussian Distributions on Convex Quadratic Problems

Tobias Glasmachers

Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany
tobias.glasmachers@ini.rub.de

Abstract. The information geometric optimization (IGO) flow has been introduced recently by Arnold et al. This distinguished mathematical flow on the parameter manifold of a family of search distributions constitutes a novel approach to the analysis of several randomized search heuristics, including modern evolution strategies. Besides its appealing theoretical properties, it offers the unique opportunity to approach the convergence analysis of evolution strategies in two independent steps. The first step is the analysis of the flow itself, or more precisely, the convergence of its trajectories to Dirac peaks over the optimum. In a second step it remains to study the deviation of actual algorithm trajectories from the continuous flow. The present study approaches the first problem. The IGO flow of isotropic Gaussian search distributions is analyzed on convex, quadratic fitness functions. Convergence of all trajectories to the Dirac peak over the optimum is established.

1 Introduction

Our theoretical understanding of evolution strategies (ESs) lags behind their practical successes. ESs are powerful optimization techniques that can work under adverse conditions, like non-smooth, discontinuous, or even noisy fitness functions. However, useful convergence guarantees exist only for the simplest algorithms on restricted problem classes [5,2]. Narrowing this gap between practically relevant problems and theoretical guarantees is a long-standing goal of evolutionary algorithms research.

In this context we view the recently introduced information geometric optimization (IGO) flow [1] as a promising tool towards a unified analysis of randomized search algorithms. Various invariance properties make this flow on the parameter manifold of a family of search distributions a canonical means for optimization. It can be interpreted as a continuous time version of various iterative, randomized search techniques. In particular, it resembles the behavior of existing evolution strategies [4,3] in the limit of large populations and small search strategy updates.

This hints at a two-step analysis: Convergence of the flow trajectories on an as large as possible class of problems should be separated from bounding

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 1–10, 2012.

the deviation of discrete trajectories of actual algorithms from the continuous trajectories of the flow. With the present work we progress towards the first goal. We provide a complete convergence analysis of the IGO flow of isotropic Gaussian distributions on convex, quadratic fitness functions to the Dirac peak over the optimum. This problem class is of prime interest, since it approximates local optima of twice differentiable fitness functions. The result is non-trivial, since there exist counter examples where the flow converges prematurely.

2 The Information Geometric Optimization Flow

The IGO flow is defined in the context of randomized search for the minimum of a fitness function $f : X \to \mathbb{R}$ in the black box model. Iterative, randomized search algorithms like evolutionary algorithms can be interpreted as defining a sequence of search distributions. The IGO flow resembles this process with a continuous time flow on the parameter manifold Θ of a family P_θ of search distributions (with densities p_θ). In the limit of large populations and small learning rates popular ESs such as CMA-ES [4] and NES [6,3] closely follow this flow [1]. The IGO framework lifts optimization from the search space X to the parameter manifold Θ. For example, for isotropic Gaussians the parameter space $\Theta = \mathbb{R}^d \times \mathbb{R}^+$ is composed of the mean vector and the standard deviation.

In a first step the fitness function is normalized w.r.t. the current search distribution, which also makes it invariant under monotonic transformations. We need the following notation. Let $B(x_0, r) = \{x \in \mathbb{R}^d \,|\, \|x - x_0\| < r\}$ denote the open ball of radius r around x_0. Let $u^f(y) = \{x \in \mathbb{R}^d \,|\, f(x) < y\}$ denote the sub-level sets of the fitness function, and let $q^f_\theta(y) = P_\theta(u^f(y))$ denote the (lower) quantile function, measuring the probability to sample a solution x with fitness $f(x)$ better than y under the search distribution encoded by θ. This function is assumed to be continuous.[1] Combining these definitions we write $u^f_\theta(q) = u^f(y)$ if $q = q^f_\theta(y)$. The composition $q^f_\theta \circ f$ assigns to each point the probability to sample a better point under P_θ. Importantly, this is a monotone, rank preserving transformation of the fitness function, which is itself (by construction) invariant under rank-preserving transformations of the fitness values.

In a second step a non-increasing weight function $w : [0, 1] \to \mathbb{R}$ is introduced that puts user-defined emphasis on different quantiles. A simple choice is the indicator $w = \mathbf{1}_{[0,q]}$ for some quantile q. The function $W^f_\theta = w \circ q^f_\theta \circ f$ is a monotonically decreasing transformation of f. Thus, for fixed θ, maximization of W^f_θ is equivalent to minimization of f. This is an objective function on the search space X, which can be transferred to the parameter manifold Θ in the form $J(\theta, \theta') = \mathbb{E}_{\theta'}[W^f_\theta(x)]$. For fixed θ this is an objective function in θ'.

The parameter manifold Θ is naturally equipped with the Fisher metric. Maximization in the resulting statistical manifold can be achieved locally by gradient

[1] Otherwise the subsequent technical analysis is unnecessarily complicated by the need to distinguish upper and lower quantiles, see e.g. [1], equation (3). The assumption is always fulfilled for the distributions and fitness functions considered in this paper.

ascent. The gradient in the inner geometry of distributions pulled back to the parameter manifold Θ is the natural gradient, denoted by the symbol $\widetilde{\nabla}$. Steepest ascent is thus realized by following the vector field $V(\theta) = \widetilde{\nabla}_{\theta'}|_{\theta'=\theta} J(\theta, \theta')$. The formula

$$V(\theta) = \int W_\theta^f(x)\, \widetilde{\nabla}_{\theta'} \log\left(p_{\theta'}(x)\right) dP_\theta(x) \tag{1}$$

(equation (10) in [1]) connects the vector field V to the natural gradient of the logarithmic density in equation (2). It also ensures the continuity of V, provided that p_θ is non-zero and continuous.

The IGO flow is the solution of the differential equation $\dot{\phi}^t(\theta) = V(\phi^t(\theta))$. Here $\phi^t(\theta)$ denotes a trajectory with initial condition $\phi^0(\theta) = \theta$. The upper index t denotes time. This flow is invariant under coordinate changes of θ and under rank-preserving (strictly monotone) transformations of fitness values [1].

The IGO vector field is defined by means of a natural gradient operator. However, its definition is *not* of the form $V(\theta) = \widetilde{\nabla}_\theta J'(\theta)$ for some potential function J'. The existence of such a potential function would greatly simplify the analysis of the IGO flow, but there are counter-examples where it does not exist. It remains unclear whether such a function exists for the case of isotropic Gaussians and convex, quadratic fitness function.

In this context it is worth mentioning that the family of NES algorithms [6,3] is commonly derived for the potential function $J'(\theta) = \mathbb{E}_\theta\left[f(x)\right]$ of expected fitness. It has been argued in [1] that practical NES algorithms follow the IGO flow instead. This is because NES algorithms are rendered invariant under rank-preserving transformations of the fitness function by a technique called fitness shaping. Expected fitness has the desirable property to be a potential function of the corresponding flow. However, its drawbacks are that depending on the fitness function (over which there is no control in a black box setting) the expectation may not always exist, the resulting flow is not invariant under monotone fitness transformations, and existing algorithms do not approximate the corresponding flow. Consequently we focus on the IGO flow in this study, albeit expected fitness has its merits, e.g., on finite search spaces (where the expectation always exists).

Isotropic Gaussian search distributions on $X = \mathbb{R}^d$ with densities

$$p_{\mu,\sigma}(x) = \frac{1}{(\sqrt{2\pi}\cdot\sigma)^d} \cdot \exp\left(-\frac{\|x-\mu\|^2}{2\sigma^2}\right)$$

are characterized by a mean vector $\mu \in \mathbb{R}^d$ and a variance $\sigma^2 \in \mathbb{R}^+$. In this paper we use the parameterization $\theta = (\mu, \sigma) \in \mathbb{R}^d \times \mathbb{R}^+ = \Theta$. For isotropic Gaussians the flow is invariant under translation, scaling, and rotation of the search space (provided that the initial conditions are transformed accordingly).

The natural gradient of the logarithmic density (see equation (1)) can be computed as

$$G(\theta, x) = \widetilde{\nabla}_\theta \log\left(p_\theta(x)\right) = \begin{pmatrix} x-\mu \\ \frac{\sigma}{4d}\left[\left(\frac{\|x-\mu\|}{\sigma}\right)^2 - d\right] \end{pmatrix} . \tag{2}$$

Analogously, the vector field is decomposed into components $V = (V_\mu, V_\sigma)$ describing the evolution of the mean and the standard deviation under the flow.

The IGO flow of Gaussian distributions is of particular interest for its connection to evolution strategies [1,4,3].

3 Analysis of the IGO Flow

We start with the core technical lemma. This auxiliary result decomposes the IGO vector field into additive components, with each component corresponding to a tractable, geometric problem.

Lemma 1. *The IGO flow vector field $V(\theta)$ can be written in the form*

$$V(\theta) = \int_{[0,1]} \left[\int_{u_\theta^f(q)} G(\theta, x) \; dP_\theta(x) \right] dg(q)$$

$$= \int_{[0,1]\times[0,\infty)} \left[\int_{u_\theta^f(q)\cap B(\mu,r)} G(\theta, x) \; dx \right] dh_\theta(q, r)$$

w.r.t. non-negative measures $g(q)$ and $h_\theta(q, r)$.

Proof. We rewrite $W_\theta^f(x) = \int_0^1 \mathbf{1}_{u_\theta^f(q)}(x)\, dg(q)$ as an integral of constant functions on sub-level sets of f (which are super-level sets of W_θ^f). Analogously, we rewrite $P_{\mu,\sigma} = \int_0^\infty U_{B(\mu,r)}\, d\beta_\sigma(r)$ as a superposition of uniform distributions $U_{B(\mu,r)}$ over balls around the center μ. By construction the measures g and β_σ are non-negative. Plugging both decompositions into equation (1) and choosing h_θ as the product of g and β_σ completes the proof. ∎

The above lemma allows us to analyze the IGO flow based on the natural gradient of the logarithmic density given by equation (2), restricted to the intersection of a ball with a sub-level set. For convex functions the integration area $u_\theta^f(q)\cap B(\mu, r)$ is convex (possibly empty). This lemma will be applied multiple times in the following.

3.1 Linear Objective Functions

The goal of minimization of a linear fitness function $f(x) = v^T x$ is to move the center of the distribution into the direction $-v$ as quickly as possible, and to drive the step size σ to infinity. The invariance properties of the IGO flow allow us to assume $v = (1, 0, \ldots, 0) \in \mathbb{R}^d$, $\mu = 0$, and $\sigma = 1$.

Using Lemma 1 we write $V_\mu(0, 1)$ as an integral over terms of the form $\int_{u_\theta^f(q)\cap B(0,r)} x\, dx$. The half-space $u_\theta^f(q)$ is given by the inequality $x_1 < y$, with $y = 0$ for the median ($q = 1/2$). The inner product of the above term with

v yields the same expression with integrand x_1 (first component of x) instead of x. There are three cases: The integral is zero if the ball is fully contained in or disjoint to the half-space. Otherwise it is negative. It follows $V_\mu(0,1) = (-c, 0, \ldots, 0) \in \mathbb{R}^d$ for some $c > 0$, and for symmetry and invariance reasons it holds $V_\mu(\mu, \sigma) = -\sigma \cdot c \cdot v$. Thus, the flow moves the center μ in direction $-v$. However, it may converge prematurely if σ decays too quickly.

Lemma 1 allows us to write the component $V_\sigma(0,1)$ as an integral over terms of the form $\int_{u_\theta^f(q)} (\|x\|^2 - d)\, dP_{(0,1)}(x)$. The expectation of the integrand over the whole space vanishes, and so it does (for symmetry reasons) restricted to the half-space $x_1 < 0$ ($q = 1/2$). However, for $x_1 < y$ with $y < 0$ ($q < 1/2$) the integral is positive, since compared to the half-space $x_1 < 0$ probability mass is missing particularly for shorter-than-average vectors x. It follows with an analog argument that the integral is negative for $y > 0$ ($q > 1/2$). Thus, depending on the choice of the weight function w, it is possible that $V_\sigma(0,1)$ is negative. In this case the step size σ decays exponentially, resulting in (premature) convergence of the IGO flow trajectories to Dirac delta peaks. For example, for the so-called "selection quantile" weight function $w(t) = \mathbf{1}_{[0,q]}(t)$ trajectories convergence prematurely for $q > 1/2$, and σ grows exponentially for $q < 1/2$. The ability to handle a (close to) linear objective function is a must for any reasonable optimization scheme. Care should be taken to impose sufficient selection pressure by the choice of the weight function. This assumption is formalized as follows:

Assumption. Let L be defined as $V_\sigma(0,1)$ for a linear objective function $f(x) = v^T x$ with slope $\|v\| \neq 0$. Using translation and scale invariance this is equivalent to $V_\sigma(\mu, \sigma) = \sigma \cdot L$. We assume in the following that w is chosen such that $L > 0$.

3.2 Convex Quadratic Objective Functions

The core contribution of the present work is the analysis of the IGO flow on objective functions of the form $f(x) = x^T Q x$, where $Q \in \mathbb{R}^{d \times d}$ is symmetric and positive definite. This situation is analyzed in the following lemmas.

Lemma 2. *V is scale invariant: it holds $V(\lambda \cdot \theta) = \lambda \cdot V(\theta)$ for all $\lambda > 0$.*

Proof. The lemma follows directly from equations (1) and (2) and the scale invariance of the level sets of $f(x) = x^T Q x$. ∎

As a consequence, the vector field V is fully described by its values on a section of co-dimension one through the equivalence classes $[\theta] = \mathbb{R}^+ \cdot \theta$ in Θ. The set $S = \left\{\theta \in \Theta \,\middle|\, \|\theta\| = 1\right\}$ is such a section, with $\|\cdot\|$ denoting the Euclidean two-norm on $\Theta \subset \mathbb{R}^{d+1}$.

Lemma 3. *For $\mu \neq 0$ the inner product $\langle V_\mu(\mu, \sigma), \mu \rangle$ is negative.*

Note that the above inner product is the time derivative of $\frac{1}{2}\|\mu\|^2$ under the flow (since by definition V_μ is the time derivative of μ). Thus, the center component μ moves towards the optimum, although not necessarily straight.

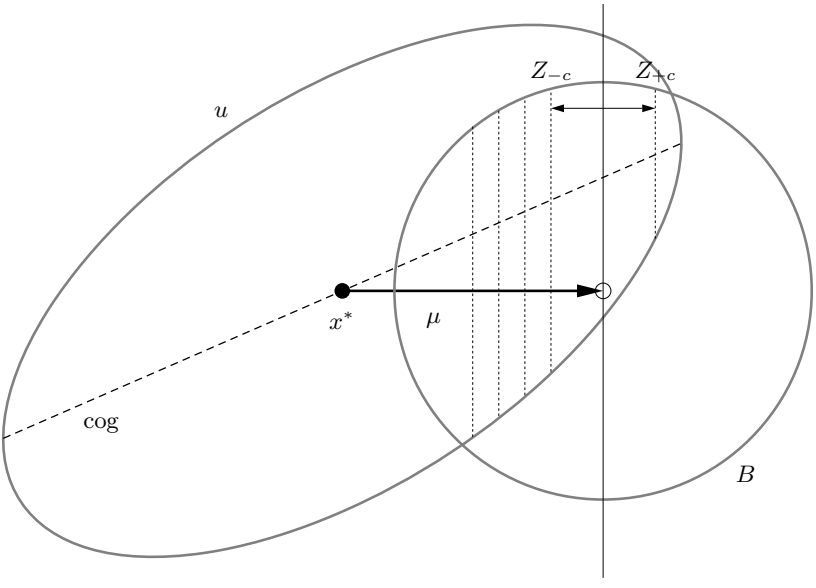


Fig. 1. The figure depicts the sets B (circular outline), u (elliptic outline), the optimum x^* in the origin, the mean vector μ (arrow), the hyperplane H_0 (vertical line), the parameterized line $\mathrm{cog}(c)$ of centers of gravity of $Y_c = H_c \cap u$ (dashed line), as well as a number of sets Z_c (vertical, dotted lines). Refer to the proof of Lemma 3 for further details.

Proof. This proof amounts to a non-trivial application of Lemma 1. The proof is based on an involved construction, see Figure 1.

Fix $q \in [0,1]$ and $r > 0$, and the corresponding sets $u = u_\theta^f(q)$ and $B = B(\mu, r)$. We define the hyperplanes $H_c = \{x \in \mathbb{R}^d \mid \langle x, \mu \rangle = \|\mu\|^2 + c\}$ orthogonal to μ, as well as their subsets $Y_c = H_c \cap u$ and $Z_c = Y_c \cap B$. Let $\mathcal{M}$ denote the (d-1)-dimensional Lebesgue measure on the hyperplanes H_c. Then the center of gravity of Y_c is defined as $\mathrm{cog}(c) = 1/\mathcal{M}(Y_c) \cdot \int_{Y_c} x \, dx$. For a convex, quadratic objective function the set u is the interior of an ellipsoid, and analogously, each set Y_c is the interior on an ellipsoid in $d-1$ dimensions. The centers of gravity $\mathrm{cog}(c)$ as a function of c form a parameterized line.

Recall that the μ-component of the natural gradient of the logarithmic density is $x - \mu$. The relevant expression for the application of Lemma 1 is the inner product of $x - \mu$ with μ. The sets Z_c form sections of $u \cap B$ such that $\langle x - \mu, \mu \rangle$ takes the constant value c. Now fix a positive constant $c > 0$ and consider the pair of sections Z_{+c} and Z_{-c}, as well as the translation $\psi_c : H_{+c} \to H_{-c}$ along the line $\mathrm{cog}(c)$.

By construction it holds $\psi_c(Y_{c+}) \subset Y_{-c}$, and again by construction it holds $\psi_c(Z_{+c}) \subset Z_{-c}$ (see Figure 1), and since the translation ψ_c is measure preserving

it follows $\mathcal{M}(Z_{+c}) \leq \mathcal{M}(Z_{-c})$. In those cases where Z_c is bounded by the ellipsoid and not only by the ball (these cases exist if $u \cap B \neq \emptyset$ and $u \cap B \neq B$) the inequality is strict, because the ellipsoid Y_{+c} is strictly smaller than for Y_{-c} (see also Figure 1).

The inner term of Lemma 1 projected onto the direction μ becomes

$$\left\langle \int_{u \cap B} (x - \mu)\, dx, \mu \right\rangle = \int_{-\infty}^{\infty} c \cdot \mathcal{M}(Z_c)\, dc$$
$$= \int_{0}^{\infty} c \cdot \big(\mathcal{M}(Z_{+c}) - \mathcal{M}(Z_{-c})\big)\, dc \; < \; 0 \; .$$

Finally, the application of Lemma 1 yields $\langle V_\mu(\mu, \sigma), \mu \rangle < 0$. ■

The next three lemmas analyze the evolution of the step size. Their proofs rely on the following types of topological arguments: Continuous functions map compact sets in the preimage onto compact sets in the image, and preimages of open sets are open. This implies two handy properties: First, a continuous function attains infimum and supremum on a compact set, which means that minimum and maximum exist. Second, if a continuous function is positive in one point, then it is positive in a (small) open neighborhood of this point.

We define the set $M = (\mathbb{R}^d \times \mathbb{R}_0^+) \setminus \{(0, 0)\}$ and the continuous[2] function $n : M \to [0, \infty]$, $n(\mu, \sigma) = \|\mu\|/\sigma$, measuring normalized distance of the search distribution to the optimum. Because of $n(\theta) = n(\lambda \cdot \theta)$ for all $\theta \in M$ and $\lambda > 0$ the function is uniquely described by its values on the compact half-sphere $\overline{S} = \big\{\theta \in M \,\big|\, \|\theta\| = 1\big\}$, which is the topological closure of the open half-sphere $S \subset \Theta$.

Lemma 4. *It holds $V(0, \sigma) = (0, -c \cdot \sigma)$ for some $c > 0$.*

Proof. We apply Lemma 1 to compute $V_\mu(0, \sigma)$. The sub-level set $u_\theta^f(q)$ as well as the ball $B(\mu, r) = B(0, r)$ are symmetric around the origin, and so is their intersection. The inner term in the integration is x, such that the integral over $u_\theta^f(q) \cap B(0, r)$ vanishes.

The form $V_\sigma(0, \sigma) = -c \cdot \sigma$ follows from Lemma 2. It remains to show that V_σ is negative. We apply Lemma 1 again and consider the inner term

$$\int\limits_{u_\theta^f(q)} \frac{\sigma}{4d} \left[\left(\frac{\|x\|}{\sigma} \right)^2 - d \right] dP_\theta(x) \; .$$

The integration, when spanning the whole search space, amounts to zero. However, the set $u_\theta^f(q)$ is convex and symmetric around the origin and thus puts more probability mass on smaller-than-average vectors. As a result the above expression is negative, and we obtain $V_\sigma(0, \sigma) < 0$ from Lemma 1. ■

Lemma 5. *There exists $c_1 < \infty$ such that $n(\mu, \sigma) > c_1$ implies $V_\sigma(\mu, \sigma) > 0$.*

[2] The set $[0, \infty]$ is equipped with the standard one-point-compactification topology.

Proof. The objective function $f(x) = x^T Q x$ is differentiable and can thus, locally, be approximated arbitrarily well by its first order Taylor expansion. Thus, for fixed $\mu \neq 0$ the fitness approaches an affine linear function with non-zero slope in the limit $\sigma \to 0$. The limit $\lim_{\sigma \to 0} V_\sigma(\mu, \sigma)/\sigma = L > 0$ exists for all $\mu \neq 0$, and V_σ/σ is continuous. This allows us to extend the domain of V_σ/σ as a continuous function from Θ to M, or analogously from S to $\overline{S}$. Let $S_\mu = \left\{\mu \in \mathbb{R}^d \,\middle|\, \|\mu\| = 1\right\}$ denote the unit sphere in $\mathbb{R}^d$. We use $\overline{n} = n|_{\overline{S}}$ as a shorthand notation for the function n restricted to $\overline{S}$. Then the pre-image of infinity under $\overline{n}$ takes the form $\overline{n}^{-1}(\infty) = S_\mu \times \{0\} \subset M$, and the function V_σ/σ has the constant value L on this set.

The continuity of V_σ/σ implies that there exists an open neighborhood $N \subset \overline{S}$ of $S_\mu \times \{0\}$ with $V_\sigma(\mu, \sigma)/\sigma > 0$ for all $(\mu, \sigma) \in N$. The set $\overline{S} \setminus N$ is compact, and therefore also its image $\overline{n}(\overline{S} \setminus N)$. By construction this set does not contain infinity. Thus, the choice $c_1 = \max\left(\overline{n}(\overline{S} \setminus N)\right)$ concludes the proof. ∎

Lemma 6. *There exists $c_2 > 0$ such that $n(\mu, \sigma) < c_2$ implies $V_\sigma(\mu, \sigma) < 0$.*

Proof. The proof is analogous to the previous one. Consider the point $(\mu, \sigma) = (0, 1) \in \overline{S}$. Lemma 4 implies $V_\sigma(0, 1) < 0$, and it holds $\overline{n}^{-1}(\{0\}) = \{(0, 1)\}$. From the continuity of V_σ we conclude the existence of an open neighborhood $N' \subset \overline{S}$ of $(0, 1)$ with $V_\sigma(\mu, \sigma) < 0$ for all $(\mu, \sigma) \in N'$. The set $\overline{n}(\overline{S} \setminus N') \subset [0, \infty]$ is closed and does not contain zero, which allows for the choice $c_2 = \min\left(\overline{n}(\overline{S} \setminus N')\right)$. ∎

Theorem 1. *For all $\theta_0 \in \Theta$ the IGO flow trajectory $\phi^t(\theta_0)$ converges to a Dirac peak over the optimum: It holds $\lim_{t \to \infty} \phi^t(\theta_0) = (0, 0)$.*

Proof. For $b > 0$ we define the open neighborhood $B \subset \overline{\Theta} = \mathbb{R}^d \times \mathbb{R}_0^+$ of $\theta^* = (0, 0) \in \overline{\Theta}$ as $B = \{(\mu, \sigma) \in \overline{\Theta} \,|\, \sigma < b, \|\mu\| < c_2 \cdot b\}$. Since b is arbitrary, showing that the trajectory $\phi^t(\theta_0)$ enters B in finite time and stays there will prove the statement. Based on lemmas 5 and 6 we split the parameter space into three dynamic regimes

$$
\begin{aligned}
R_1 &= \{\theta_1 \in \Theta \,|\, c_1 \leq n(\theta_1)\} \\
R_2 &= \{\theta_2 \in \Theta \,|\, c_2 \leq n(\theta_2) \leq c_1\} \\
R_3 &= \{\theta_3 \in \Theta \,|\, n(\theta_3) \leq c_2\}
\end{aligned}
$$

of qualitatively different behavior. The constraints imposed by the various lemmas on the vector field are illustrated in Figure 2. In particular, Lemma 3 implies that the flow can only shrink μ, which corresponds to the vector field pointing to the "left" in Figure 2. In addition, the vertical component is by Lemma 5 restricted to point "upwards" ($V_\sigma > 0$) in R_1, and according to Lemma 6 "downwards" ($V_\sigma < 0$) in R_3.

For initial conditions $\theta_1 \in R_1$, $\theta_2 \in R_2$, or $\theta_3 \in R_3$ we define compact sets C_1, C_2, and C_3 in which the trajectory $\phi^t(\theta_i)$ is restricted to stay for $t > 0$ according to the above conditions until it enters the set B. Figure 2 (right) illustrates these sets, which will be considered w.l.o.g. as closed (otherwise consider the closure).

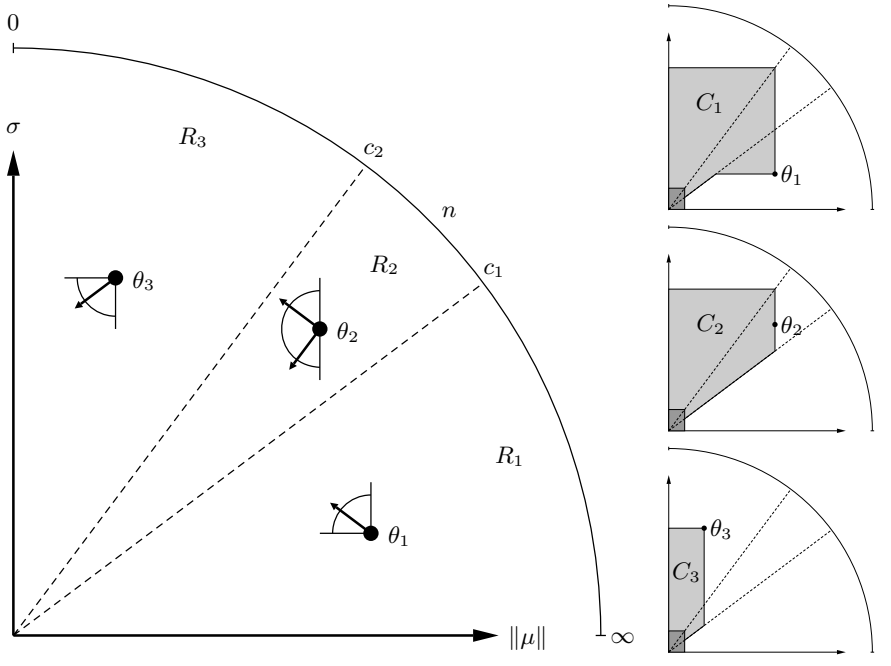


Fig. 2. Left: Illustration of the different dynamic regimes R_1, R_2, and R_3. The quarter-circles and the half-circle attached to the prototypical points $\theta_i \in R_i$, $i \in \{1, 2, 3\}$, illustrate how the vector field $V(\theta)$ is constrained by the various lemmas. Right: Illustration of the compact regions C_i (gray areas), in downscaled versions of the same figure. The second and third of the small figures also depict the open neighborhood B of $(\mu, \sigma) = (0, 0)$ (dark gray area).

They are compact, since they are also bounded away from infinity and from the boundary of Θ. These sets are split into

$$C_i' = \left\{ (\mu, \sigma) \in C_i \;\middle|\; \|\mu\| \geq \frac{c_2 \cdot b}{2} \right\} \quad \text{and} \quad C_i'' = \left\{ (\mu, \sigma) \in C_i \;\middle|\; \|\mu\| \leq \frac{c_2 \cdot b}{2} \right\} .$$

Lemma 3 together with the condition $\mu \geq c_2 \cdot b/2$ implies that restricted to the sets C_i' it holds $\langle V_\mu, \mu \rangle < 0$. Each of these sets is compact, and thus the maximum of this function exists, which is a negative value. This value provides a non-zero lower bound on the velocity of the movement of the trajectory towards smaller $\|\mu\|$ ("to the left" in Figure 2). Thus, the flow leaves the set C_i' in finite time. Assume the flow did not reach B, then it must enter the corresponding set C_i''. By construction, these compact sets are fully contained in regime R_3. There the function V_σ is negative, and with the same argument the maximum exists and is negative, which provides a lower bound on the velocity of the flow moving towards smaller σ ("downwards"). Thus, the flow enters B in finite time. The shape of B is constructed so that the flow stays inside (see Lemmas 3 and 6). ■

As a comment and without proof we want to add that the same compactness arguments give rise to the existence of a linear convergence rate.

4 Discussion

It has been proven that all trajectories of the IGO flow on isotropic Gaussian distributions converge to the Dirac peak over the optimum. Due to invariance properties this result holds for all convex quadratic functions and rank-preserving transformations thereof, given that the quantile weights are chosen so that the flow does not get stuck on a linear slope. The importance of this result is that it describes the dynamics of the flow in the proximity of local optima of twice differentiable fitness functions.

This is a promising result, although we view it rather as a first step. The author has good faith that most of the statements brought forward in the various lemmas can be generalized. This is because the proof techniques are kept as general as possible. In particular, geometric and topological arguments have been preferred over an algebraic treatment of the (linear or quadratic) objective function. Thus, large parts of the analysis should be generalizable, which holds in particular for the proof of the theorem.

This leaves us with a considerable body of future work. The analysis can be extended into different directions. First, the class of search distributions can be broadened. Gaussian distributions with fully adaptive covariance matrix are of primary interest, since the corresponding flow is resembled by state-of-the-art evolution strategies [1,4,3]. Second, the class of fitness functions can be extended. An ambitious goal is to cover the full class of all smooth, uni-modal problems. Third, the present understanding of how closely actual evolutionary algorithms follow the IGO flow is limited. The idea of transferring results from the IGO flow to evolution strategies drives the present ongoing investigation and is therefore a primary research goal.

References

1. Arnold, L., Auger, A., Hansen, N., Ollivier, Y.: Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles. Technical Report arXiv:1106.3708v1, arxiv.org (2011)
2. Auger, A.: Convergence results for the $(1, \lambda)$-SA-ES using the theory of φ-irreducible Markov chains. Theoretical Computer Science 334(1-3), 35–69 (2005)
3. Glasmachers, T., Schaul, T., Sun, Y., Wierstra, D., Schmidhuber, J.: Exponential Natural Evolution Strategies. In: Genetic and Evolutionary Computation Conference, GECCO (2010)
4. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation 9(2), 159–195 (2001)
5. Jägersküpper, J.: Analysis of a Simple Evolutionary Algorithm for Minimization in Euclidean Spaces. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1068–1079. Springer, Heidelberg (2003)
6. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural Evolution Strategies. In: Congress on Evolutionary Computation (CEC 2008), Hongkong. IEEE Press (2008)

Homogeneous and Heterogeneous Island Models for the Set Cover Problem

Andrea Mambrini[1], Dirk Sudholt[2], and Xin Yao[1]

[1] University of Birmingham, Birmingham, UK
[2] University of Sheffield, Sheffield, UK

Abstract. We propose and analyse two island models that provably find good approximations for the SetCover problem. A homogeneous island model running parallel instances of the SEMO algorithm—following Friedrich *et al.* (Evolutionary Computation 18(4), 2010, 617-633)—leads to significant speedups over a single SEMO instance, but at the expense of large communication costs. A heterogeneous island model, where each island optimises a different single-objective fitness function, provides similar speedups at reduced communication costs. We compare different topologies for the homogeneous model and different migration policies for the heterogeneous one.

Keywords: Parallel evolutionary algorithms, set cover, island model, theory, runtime analysis.

1 Introduction

Due to the current development in computer architecture and the steeply rising number of processors in modern devices, parallelisation is becoming a more and more important issue. Evolutionary algorithms (EAs) can be parallelised by using island models, also called coarse-grained EAs or multi-deme models [1,2]. Several subpopulations are evolved on different processors. Subpopulations coordinate their search by a process called migration, where selected individuals, or copies thereof, are sent to other islands. Migration often happens periodically or probabilistically and islands are typically connected by spatial structures such as rings or torus graphs [3]. Compared to panmictic populations, this decreases the spread of information. A slower spread of information can increase the diversity in the whole system, and by choosing the right topology and the frequency or probability of migration, the communication effort can be tuned.

Despite being applied and researched intensively, the theoretical foundation of parallel EAs is still in its infancy. Even the effect of the most fundamental parameters on performance is not well understood [1] and more research is needed to understand the search dynamics in island models [4]. Present theoretical studies include takeover times and growth curves (see, e. g., [5] or [1, Chapter 4]). Recently the expected running time of parallel EAs has been studied, leading to a constructed example where island models excel over panmictic populations [6,7] and examples where the diversity in island models makes crossover

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 11–20, 2012.

a powerful operator [8,9]. Also the speedup in island models has been studied rigorously: how the number of generations can be decreased by running multiple islands instead of one. Studies include pseudo-Boolean optimisation [10,11] and polynomial-time solvable problems from combinatorial optimisation [12].

These works form a solid foundation towards a theory of parallel metaheuristics, but they leave open many important questions. None of these works addresses how island models behave on general instances of NP-hard problems, or how they deal with multiobjective fitness functions. Furthermore, studies have been limited to homogeneous island models, where all islands run the same algorithm. In many settings heterogeneous models make more sense—islands can use different parameters, different operators, and even different fitness functions. This closely relates to the emerging area of hyper-heuristics [13].

In this work we propose and analyse homogeneous and heterogeneous island models for the SetCover problem. Given a set S with m elements and a collection of n subsets of S with associated costs, the SetCover problem asks for a selection of subsets of S that cover the whole set and have minimum cost. This classic NP-hard problem is one of the most fundamental problems in computer science. Friedrich *et al.* [14] studied a SEMO algorithm on a biobjective formulation of the problem and showed that SEMO efficiently computes an H_m-approximation, where $H_m = \sum_{i=1}^{m} \frac{1}{i}$ is the m-th Harmonic number.

We study a parallel version of this algorithm where each island runs an instance of SEMO. Each island use the same bi-objective fitness functions to be minimised: one criterion counting the number of uncovered elements and the other representing the cost of the selection. Each island stores a population of non-dominated solutions. At the end of each generation migration occurs transmitting a copy of the whole population to all neighbouring islands.

We show that this leads to significant speedups, depending on the topology and the migration probability, for probabilistic migration policies. However, this homogeneous island model has large communication costs as whole populations are exchanged between islands. To this end, we propose a heterogeneous island model that has a lower communication cost and islands run simpler algorithms.

The heterogeneous island model consists of $m + 1$ islands using different single-objective fitness functions. Each island stores one individual and runs a (1+1) EA (or RLS that just differs in using local instead of global mutation). The fitness functions are such that on island i only selections covering i elements are feasible. Therefore, each island i keeps the best individual covering i elements of S. The island model can be implemented on fewer than $m + 1$ processors by running multiple islands on each processor. We show that the collection of islands is able to guarantee the same performance and approximation quality as in the homogeneous model, but with lower communication costs and simpler operations. We also study different migration policies for the heterogeneous model and show how the migration policy affects running time and communication costs.

Due to space restrictions, many proofs are omitted or reduced to proof sketches.

2 Preliminaries

Let $S = \{s_1, \cdots, s_m\}$ be a set containing m elements and $C = \{C_1, \cdots, C_n\}$ be a collection of non-empty sets such that $C_i \subset S$ for $1 \leq i \leq n$ and $\bigcup_{i=1}^{n} C_i = S$. Each set C_i has a cost $c_i > 0$. We call $X = x_1 \cdots x_n$ a *selection* of C and we say that C_i is in the selection X iff $x_i = 1$. The optimal solution to the SetCover problem is an X such that $\bigcup_{i:x_i=1} C_i = S$ and $\sum_{i:x_i=1} c_i$ is minimum.

We define the following measures:

- $c(X) = |\bigcup_{i:x_i=1} C_i|$ is the number of covered elements of the selection.
- $|X|_1 = \sum_{i=1}^{n} x_i$ is the number of selected sets of a selection.
- $\text{cost}(X) = \sum_{i:x_i=1} c_i$ is the cost of a selection.
- $c_{\max} = \max_i c_i$ is the maximum cost of a set.
- $c_e(C_i, X) = \frac{\left|C_i \setminus \bigcup_{j:x_j=1} C_j\right|_1}{c_i}$ is the cost-effectiveness of a set w. r. t. X.

The homogeneous island model consists of an archipelago of μ islands each one running the SEMO algorithm, minimising the fitness function $f(X) = (m - c(X), \text{cost}(X))$. SEMO always maintains a set of non-dominated search points. New solutions are created by selecting uniformly a search point from the current population and mutating it. The offspring is added to the current population and then all dominated search points are removed. SEMO uses local mutations: one bit is chosen uniformly at random and then flipped. A variant called *global SEMO* uses standard bit mutations instead (called *global mutations*), flipping each bit independently with probability $1/n$. In the homogeneous island model based on SEMO or global SEMO (see Algorithm 1), each island maintains such a population. For migration, a copy of this whole set is transmitted to all neighbouring islands. The union of this set with the target island's set is considered and then all dominated solutions are removed. This way, the best solutions among source and target islands are maintained and combined.

Algorithm 1. Homogeneous island model based on (global) SEMO

1: Initialise $P^{(0)} = \{P_1^{(0)}, \ldots, P_\mu^{(0)}\}$, where $P_i^{(0)} = \{0^n\}$ for $1 \leq i \leq \mu$. Let $t := 0$.
2: **repeat forever**
3: **for** each island i **do in parallel**
4: Simulate one generation of (global) SEMO, updating $P_i^{(t)}$.
5: Send a copy of the population $P_i^{(t)}$ to all neighbouring islands.
6: Unify $P_i^{(t)}$ with all populations received from other islands.
7: Remove all dominated search points from $P_i^{(t)}$.
8: Let $t := t + 1$.

The heterogeneous island model consists of a fully connected archipelago of $m + 1$ islands indexed $0, \ldots, m$. Each island stores just one individual and runs an (1+1) EA (or RLS) using a single-objective function that is different on each island. For island i we define the fitness function (to maximise) as:

$$f_i(X) = \begin{cases} nc_{\max} - \text{cost}(X) & \text{if } c(X) = i \\ -|c(X) - i| & \text{if } c(X) \neq i \end{cases}$$

The idea is that island i stores an individual that represents the so far best selection covering i elements (referred to as *feasible*). If the solution does not cover i elements, the fitness is negative and hints are given towards covering i elements[1]. Each island is thus assigned a different part of the search space to optimise. This is similar to what happens in dynamic programming [15].

The heterogeneous island model is shown in Algorithm 2. Note that the heterogeneous island model can be easily implemented on $\mu \leq m$ processors by running up to $\lceil \frac{m+1}{\mu} \rceil$ islands on each processor.

Both island models are initialised with empty selections. This is a sensible strategy for SetCover and theoretical results [14] as well as preliminary experiments have shown that this only speeds up computation.

Algorithm 2. Heterogeneous island model based on (1+1) EA (or RLS)

1: Initialise the island individuals $X_0^{(0)}, \ldots, X_m^{(0)}$ to 0^n. Let $t := 0$.
2: **repeat forever**
3: **for** each island i **do in parallel**
4: Produce a global (or local) mutation $\tilde{X}_i^{(t)}$ of the individual $X_i^{(t)}$.
5: Send a copy of $\tilde{X}_i^{(t)}$ to each other island.
6: Choose $X_i^{(t+1)}$ with maximal f_i-value among $X_i^{(t)}$, $\tilde{X}_i^{(t)}$ and all immigrants.
7: Let $t := t + 1$.

The homogeneous and heterogeneous island models differ fundamentally in their search behaviour. Following Skolicki [16], we distinguish *intra-island evolution* (the evolution within each island) and *inter-island evolution* (evolution among and between islands). The homogeneous model uses intra-island evolution to generate improvements by mutation, and migration helps to propagate these improvements to other islands. The heterogeneous island model strongly relies on inter-island evolution; in fact, beneficial mutations as in the homogeneous model yield solutions that are only feasible on other islands. The two island models also differ in the population size. In the heterogeneous model the population of each island consists of just one individual, while in the homogeneous model the population size of each island is upper bounded by m. This generally means that the time and space required to compute a generation in the homogeneous model is larger than in the heterogeneous one.

We define the *parallel running time* as the number of generations of an island model until it has found a satisfactory solution, in our case an H_m-approximation. We also refer to the *sequential running time* as the product between the parallel running time and the number of islands. This represents the computational effort to simulate the model on a single processor. The *speedup* of an island model with μ islands is defined as the rate between the expected parallel running time of the island model and the expected running time of the same EA using only a single island. This kind of speedup is called *weak orthodox speedup* in Alba's taxonomy [17]. If the speedup is of order $\Theta(\mu)$, we speak of

[1] Our analysis holds for any negative function for the second case of f_i.

a *linear speedup*. Furthermore, we also consider the effort for performing migration. We define the *communication effort* as the total number of individuals sent between islands, throughout a run of an island model. The (expected) communication effort is given by the (expected) parallel time, multiplied by the number of islands and the (expected) number of emigrants sent by one island.

In order to achieve a good balance between the communication effort and the parallel running time, we consider the following migration policies. The first two policies make sense for both island models. The last two policies are tailored towards the heterogeneous model.

complete migration: each island sends migrants to all other islands.
uniform probabilistic: each island sends migrants to every other island independently with a migration probability p.
non-uniform probabilistic: each island i sends migrants to every other island $(i+k) \bmod (m+1)$ independently with probability $1/k$.
smart migration: Each island i sends migrants to island $c(\tilde{X}_i)$, where $\tilde{X}_i$ is the offspring generated on island i.

3 Analysis of the Homogeneous Island Model

We first consider the homogeneous model with uniform probabilistic migration as this includes complete migration. In their analysis of SEMO, Friedrich *et al.* [14] consider the time until SEMO finds an empty selection, and how long it takes to get a H_m-approximate solution from there. Their results are as follows.

Theorem 1 (Friedrich *et al.* [14]). *For any initialisation and every* Set-Cover *instance, SEMO and global SEMO find an H_m-approximate solution in $O(m^2 n + mn \log(nc_{\max}))$ expected generations. When starting with a population containing only an empty selection, the time bound is $O(m^2 n)$ generations.*

The following lemma is at the heart of their–and our–analysis. It goes back to Chvatal's analysis of the greedy algorithm [18]. Starting with an empty set, the greedy algorithm subsequently adds the most cost-effective set to the current solution. When k elements are covered, for some $0 \leq k \leq m$, the cost of this partial solution is at most $\text{cost}(X) \leq (H_m - H_{m-k})\,\text{OPT}$, where OPT denotes the cost of an optimal solution. For $k = m$ this gives an H_m-approximation.

Lemma 1. *Let* OPT *be the cost of an optimal set cover and X be such that $c(X) = k$ (with $k < m$) and $\text{cost}(X) \leq (H_m - H_{m-k})\,\text{OPT}$. Adding the most cost-effective set to X creates X' with $c(X') = k'$ and $\text{cost}(X') \leq (H_m - H_{m-k'})\,\text{OPT}$.*

Proof. The selection X leaves $m-k$ elements of S uncovered. These elements can be covered at cost OPT since the optimal cover covers the whole set. Then there is a set with cost-effectiveness at least $\frac{m-k}{\text{OPT}}$. Let i be the number of newly covered elements by adding this set, then after adding the set we get a solution covering $k' = k + i$ elements at cost no more than

$$\left(H_m - H_{m-k} + \frac{i}{m-k}\right) \cdot \text{OPT} \leq (H_m - H_{m-k'}) \cdot \text{OPT}\,. \qquad \square$$

This behaviour can be mimicked by SEMO [14] and the homogeneous island model. Friedrich *et al.* [14] define the *potential* of the population of the archipelago as the largest k such that there is an individual in the population that covers k elements and costs at most $(H_m - H_{m-k}) \cdot \mathrm{OPT}$. The potential can never decrease as SEMO always keeps some solution with k covered elements in the population. Starting with empty selections, the initial potential is at least 0.

The probability of increasing the potential is at least $1/((m+1)en)$ for the following reasons. It is sufficient to select the solution defining the potential and to add a set with maximum cost-effectiveness (Lemma 1). The population contains at most $m+1$ individuals, so the probability of selecting the right parent is at least $1/(m+1)$. The probability of a specific 1-bit mutation is at least $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$ for both local and global SEMO.

This analysis can be transferred to our homogeneous island model using the general method by Lässig and Sudholt [10] based on fitness levels. Assume the search space can be partitioned into fitness-level sets ordered w. r. t. fitness such that an EA never decreases its current level. If we have lower bounds on the probability that the EA will leave a current level towards a better fitness-level set, we get an upper bound on the expected hitting time of the final level. For island models we get upper bounds on the expected parallel running time that depend on the topology at hand and the probability that migration successfully transmits information about the current best fitness level. A rapid spread of information enables more islands to search on the current best fitness level, which gives better performance guarantees than a slow spread of information.

In [10] upper bounds are stated for common topologies: ring graphs, torus or grid graphs, and the complete topology. In our case instead of using fitness levels, we argue with the potential of islands. As seen above, the potential can never decrease. We have $m+1$ potential values, and the probability of increasing the potential on any island is at least $1/((m+1)en)$. Plugging this into the results from [10,19], we get the following bounds on the expected parallel time. The expected communication effort is by a factor of $pd(m+1)\mu$ larger than the expected parallel time, where d is the degree of any node in the topology.

Theorem 2. *For the homogeneous island model based on (global) SEMO on μ islands and migration probability $p > 0$ the expected parallel time until an H_m-approximation for* SetCover *is found is bounded by*

- $O\left(\frac{n^{1/2}m^{3/2}}{p^{1/2}} + \frac{nm^2}{\mu}\right)$ *for any ring topology,*
- $O\left(\frac{n^{1/3}m^{4/3}}{p^{2/3}} + \frac{nm^2}{\mu}\right)$ *for any undirected $\sqrt{\mu} \times \sqrt{\mu}$ grid or torus graph*
- $O\left(\frac{m}{p} + \frac{nm^2}{\mu}\right)$ *for the complete topology K_μ.*

The expected communication effort is $O\left(p^{1/2}\mu n^{1/2}m^{5/2} + pnm^3\right)$ for rings, $O\left(p^{1/3}\mu n^{1/3}m^{7/3} + pnm^3\right)$ for grids and $O\left(\mu^2 m^2 + p\mu nm^3\right)$ for K_μ.

The upper bounds are asymptotically minimised for choosing the number of islands as $\mu = \sqrt{pnm}$, $\mu = (pnm)^{2/3}$, and $\mu = pnm$, respectively. With these choices we get expected parallel times of $O(n^{1/2}m^{3/2}/p^{1/2})$, $O(n^{1/3}m^{4/3}/p^{2/3})$,

and $O(m/p)$, respectively (see Table 1 in Section 5). The expected communication effort is $O(pnm^3)$, $O(pnm^3)$, and $O(p^2n^2m^4)$, respectively. Multiplying all parallel times by μ, we see that the expected sequential time is bounded by $O(nm^2)$ in all three cases. This asymptotically matches the upper bound from Theorem 1 for initialisation with empty selections. This means that, apart from constant factors hidden in the asymptotic notation, in these cases parallelization does not increase the (upper bounds on the) total running time, but the (upper bounds on the) parallel time can decrease significantly. In fact, all numbers of islands up to the values mentioned above yield linear speedups—for cases where the $O(nm^2)$-bound for a single (global) SEMO is asymptotically tight.

As remarked in [11], the bound for the complete topology with $p = 1$ also applies to an offspring population-version of SEMO where λ offspring are created and added to the population, before removing dominated solutions.

4 Analysis of the Heterogeneous Island Model

For the heterogeneous model based on (1+1) EA or RLS we first present an analysis for the complete migration policy.

Theorem 3. *The heterogeneous island model with complete migration finds an H_m-approximate solution for* SetCover *in an expected parallel time of $O(n \cdot \min(m, n))$. The expected communication effort is $O(nm^2 \cdot \min(m, n))$.*

Proof. As in Theorem 2 we calculate the expected time to produce a solution that is at least as good as the greedy solution, starting from 0^n and always adding the most cost-effective set. We define again the potential of the population of the archipelago as the largest k such that there is an individual in the population that covers k elements and costs at most $(H_m - H_{m-k}) \cdot \mathrm{OPT}$. At the end of each generation (after migration and selection) the potential can't decrease. In fact the individual X^k on island k can only be replaced by an individual with the same number of covered elements but a lower cost (and that would not affect the potential). Instead the potential can be increased to k' mutating X^k such that the most cost-effective set is added. That would produce an individual $\tilde{X}^k$ such that $c(\tilde{X}^k) = k' > k$ and $\mathrm{cost}(\tilde{X}^k) \leq (H_k - H_{m-k'}) \cdot \mathrm{OPT}$ (Lemma 1).

After migration and selection this individual will replace the individual on the island k' (which had higher cost and therefore lower fitness). This specific 1-bit mutation happens with probability at least $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$ for both local and global mutation. At most n sets can be included in a selection but, if $n > m$, at most m of them will be selected since each most cost-effective set covers at least one new element (otherwise its cost-effectiveness would be 0). So after $O(n \cdot \min(m, n))$ expected generations $k = m$ and then on island m we get an $H_m - H_{m-m} = H_m$-approximate solution. □

Comparing this time with [14] and assuming $n = O(m)$, we get that our parallel time is by a factor of $\Theta(m)$ lower, while we get the same upper bound for the sequential running time.

For uniform probabilistic migration with migration probability $p < 1$, the island model only increases the potential if migration happens on the edge that links the two islands involved (k and k'). The probability estimate for this event decreases by a factor of p, and the waiting time thus increases by $1/p$.

Theorem 4. *The heterogeneous island model with uniform probabilistic migration and migration probability p finds an H_m-approximate solution for* SetCover *in an expected parallel time of $O(n \cdot \min(m,n)/p)$. The expected communication effort is $O(nm^2 \cdot \min(m,n))$.*

We see that our estimate of the communication effort has not improved. This is not surprising as we only rely on inter-island evolution for making progress. A uniform migration probability delays the inter-island evolution and the reduced communication effort in a single generation is nullified by a larger parallel running time.

With non-uniform probabilistic migration, the chance of making the right migration is generally higher than for uniform migration probabilities. Typically only few new elements are covered, when adding a most cost-effective set. A large number of new elements implies that we make large progress. This balances out a small migration probability: if adding the most cost-effective set covers j new elements, the probability of making this move is at least $1/j \cdot 1/(en)$. In expectation, the potential increases by at least $j \cdot 1/j \cdot 1/(en) = 1/(en)$, regardless of j. A straightforward drift analysis gives the following.

Theorem 5. *The heterogeneous island model with non-uniform probabilistic migration finds an H_m-approximate solution for* SetCover *in an expected parallel time of enm. The expected communication effort is at most $enm^2 H_m$.*

Smart migration sends emigrants only to the unique island where they are considered feasible. The proof of Theorem 3 only relies on such migrations. Hence the upper bound also holds for smart migration.

Theorem 6. *The heterogeneous island model with smart migration finds an H_m-approximate solution for* SetCover *in an expected parallel time of $O(n \cdot \min(m,n))$. The expected communication effort is $O(nm \cdot \min(m,n))$.*

In our setting, smart migration outperforms all other migration policies as it leads to the best upper bound for the communication effort.

5 Discussion and Conclusions

We have proposed and analysed two parallel EAs for the SetCover problem that provably find good approximations. Table 1 gives an overview of our results, regarding parallel and sequential expected running times as well as the communication effort. In order to fairly compare heterogeneous and homogeneous models we consider them running on μ processors. For the heterogeneous model this means (for $\mu \leq m$) running up to $\lceil \frac{m+1}{\mu} \rceil$ islands on the same processor and thus increasing the parallel running time by a factor of $\Theta(\frac{m}{\mu})$.

Table 1. Upper bounds on expected parallel times (general bounds and bounds for best μ), expected sequential times and expected communication effort for homogeneous island models with various migration topologies and for heterogeneous island models with various migration policies, until an H_m-approximation is found for any SetCover instance with m elements and n sets. p denotes the migration probability. We simplified $\min(n, m) \leq m$ and we constrained μ to yield linear speedups.

Algorithm	parallel time bounds general b. $\rightsquigarrow$ best bound		seq. time	comm. effort
Non-parallel SEMO	$O(nm^2)$	$\rightsquigarrow O(nm^2)$	$O(nm^2)$	0
Homogeneous island model based on (global) SEMO and topology...				
– ring ($\mu \leq \sqrt{pnm}$)	$O\left(\frac{nm^2}{\mu}\right)$	$\rightsquigarrow O\left(\frac{n^{1/2}m^{3/2}}{p^{1/2}}\right)$	$O(nm^2)$	$O(pnm^3)$
– grid ($\mu \leq (pnm)^{2/3}$)	$O\left(\frac{nm^2}{\mu}\right)$	$\rightsquigarrow O\left(\frac{n^{1/3}m^{4/3}}{p^{2/3}}\right)$	$O(nm^2)$	$O(pnm^3)$
– complete ($\mu \leq pnm$)	$O\left(\frac{nm^2}{\mu}\right)$	$\rightsquigarrow O\left(\frac{m}{p}\right)$	$O(nm^2)$	$O(p^2n^2m^4)$
Heterogeneous island model with $\mu \leq m$ based on (1+1) EA (or RLS) and policy...				
– complete	$O\left(\frac{nm^2}{\mu}\right)$	$\rightsquigarrow O(nm)$	$O(nm^2)$	$O(nm^3)$
– uniform prob.	$O\left(\frac{nm^2}{\mu p}\right)$	$\rightsquigarrow O\left(\frac{nm}{p}\right)$	$O\left(\frac{nm^2}{p}\right)$	$O(nm^3)$
– non-uniform prob.	$O\left(\frac{nm^2}{\mu}\right)$	$\rightsquigarrow O(nm)$	$O(nm^2)$	$O(nm^2 \log m)$
– smart migration	$O\left(\frac{nm^2}{\mu}\right)$	$\rightsquigarrow O(nm)$	$O(nm^2)$	$O(nm^2)$

For the homogeneous model based on (global) SEMO, the topology determines how many islands still give a linear speedup. For dense topologies more islands can be used. The migration probability gives a smooth trade-off between this maximum number of islands and the communication effort. For large migration probabilities the heterogeneous island model based on the (1+1) EA (or RLS) has lower communication costs, when comparing complete topologies or using the right migration policies. It is also easier to implement as unlike for the SEMO-based model it is not necessary for each island to handle large populations and to remove many dominated solutions. Thus the heterogeneous model is also faster when considering the time and space required to compute a generation.

The discussion on migration policies has revealed how adding more knowledge about the problem can decrease the communication effort. The complete migration and uniform migration policies do not require any knowledge about the problem at hand, while non-uniform migration only needs a sensible ordering of islands to work. This ordering should be consistent with the similarity between different islands. We believe that this approach can be fruitful for other heterogeneous island models. Smart migration requires knowledge about the problem at hand since it needs to inspect the genotype to determine the island to send it to. But it leads to the best performance guarantees among all considered policies.

Experiments (not included here) show that on random SetCover instances both island models quickly find better solutions than the greedy algorithm. An experimental study is left for future work. Future work should also investigate whether the approach used in the heterogeneous island model (i. e. assigning a portion of the search space to each island) can solve a broader class of problems.

Acknowledgments. This research was partially supported by EPSRC grants EP/D052785/1 and EP/I010297/1.

References

1. Luque, G., Alba, E.: Parallel Genetic Algorithms–Theory and Real World Applications. Springer (2011)
2. Nedjah, N., de Macedo Mourelle, L., Alba, E.: Parallel Evolutionary Computations. Springer (2006)
3. Tomassini, M.: Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time. Springer (2005)
4. Skolicki, Z., De Jong, K.: The influence of migration sizes and intervals on island models. In: Proc. of GECCO 2005, pp. 1295–1302. ACM (2005)
5. Rudolph, G.: Takeover time in parallel populations with migration. In: Filipic, B., Silc, J. (eds.) Proc. of BIOMA 2006, pp. 63–72 (2006)
6. Lässig, J., Sudholt, D.: The benefit of migration in parallel evolutionary algorithms. In: Proc. of GECCO 2010, pp. 1105–1112. ACM (2010)
7. Lässig, J., Sudholt, D.: Experimental Supplements to the Theoretical Analysis of Migration in the Island Model. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 224–233. Springer, Heidelberg (2010)
8. Watson, R.A., Jansen, T.: A building-block royal road where crossover is provably essential. In: Proc. of GECCO 2007, pp. 1452–1459. ACM (2007)
9. Neumann, F., Oliveto, P.S., Rudolph, G., Sudholt, D.: On the effectiveness of crossover for migration in parallel evolutionary algorithms. In: Proc. of GECCO 2011, pp. 1587–1594. ACM (2011)
10. Lässig, J., Sudholt, D.: General Scheme for Analyzing Running Times of Parallel Evolutionary Algorithms. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 234–243. Springer, Heidelberg (2010)
11. Lässig, J., Sudholt, D.: Adaptive population models for offspring populations and parallel evolutionary algorithms. In: Proc. of FOGA 2011, pp. 181–192. ACM (2011)
12. Lässig, J., Sudholt, D.: Analysis of Speedups in Parallel Evolutionary Algorithms for Combinatorial Optimization. In: Asano, T., Nakano, S.-i., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 405–414. Springer, Heidelberg (2011)
13. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology, pp. 457–474. Kluwer (2003)
14. Friedrich, T., He, J., Hebbinghaus, N., Neumann, F., Witt, C.: Approximating covering problems by randomized search heuristics using multi-objective models. Evolutionary Computation 18(4), 617–633 (2010)
15. Doerr, B., Eremeev, A.V., Neumann, F., Theile, M., Thyssen, C.: Evolutionary algorithms and dynamic programming. Theoretical Computer Science 412(43), 6020–6035 (2011)
16. Skolicki, Z.: An Analysis of Island Models in Evolutionary Computation. PhD thesis, George Mason University, Fairfax, VA (2000)
17. Alba, E.: Parallel evolutionary algorithms can achieve super-linear performance. Information Processing Letters 82(1), 7–13 (2002)
18. Chvatal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research 4(3), 233–235 (1979)
19. Lässig, J., Sudholt, D.: General upper bounds on the running time of parallel evolutionary algorithms. ArXiv e-prints (2012) Extended version of [10], http://arxiv.org/abs/1206.3522.

Geometric Semantic Genetic Programming

Alberto Moraglio[1], Krzysztof Krawiec[2], and Colin G. Johnson[3]

[1] School of Computer Science, University of Birmingham, UK
A.Moraglio@cs.bham.ac.uk
[2] Institute of Computing Science, Poznan University of Technology, Poland
kkrawiec@cs.put.poznan.pl
[3] School of Computing, University of Kent, UK
C.G.Johnson@kent.ac.uk

Abstract. Traditional Genetic Programming (GP) searches the space of functions/programs by using search operators that manipulate their syntactic representation, regardless of their actual semantics/behaviour. Recently, semantically aware search operators have been shown to outperform purely syntactic operators. In this work, using a formal geometric view on search operators and representations, we bring the semantic approach to its extreme consequences and introduce a novel form of GP – Geometric Semantic GP (GSGP) – that searches *directly* the space of the underlying semantics of the programs. This perspective provides new insights on the relation between program syntax and semantics, search operators and fitness landscape, and allows for principled formal design of semantic search operators for different classes of problems. We derive specific forms of GSGP for a number of classic GP domains and experimentally demonstrate their superiority to conventional operators.

1 Introduction

Traditional genetic programming ignores the *meaning* of programs, as the search operators it employs act on their syntactic representations, regardless of their semantics. E.g., subtree swap crossover is used to recombine functions represented as parse trees, regardless of trees representing boolean expressions, mathematical functions, or computer programs. Whereas this guarantees producing syntactically well-formed expressions, why should such a *blind* syntactic search work well for different problems and across domains? In the end, it is the meaning of programs that determines how successful search is at solving the problem.

The semantics of a program can be formally defined in a number of ways. It can be a canonical representation, so that any two programs with the same semantics/behaviour have the same canonical representation (e.g., Binary Decision Diagrams (BDD) for boolean expressions). It can be a description of the behaviour of the program using a logical formalism. This is used in formal methods to reason formally about programs. From a strict search viewpoint, it may be argued that the semantics of a program is just its fitness. Finally, it can also be defined as the mathematical function computed by a program, i.e., the set of input-output pairs making up the computed function.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 21–31, 2012.

In the literature, there are a number of works using the semantics of programs to improve GP. As many individuals encode the same function, some researchers use canonical representations of functions to enforce semantic diversity throughout evolution, by creating semantically unique individuals in the initial population [2,4], and by discarding offspring of crossover and mutation when semantically coinciding with their parents [3,1]. Uy et al. [11] propose a measure of semantic distance between individuals based on how their outputs differ for the same set of inputs sampled at random. This distance is then used to bias semantically the search operators: mutation rejects offspring that are not sufficiently semantically similar to the parent; crossover chooses only semantically similar subtrees to swap between parents. Also Krawiec et al. [5,6] have used a notion of semantic distance to propose a crossover operator for GP trees that is approximately a geometric crossover [10,8] in the semantic space (see Section 2). Interestingly, the fitness landscape induced by this operator has perfect fitness-distance correlation. The operator was implemented approximately by using a traditional crossover, generating a large number of offspring, and accepting only those offspring that were "semantically intermediate" with respect to the parents.

Whereas, overall the semantically aware methods above produced superior performance to traditional methods, they are *indirect*: search operators are implemented via acting on the syntax of the parents to produce offspring, which are accepted only if some semantic criterion is satisfied. This has two drawbacks: (i) these implementations are very wasteful as heavily based on trial-and-error; (ii) they do not provide insights on how syntactic and semantic searches relate to each other. Would it then be possible to search *directly* the semantic space of programs? More precisely, would it be possible to build search operators that, acting on the syntax of the parent programs, produce offspring that are *guaranteed* to respect some semantic criterion/specification by construction? Krawiec et al. [5,6] stated that due to the complexity of the genotype-phenotype mapping in GP, a direct implementation of exact semantic operators is probably impossible.

The present paper brings the following contributions: (i) it formalises the notions of semantic distance, semantic geometric operators and semantic fitness landscapes; (ii) it proves that the fitness landscapes seen by geometric semantic operators are always cone landscapes, which are easy to search; (iii) it shows that, contrary to widespread belief, the genotype-phenotype map of commonly considered GP domains is, in an important sense, very easy, not complex; (iv) it introduces a general method to derive *exact* semantic geometric crossovers and mutations for different problem domains that search *directly* the semantic space; (v) it derives semantic operators for the Boolean domain, arithmetic domain, and program domain; (vi) it reports experimental results for a standard test-bed of GP problems.

2 Abstract Geometric Semantic Search

In this section, we report non-operational definitions of geometric semantic operators and their properties. They are characterised algorithmically in Section 3.

A search operator $CX : S \times S \to S$ is a *geometric crossover* w. r. t. the metric d if for any choice of parents p_1 and p_2, any of their offspring $o = CX(p_1, p_2)$

is in the metric segment between parents. A search operator $M : S \to S$ is a *geometric ϵ-mutation* w. r. t. the metric d if for any choice of the parent p, any of its offspring $o = M(p)$ is in the metric ball of radius ϵ centered in the parent. Given a fitness function $f : S \to \mathbb{R}$, the geometric search operators induce or see the fitness landscape (f, S, d). Many well-known recombination operators across representations are geometric crossovers [8], e. g., all mask-based crossovers on binary strings are geometric crossovers w. r. t. Hamming distance. Point mutation on binary strings is geometric 1-mutation w. r. t. Hamming distance. Geometric operators can also be derived for new spaces and representations by using in their definitions a distance based on a target representation (e.g., edit distance). If the distance is not directly linked to a representation, the geometric operators are well-defined but an algorithmic description for them can be hard to derive.

Genetic programming is essentially a supervised learning method: given a fixed set of input-output pairs $T = \{(x_1, y_1), ..., (x_N, y_N)\}$ (i.e., training set or fitness cases), a function $h : X \to Y$ belonging to a certain fixed class H – specified by the chosen terminal and function sets – is sought (evolved) that interpolates the known input-output pairs, i.e., $\forall (x_i, y_i) \in T : h(x_i) = y_i$. The fitness function $F_T : \mathrm{H} \to \mathbb{R}$ measures the error of a function h on the training set T. Compared to other learning methods, two distinctive features of GP are that (i) it can be applied to learn virtually any type of functions, and (ii) it is a black-box method, as it does not need explicit knowledge of the training set, but only of the errors on the training set.

Let $I = (x_1, ..., x_N)$ and $O = (y_1, ..., y_N)$ be the input and the output vectors, respectively, associated with the training set T. Let $O(h)$ be the vector of the outputs of a function h when queried with the inputs I, i.e., $O(h) = (h(x_1), ..., h(x_N))$. The function $O : \mathrm{H} \to Y^N$ can be interpreted as *genotype-phenotype mapping* as it maps a representation of a function h (i.e., genotype) to the actual outcome of the application of function h on the input vector I (i.e., phenotype) represented by its output vector.

Traditional measures of error of a function h on the training set T can be *interpreted as distance* between the target output vector O and the output vector $O(h)$ measured using some suitable metric D, i.e., $F_T(h) = D(O, O(h))$ (to minimise). For example, when the space H of functions considered is the class of Boolean functions, the input and output spaces are $X = \{0, 1\}^n$ and $Y = \{0, 1\}$, and the output vector is a binary vector of size N (i.e., Y^N). A suitable metric D to measure the error as a distance between binary vectors is the Hamming distance. For functions returning real values (e.g., in regression applications), the output vectors are real vectors. In this case, suitable metrics to measure the error are Euclidean and Manhattan distances, each of which gives rise to a different type of fitness function.

We define *semantic distance* SD between two functions $h_1, h_2 \in \mathrm{H}$ as the distance between their corresponding output vectors w. r .t. the input vector of all possible inputs (i.e., $I = (x_i)$ for all $x_i \in X$) measured with the metric D used in the definition of the fitness function F_T, i.e., $SD(h_1, h_2) = D(O(h_1), O(h_2))$. The semantic distance SD is a genotypic distance induced from a phenotypic metric

D, via the genotype-phenotype mapping O. As O is generally non-injective (i.e., different genotypes may have the same phenotype), SD is only a pseudometric (i.e., distinct functions can have distance zero). This naturally induces an equivalence relation on genotypes: genotypes belong to the same semantic class $\overline{h}$ iff their semantic distance is zero. Then, SD can be interpreted as a metric on the set of semantic classes of genotypes $\overline{H}$.

We define *semantic geometric operators* as geometric crossover and mutation specified on the space of (classes of) functions endowed with the distance SD. E.g., semantic geometric crossover on boolean functions returns offspring boolean functions such that the output vectors of the offspring are in the Hamming segment between the output vectors of the parents (w. r .t. all $x_i \in X$). The effect of SD being defined on the space of classes of functions $\overline{H}$, rather than on the space of functions H, is that the geometric crossover is only a function of the semantic classes of the parents $\overline{h_1}$, $\overline{h_2}$ rather than directly of the parents h_1, h_2 (i.e, their specific representations), and the returned offspring can be any function h_3 belonging to the offspring class $\overline{h_3}$ (i.e., any function with the prescribed output vector/semantics).

The *semantic fitness landscape* seen by an evolutionary algorithm with semantic geometric operators has a nice shape by construction: from the definition of semantic distance, *the fitness of a solution is its distance in the search space to the optimum* (cone landscape).[1] This observation is *remarkably general*, as it holds for any domain of application of GP (e.g., Boolean, Arithmetic, Program), any specific problem within a domain (e.g., Parity and Multiplexer problems in the Boolean domain) and for any choice of metric for the error function. Furthermore, there is some formal evidence [9] that EAs with geometric operators can optimise cone landscapes efficiently very generally for virtually any metric.

GP search with geometric operators w. r .t. the semantic distance SD on the space of function classes $\overline{H}$ is formally equivalent to EA search with geometric operators w. r .t. the distance D on the space of output vectors. This is because: (i) semantic classes of functions are in bijective correspondence with output vectors, as "functions with the same output vector" is the defining property of a semantic class of function; (ii) semantic geometric operators on functions are isomorphic to geometric operators on output vectors, as SD is induced from D via the genotype-phenotype mapping (see diagram (1)).[2] E.g., for Boolean functions, semantic GP search is equivalent to GA search on binary strings on OneMax of dimension N.

[1] The landscape includes also a form of neutrality. As the training set covers a fraction of all possible input-output pairs of a function, only that part of the output vector of a function affects its fitness, the remaining large part is "inactive". This does not affect crossover, but it may make mutation ineffective.

[2] Despite this formal equivalence, actually encoding a function in a EA using its output vector instead of, say, a parse tree, is futile: in the end we want to find a function represented in an *intensive form* that can represent concisely "interesting" functions and that allows for meaningful generalisation of the training set.

3 Construction of Geometric Semantic Operators

The commutative diagram below illustrates the relationship between the semantic geometric crossover GX_{SD} on genotypes (e.g., trees) on the top, and the geometric crossover (GX_D) operating on the phenotypes (i.e., output vectors) induced by the genotype-phenotype mapping O, at the bottom. It holds that for any $T1, T2$ and $T3 = GX_{SD}(T1, T2)$ then $O(T3) = GX_D(O(T1), O(T2))$.

$$\begin{array}{ccc} T1 \times T2 & \xrightarrow{GX_{SD}} & T3 \\ \downarrow O \quad \downarrow O & & \downarrow O \\ O1 \times O2 & \xrightarrow{GX_D} & O3 \end{array} \tag{1}$$

The problem of finding an algorithmic characterization of semantic geometric crossover can be stated as follows: given a family of functions H, find a recombination operator GX_{SD} (unknown) acting on elements of H that induces via the genotype phenotype mapping O a geometric crossover GX_D (known) on output vectors. E.g., for the case of boolean functions with fitness measure based on Hamming distance, output vectors are binary strings and GX_D is a mask-based crossover. We want to derive a recombination operator acting on Boolean functions that corresponds to a mask-based crossover on their output vectors. Note that there is a different type of semantic geometric crossover for each choice of space H and distance D. Consequently, there are different semantic crossovers for different GP domains. We will give a recipe to derive specific semantic crossovers for new domains.

Definition 1. *Given two parent functions $T1, T2 : \{0,1\}^n \to \{0,1\}$, the recombination SGXB returns the offspring boolean function $T3 = (T1 \wedge TR) \vee (\overline{TR} \wedge T2)$ where TR is a randomly generated boolean function (see Fig. 1).*

Theorem 1. *SGXB is a semantic geometric crossover for the space of boolean functions with fitness function based on Hamming distance, for any training set and any boolean problem.*

Proof. The offspring function is $T3 = (T1 \wedge TR) \vee (\overline{TR} \wedge T2)$. Expanding it for any input i: $T3(i) = (T1(i) \wedge TR(i)) \vee (\overline{TR(i)} \wedge T2(i))$. So, for any entry i of the output vectors: $O(T3)(i) = (O(T1)(i) \wedge O(TR)(i)) \vee (\overline{O(TR)(i)} \wedge O(T2)(i))$. In the last expression, the Boolean expression at each position i is a multiplexer function which, depending on the bit-value of $O(TR)(i)$ (piloting bit), assigns either $O(T1)(i)$ or $O(T2)(i)$ to $O(T3)(i)$. Then, the output vector $O(TR)$ acts as a crossover mask on the parent output vectors $O(T1)$ and $O(T2)$ to produce the offspring output vector $O(T3)$. This is a geometric crossover on output vectors w. r. t. the Hamming distance.

Let us now consider the Real Functions domain (e.g., for symbolic regression).

Definition 2. *Given two parent functions $T1, T2 : \mathbb{R}^n \to \mathbb{R}$, the recombinations SGXE and SGXM return the real function $T3 = (T1 \cdot TR) + ((1 - TR) \cdot T2)$ where TR is a random real constant in $[0, 1]$ (SGXE), or a random real function with codomain $[0, 1]$ (SGMX).*

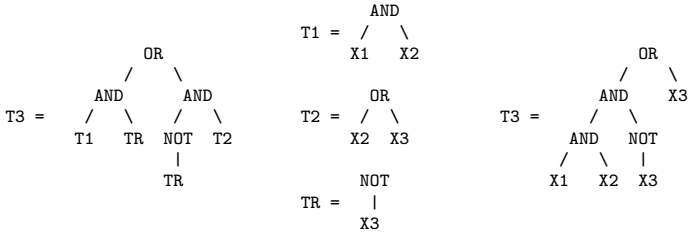


Fig. 1. Left: Semantic Crossover scheme for Boolean Functions; Centre: Example of parents (T1 and T2) and random mask (TR); Right: Offspring (T3) obtained by substituting T1, T2 and TR in the crossover scheme and simplifying

Theorem 2. *SGXE and SGXM are semantic geometric crossovers for the space of real functions with fitness function based on Euclidean and Manhattan distances, respectively, for any training set and any real problem.*

Proof. By expanding the offspring function on the inputs and considering every entry i of the output vectors: $O(T3)(i) = (O(T1)(i) \cdot O(TR)(i)) + ((1 - O(TR)(i)) \cdot O(T2)(i))$. As $O(TR)(i) \in [0, 1]$, at each position the value of $O(T3)(i)$ is a convex combination of the values of $O(T1)(i)$ and $O(T2)(i)$. So, the vector $O(T3)$ is within the hyper-box delimited by $O(T1)$ and $O(T2)$, i.e., it is in their Manhattan segment. Expressing the above relation in functional form: $O(T3) = (O(T1) \cdot O(TR)) + ((1 - O(TR)) \cdot O(T2))$. When additionally $O(TR)$ is constant in i, we see that $O(T3)$ is a convex combination of the vectors $O(T1)$ and $O(T2)$, i.e., it is in their Euclidean segment.

Let us now consider the Computer Program domain intended as functions with symbols as inputs (IS) and outputs (OS). The following can be easily extended to other types of inputs and outputs.

Definition 3. *Given two parent programs* `T1, T2` $: IS^n \rightarrow OS$*, the recombination SGXP returns the offspring program* `T3 = IF CONDR THEN T1 ELSE T2` *where* `CONDR` *is a random program whose output is interpreted as a logical value.*

Theorem 3. *SGXP is a semantic geometric crossover for the space of programs with fitness function based on Hamming distance, for any training set and any problem.*

Proof. By expanding the offspring program on the inputs and considering every entry i of the output vectors: `O(T3)(i) = IF O(CONDR)(i) THEN O(T1)(i) ELSE O(T2)(i)`. This means that for each input, the output value of `T3` is that of `T1` or `T2` depending of the value of `CONDR`, which is then acting as a crossover mask on `T1` and `T2`. This is a geometric crossover on the output vectors w. r. t. the Hamming distance (for symbolic vectors).

Definition 4. ***Semantic Mutations. Boolean:*** *Given a parent function* $T : \{0, 1\}^n \rightarrow \{0, 1\}$*, the mutation SGMB returns the offspring boolean function*

$TM = T \vee M$ with probability 0.5 *and* $TM = T \wedge \overline{M}$ *with probability* 0.5 *where* M *is a random minterm of all input variables.* ***Arithmetic:*** *Given a parent function* $T : \mathbb{R}^n \to \mathbb{R}$*, the mutation SGMR with mutation step* ms *returns the real function* $TM = T + ms \cdot (TR1 - TR2)$ *where* $TR1$ *and* $TR2$ *are random real functions.* ***Programs:*** *Given a parent program* `T`*, the mutation SGMP returns the offspring program* `TM = IF CONDR THEN OUTR ELSE T` *where* `CONDR` *is a condition which is true only for a single random setting of all input parameters, and* `OUTR` *is a random output symbol. The offspring can be expressed as nested* `IF-THEN-ELSE` *statements with simple conditions of a single input parameter each.*

Theorem 4. *SGMB and SGMP are semantic* 1*-geometric mutations for boolean functions and of programs, respectively, with fitness function based on Hamming distance. SGMR is a semantic* ϵ*-geometric mutation for real functions with fitness function based on Euclidean and Manhattan distances. The mean of its probability distribution is the parent, and* ϵ *is proportional to the step* ms*.*

General Construction Method: It can be obtained by reversing the common argument in the proofs above: (i) take the geometric crossover on output vectors associated with the distance used in the fitness function; (ii) consider the action of the recombination operator on a single entry of the output vectors; (iii) use the domain-specific language of the particular class of functions considered to describe the recombination action on a single entry; (iv) that *description* is the scheme to produce the offspring. Note that the offspring is not only the *effect* of crossover, it is also the *description* of how to crossover its parents. The target domain-specific language must be expressive enough to describe the recombination. This seems to be the case for most GP problems.

Simplification: As the syntax of the offspring of semantic crossover contains both parents, the size of individuals *grows exponentially* with the number of generations. To keep their size manageable, we need to simplify offspring sufficiently and efficiently (not optimally, as that is NP-Hard on many domains) *without changing the computed function.* The search of semantic crossover is completely unaffected by syntactic simplification, which can be done at any moment and to any extent. For boolean functions, there are quick function-preserving simplifiers (e.g., Espresso). Computer algebra systems (e.g., Maple) can be used to simplify symbolically mathematical functions, like polynomials, and more complicated expressions including *sin*, *cos*, *exp*, etc. if used in disciplined ways (e.g., nested *sin* not allowed). Formal methods (e.g., static analysis) can be used to simplify computer programs (but loops/recursion may be a challenge).

Does Syntax Matter? *In theory*, it does not matter! The offspring is a function obtained from a functional combination of parent functions. The offspring is defined purely functionally and does not depend on how functions are actually represented (e.g., trees, graphs, sequences) and what language is used (e.g., Java, Lisp, Prolog), as long as the semantic operators can be described in that language. *In practice*, syntax does matter! As genotype structure and language influence the way random genotypes are generated, as different representations suggest different "natural" ways of generating them. This affects the

offspring distribution of semantic operators, the semantic diversity in the initial population, and the dependencies in the crossover mask. Furthermore, some representations may be easier to simplify, and may have preferable inductive bias (i.e., generalise better on unseen inputs).

4 Computational Experiments

We compare GP, semantic GP (SGP), and semantic stochastic hill climber (SSHC), which employs semantic mutation to explore the neighbourhood. In all experiments GP and SGP use a generational scheme with tournament selection (size 5), crossover and mutation, which are always engaged. We give the algorithms the same number of evaluations, set as the number needed by SSHC to typically find the optimum (as SSHC is the quickest). We also compare algorithms on CPU time: GPt is GP running for the same time as the greater of average execution times of SGP and SSC. Below are the main settings of the experimental setups considered. Other parameters are set to ECJ's defaults [7].

Boolean Functions. (Table 1): *Test-bed*: standard GP benchmark. *Fitness function*: Hamming distance to the output vector of the target function queried on all inputs. *GP*: standard GP with instruction set: 'And', 'Or', 'Not'. *SGP and SSHC*: individuals are Boolean expressions in disjunctive normal form; SGMB and SGXB with a mask TR being a random minterm of a random subset of input variables; simplification of offspring by Espresso. *Comparison*: budget of $2n \cdot 2^n$ evaluations, where n is the number of input variables; as to population size, GP and SGP have $\max\{\sqrt{2^n}, 10\}$, and GPt has $\max\{\sqrt{2^n}, 50\}$ (and from 20 to 200 times more evaluations).

Polynomial Regression. (Table 2): *Test-bed*: univariate polynomials of degrees from 3 to 10, with real-valued coefficients uniformly drawn from $[-1, 1]$. *Fitness function*: Euclidean distance to the output vector of the target function queried on 20 inputs in $[-1, 1]$. *GP*: Standard GP with instruction set: '+', '-', '*', 'x', constant. *SGP and SSHC*: individuals are polynomials of degree 10, initialised with coefficients drawn uniformly from $[-1, 1]$; SGXE and SGMR with step $ms = 0.001$; implicit simplification (i.e, weighted sums of polynomials). *Comparison*: budget of 100,000 evaluations, with population size 1,000 for GP, and 20 for SGP.

Classifiers. (Table 3): *Test-bed*: $IS = \{1, ..., n_c\}$, $OS = \{1, ..., n_{cl}\}$, target functions $f : IS^{n_v} \rightarrow OS$ are $f(x_1, x_2, ..., x_{n_v}) = ((x_1 + x_2) \mod n_{cl}) + 1$, for all combinations of $n_v = 3, 4$, $n_c = 3, 4$ and $n_{cl} = 2, 4, 8$. *Fitness function*: Hamming distance to the output vector of the target function queried on all inputs. All algorithms use classifiers of the form: `<CF> := IF <COND> THEN <CF> ELSE <CF> || <OS>; <COND> := <`x_i`> = <IS>`, and Ramped-half-and-half initialisation. *SGP and SSHC* use SGXP and SGMP, and simplification of classifiers done by an Espresso-like simplifier. *Comparison*: budget of $2n_{cl}n_v n_c^{n_v}$ evaluations; as to population size, GP and SGP have $\max\{\sqrt{n_c^{n_v}}, 10\}$, and GPt has $\max\{\sqrt{n_c^{n_v}}, 50\}$ (and from 10 to 130 times more evaluations).

Table 1. Problems: standard boolean benchmark suite. Hits %: percentage of training examples correctly predicted by best solution; average (avg) and standard deviation (sd) of 30 runs. Length: logarithm base 10 of the length of the largest solution encountered in the search.

Problem	**Hits %**								**Length**			
	GP		GPt		SSHC		SGP					
	avg	sd	avg	sd	avg	sd	avg	sd	GP	GPt	SSHC	SGP
Comparator6	80.2	3.8	90.9	3.5	99.8	0.5	99.5	0.7	1.0	2.0	2.9	2.8
Comparator8	80.3	2.8	94.9	2.4	100.0	0.0	99.9	0.2	1.0	2.3	2.9	3.0
Comparator10	82.3	4.3	95.3	0.9	100.0	0.0	100.0	0.1	1.6	2.4	2.7	3.0
Multiplexer6	70.8	3.3	94.7	5.8	99.8	0.5	99.5	0.8	1.1	2.2	2.7	2.9
Multiplexer11	76.4	7.9	88.8	3.4	100.0	0.0	99.9	0.1	2.2	2.4	2.9	2.6
Parity5	52.9	2.4	56.3	4.9	99.7	0.9	98.1	2.1	1.4	1.7	2.9	2.9
Parity6	50.5	0.7	55.4	5.1	99.7	0.6	98.8	1.7	1.0	1.9	3.0	3.0
Parity7	50.1	0.2	51.7	2.8	99.9	0.2	99.5	0.6	1.0	1.7	3.0	3.1
Parity8	50.1	0.2	50.6	0.9	100.0	0.0	99.7	0.3	1.0	1.6	3.4	3.4
Parity9	50.0	0.0	50.2	0.1	100.0	0.0	99.5	0.3	1.0	1.3	3.8	3.8
Parity10	50.0	0.0	50.0	0.0	100.0	0.0	99.4	0.2	0.9	1.2	4.1	4.1
Random5	82.2	6.6	90.9	6.0	99.5	1.2	98.8	2.1	0.9	1.6	2.7	2.8
Random6	83.6	6.6	93.0	4.1	99.9	0.4	99.2	1.3	1.2	1.9	2.9	2.8
Random7	85.1	5.3	92.9	3.8	99.9	0.2	99.8	0.4	1.1	2.0	2.8	2.9
Random8	89.6	5.3	93.7	2.4	100.0	0.1	99.9	0.2	1.4	2.0	3.0	2.9
Random9	93.1	3.7	95.4	2.3	100.0	0.1	100.0	0.1	1.5	1.8	2.9	2.9
Random10	95.3	2.3	96.2	2.0	100.0	0.0	100.0	0.0	1.5	1.8	2.8	3.0
Random11	96.6	1.6	97.3	1.5	100.0	0.0	100.0	0.0	1.6	1.7	2.7	3.1
True5	100.0	0.0	100.0	0.0	99.9	0.6	100.0	0.0	1.1	1.3	2.0	2.4
True6	100.0	0.0	100.0	0.0	99.8	0.6	100.0	0.0	1.2	1.2	2.6	2.5
True7	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	1.2	1.2	2.9	2.6
True8	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.1	1.2	1.4	3.3	2.9

Table 2. Problems: Random Polynomials of degrees 3 to 10. Hits %: percentage of training examples correctly predicted by best solution with tolerance 0.01; avg and sd of 30 runs.

Problem	**Hits %**					
	GP		SSHC		SGP	
	avg	sd	avg	sd	avg	sd
Polynomial3	79.9	23.1	100.0	0.0	99.5	1.5
Polynomial4	60.5	27.6	99.9	0.9	99.9	0.9
Polynomial5	40.7	21.6	100.0	0.0	99.5	2.0
Polynomial6	37.5	23.4	100.0	0.0	98.9	3.1
Polynomial7	30.7	18.5	100.0	0.0	99.9	0.9
Polynomial8	34.7	16.0	99.5	2.0	99.7	1.3
Polynomial9	20.7	13.2	100.0	0.0	98.5	4.9
Polynomial10	25.7	16.7	99.4	1.7	99.9	0.9

Analysis: *Performance*: for all domains and problems, SSHC and SGP find consistently near-optimal solutions, beating by far GP with the same budget of evaluations, and also GPt with the same CPU time. *Size*: SSHC and SGP produce individuals larger than GP. This is due to a limited amount of simplification applied that finds shorter but usually not the shortest expressions, and, for some problems, to the optimal solution having a long encoding in the chosen representation. Importantly, experiments show that the simplification counteracts effectively the exponential growth of individuals inherent in the semantic operators, within affordable computational cost. *Bias*: semantic operators see any problem as a cone landscape, hence potentially easy. However they may

Table 3. Problems: see text. Hits % and Length same as in Table 1.

Problem			Hits %								Length			
			GP		GPt		SSHC		SGP					
n_v	n_c	n_{cl}	avg	sd	avg	sd	avg	sd	avg	sd	GP	GPt	SSHC	SGP
3	3	2	80.00	8.41	97.30	4.78	99.74	0.93	99.89	0.67	1.6	1.9	2.3	2.3
3	3	4	49.15	9.96	78.89	8.93	99.89	0.67	99.00	1.63	1.6	2.1	2.3	2.3
3	3	8	37.04	5.07	59.52	14.26	99.74	0.93	96.04	2.85	1.2	1.9	2.3	2.3
3	4	2	67.92	7.05	93.80	5.41	99.95	0.28	99.58	0.80	1.8	2.3	2.7	2.7
3	4	4	39.11	7.02	68.48	8.66	99.84	0.47	98.08	1.64	1.7	2.3	2.7	2.7
3	4	8	28.02	3.73	46.98	14.48	99.73	0.58	94.22	1.72	1.1	2.0	2.7	2.7
4	3	2	88.31	6.98	98.89	2.89	99.96	0.22	100.00	0.00	1.6	1.9	2.9	2.9
4	3	4	48.85	6.54	88.15	10.10	100.00	0.00	99.54	0.68	1.4	2.2	2.9	2.9
4	3	8	36.54	9.01	60.37	17.14	100.00	0.00	96.63	1.23	1.0	1.9	2.9	2.9
4	4	2	82.75	8.21	99.79	1.12	100.00	0.00	99.86	0.23	2.2	2.3	3.3	3.3
4	4	4	44.13	8.75	77.55	6.30	100.00	0.00	99.68	0.29	2.0	2.4	3.3	3.3
4	4	8	30.63	5.33	50.21	15.08	99.96	0.12	98.84	0.58	1.4	2.1	3.3	3.3

have heavy biases in the offspring distributions that hinder performance. Experiments show that these biases do not prevent achieving very good performance.

5 Conclusions and Future Work

We presented a new GP framework rooted in a geometric theory of representations to search *directly* the semantic space of functions/programs. Remarkably, the landscape seen by the semantic operators is *always* a cone by construction, hence generally easy to search. Seen from a geometric viewpoint, the genotype-phenotype mapping of GP becomes *very easy*. This allowed us to derive explicit algorithmic characterization of semantic operators for different domains following a *simple recipe*. Semantic operators require *simplification*, which on the domains considered was not a problem. In the experiments, the semantic approach *systematically outperformed* standard GP. There is plenty of future work and open challenges: (i) construct semantic operators for more complex domains, to explore potentials and limits of the framework; (ii) use formal methods to simplify non-trivial programs with loops/recursion, and use CAS to simplify non-polynomial functions, and, more generally, devise quick heuristic simplifiers for complex domains; (iii) investigate the practical advantages of different types of syntax/languages: e.g., programs written in minimalistic languages with strong theory, like lambda calculus, may be much easier to simplify; also, certain syntax may allow to implement easily semantic operators with probabilistic biases that make them more effective in practice; (iv) derive analytical runtime: as semantic GP search is equivalent to standard GAs/ES on cone landscapes, it should be easy to transfer analytical runtime results to semantic GP, and determine the optimal parameter settings.

References

1. Beadle, L., Johnson, C.G.: Sematically driven crossover in genetic programming. In: Proc. of IEEE WCCI 2008, pp. 111–116 (2008)
2. Beadle, L., Johnson, C.G.: Semantic analysis of program initialisation in genetic programming. Genetic Programming and Evolvable Machines 10(3), 307–337 (2009)

3. Beadle, L., Johnson, C.G.: Semantically driven mutation in genetic programming. In: Proc. of IEEE CEC 2009, pp. 1336–1342 (2009)
4. Jackson, D.: Phenotypic Diversity in Initial Genetic Programming Populations. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 98–109. Springer, Heidelberg (2010)
5. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Proc. of GECCO 2009, pp. 987–994 (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. Luke, S.: The ECJ Owner's Manual – A User Manual for the ECJ Evolutionary Computation Library (2010)
8. Moraglio, A.: Towards a Geometric Unification of Evolutionary Algorithms. PhD thesis, University of Essex (2007)
9. Moraglio, A.: Abstract convex evolutionary search. In: Proc. of FOGA 2011, pp. 151–162 (2011)
10. Moraglio, A., Poli, R.: Topological Interpretation of Crossover. In: Deb, K., Tari, Z. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 1377–1388. Springer, Heidelberg (2004)
11. Uy, N.Q., et al.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines 12(2), 91–119 (2011)

Efficient Negative Selection Algorithms by Sampling and Approximate Counting

Johannes Textor

Theoretical Biology & Bioinformatics
Universiteit Utrecht
Paudalaan 8
3584 CH Utrecht, NL
johannes.textor@gmx.de

Abstract. Negative selection algorithms (NSAs) are immune-inspired anomaly detection schemes that are trained on normal data only: A set of *consistent detectors* – i.e., detectors that do not match any element of the training data – is generated by rejection sampling. Then, input elements that are matched by the generated detectors are classified as anomalous. NSAs generally suffer from exponential runtime. Here, we investigate the possibility to accelerate NSAs by sampling directly from the set of consistent detectors. We identify conditions under which this approach yields fully polynomial time randomized approximation schemes of NSAs with exponentially large detector sets. Furthermore, we prove that there exist detector types for which the approach is feasible even though the only other known method for implementing NSAs in polynomial time fails. These results provide a firm theoretical starting point for implementing efficient NSAs based on modern probabilistic techniques like Markov Chain Monte Carlo approaches.

1 Introduction

The adaptive immune system is, alongside the central nervous system, one of the two important cognitive systems in vertebrates. Within this system, the *T cells* are responsible for performing many important cognitive tasks, like detecting viral infections in cells. Because T cells can perform actions with potentially hazardous consequences for their host organism – e.g., they can kill cells that express "anomalous" surface molecules – it is important to ensure that T cells do not incorrectly classify normal metabolic activity as anomalous. At the same time, it is crucial that an organism's T cell repertoire provides protection against the huge set of pathogens it may potentially encounter. *Negative selection* is an immunological process that helps achieve these two goals: Newborn T cells with randomly generated receptors are exposed to normal molecular structures from the host organism (self), and those that react to any self structure are killed. Only cells that survive negative selection become part of the T cell repertoire. Motivated by a need for better computer security systems, Forrest et al. [1] conceived a generic classification scheme, which they called the *negative selection algorithm* (NSA), that mimics this simple yet effective biological paradigm.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 32–41, 2012.

The NSA scheme can be applied to very diverse types of data. We will assume that the data to be classified originates from a universe $\mathcal{U}$, which is usually parameterized by some index L that characterizes the length of an input element (e.g., for an alphabet Σ, we might use $\mathcal{U} = \Sigma^L$). Moreover, we assume that a basis set of patterns (detectors) $\mathcal{P}$ is given, such that each $\pi \in \mathcal{P}$ matches a subset of $\mathcal{U}$. The patterns in $\mathcal{P}$ represent T cells, while the elements of $\mathcal{U}$ represent the molecular structures examined by T cells. For instance, a frequently used class of patterns, motivated by a model of how real T cells "see" antigen [2], are the so-called "r-contiguous detectors". Here, $\mathcal{U} = \mathcal{P} = \Sigma^L$, and a pattern is said to match a universe element if both are identical in at least r contiguous positions. E.g., for $r = 2$ and $\mathcal{U} = \{0,1\}^3$, the patterns 011 and 110 match the string 010 but the pattern 111 does not. As another example, we could use $\mathcal{U} = \{0,1\}^L$ and $\mathcal{P} = \{0,1,*\}^L$ to encode binary patterns with don't-care-symbols (e.g. $0**$ would match all strings of length 4 starting with 0).

NegativeSelection(S, M, n).
Input: Sample $S \subseteq \mathcal{U}$, set $M \subseteq \mathcal{U}$, integer n.
Output: For each $m \in M$, either $(m, +1)$ or $(m, -1)$.

```
1   D ← ∅
2   while |D| < n do        // training step
3       pick π ∈ P uniformly at random
4       if π does not match any s ∈ S then
5           D ← D ∪ {π}
6   for each m ∈ M do       // classification step
7       if any π ∈ D matches m then
8           output (m, +1)
9       else
10          output (m, −1)
```

Fig. 1. Pseudocode of the negative selection algorithm (NSA) considered in this paper. In the literature on NSAs (e.g., [3,4]), S is frequently called a *self set*, M a *monitor set*, and D is called the *detector set*. For the sake of conciseness, we treat the set D as a multi-set, i.e., D can contain the same detector more than once.

Fig. 1 shows the pseudocode for a typical NSA scheme. A set of detectors D with size n is generated by rejection sampling, i.e., detectors are sampled at uniform and added to D if they match no element of the input sample S. Subsequently, in the classification step, the elements of M are classified as anomalous $(+1)$ if matched by any detector in D and as normal (-1), otherwise.

2 Related Work and Our Contribution

Layman implementations of the NSA typically suffer from exponential runtime [5,4]. Two issues can arise: First, the rejection sampling step may be rate-limiting if the input set S is "large" such that most detectors match some $s \in S$, and have to be rejected. Second, prohibitively many detectors might be needed to achieve

acceptable detection rates in the classification step. The main contribution of this work is that we establish a novel possibility of implementing NSA algorithms efficiently: We investigate under which conditions it is possible to replace the rejection sampling in the training step by a more efficient procedure that requires only polynomial time to find a single detector. Moreover, we show that a similar sampling approach can be employed to determine the NSA outcome approximately even when n is very large.

Our research on the efficiency of NSAs has two main motivations. First, a widely held view in the field of artificial immune systems (AIS) used to be that NSAs cannot be efficiently implemented. E.g., for r-contiguous detectors, it was hypothesized that even deciding the existence of a single detector that fails to match all $s \in S$ is already NP-complete [3]. This idea led some researchers to conclusions like "negative selection [algorithms] ... can never scale" [6], or that "future work in this direction is not meaningful" [5]. It stands to reason that one wishes to verify whether unproved claims that motivate such bold statements really hold true. For some special cases, we have previously shown that polynomial-time NSAs can be obtained using the "detector compression" technique [7,8], thus disproving the NP-completeness hypothesis for r-contiguous detectors [3]. However, we subsequently proved that the detector compression technique is not applicable to many interesting types of detectors [9], which raised the question whether other methods might exist to obtain efficient NSAs. The technique that we put forward in this paper is able to address at least some cases where detector compression is infeasible.

Independently of the debate in the AIS community, NSAs find their main application in the field of theoretical immunology, where they are used as components of simulations of the real immune system (e.g. [10,11,12]). Recently, a NSA-based model was used to show that genetically determined differences in negative selection can partly explain why certain individuals are able to control HIV infections [13]. In these computational biology applications, the NSA cannot be replaced by traditional machine learning methods because it is used as a simulation of the real negative selection process rather than merely a generic classification scheme.

Therefore, increasing the efficiency of NSAs benefits not only AIS but is also important for computational immunology.

3 Our Approach

The basic idea behind our approach is very simple: Instead of generating the detector set D in the training step by rejection sampling (lines 2-5 in Fig. 1), we sample directly from the subset of those detectors in $\mathcal{P}$ that do not match any $s \in S$. For instance, in many cases we might be able to construct a graph that encodes the desired detectors, perform a "sufficiently long" random walk on this graph, and output the last vertex we visit. Provided the random walk is "rapidly mixing" (i.e., approaches the equilibrium distribution sufficiently fast),

this approach, which is known as the *Markov Chain Monte Carlo method* [14], can efficiently generate an approximately uniform sample from the set of S-consistent detectors.

To proceed, we need some notation. We abbreviate the set $\{1, \ldots, n\} \subseteq \mathbb{N}$ by $[1, n]$ and the set $\{-1, 1\}$ by ± 1. Moreover we write "u.a.r." instead of "uniformly at random". Given a universe $\mathcal{U}$, a *detector type* $\mathcal{D} = (\mathcal{P}, \mathcal{M})$ is a tuple of a set $\mathcal{P}$ of *patterns* (or detectors) and a *matching function* $\mathcal{M} : \mathcal{P} \times \mathcal{U} \to \pm 1$. Given a detector $\pi \in \mathcal{P}$ and an element $x \in \mathcal{U}$, if $\mathcal{M}(\pi, x) = 1$ we say "π matches x" and "π does not match x", otherwise. A *sample* is a labeled set $S \subseteq \mathcal{U} \times \pm 1$. A *negative sample* is a sample in which all labels are -1. A detector $\pi \in \mathcal{P}$ is called S-*consistent* if for every $(x, +1) \in S$, we have $\mathcal{M}(\pi, x) = +1$, and for every $(x, -1) \in S$, we have $\mathcal{M}(\pi, x) = -1$. A detector set $D \subseteq \mathcal{P}$ is called S-consistent if it only contains S-consistent detectors. The set of *all* S-consistent detectors is written as $\mathcal{D}[S]$. The *consistency problem* is defined as follows: Given a sample S, decide whether $\mathcal{D}[S]$ is empty. A consistency problem is called k^+-*restricted* if it is only defined for input samples that contain exactly k elements labeled with $+1$.

Given a negative sample S and an element $x \in U$, the *detector sampling distance* [9] $\Delta_{\mathcal{D}}(S, x)$ is defined by

$$\Delta_{\mathcal{D}}(S, x) = \begin{cases} \frac{|\mathcal{D}[S \cup \{(x,+1)\}]|}{|\mathcal{D}[S]|} & \mathcal{D}[S] \neq \emptyset \\ \bot & \text{otherwise} \end{cases}, \tag{1}$$

with $\bot$ denoting the undefined value. We previously showed the following.

Theorem 1 (Simulating NSAs via the Sampling Distance [9]). *If $\Delta_{\mathcal{D}}$ can be computed in expected polynomial time*[1]*, then there exists a randomized algorithm that is input-output-equivalent to* NegativeSelection(S, M, n) *and runs in expected polynomial time.*

Instead of simulating the NSA explicitly, we can also simply output $\Delta_{\mathcal{D}}(S, m)$ for every input element $m \in M$ and use this fraction as an "anomaly score". For this application, we will be satisfied if we can compute only the numerator of Equation 1, because the denominator is anyway equal for all $m \in M$. However, counting solution sets of combinatorial problems is often infeasible – we provided some according negative results previously [9]. Therefore, the method that we put forward in this paper rests on a slightly weaker precondition.

Proposition 1. *Suppose there exists an algorithm that, for every negative input sample $S \subseteq \mathcal{U} \times \{-1\}$, outputs a detector $\pi \in \mathcal{D}[S]$ sampled u.a.r. in expected polynomial time. Then there exists an input-output-equivalent algorithm for* NegativeSelection(S, M, n) *that runs in expected polynomial time in the size of S and M as well as in n.*

[1] Throughout the paper, this means that the expected runtime must be polynomial for *all* possible inputs.

Hence, by sampling directly from $\mathcal{D}[S]$, we can overcome the efficiency problems of rejection sampling. However, for achieving acceptable detection rates (in AIS) or for simulating realistic immune systems (in computational immunology), n often has to be very large. Therefore, we would like to avoid generating detectors explicitly. This too can be achieved if we sample from both $\mathcal{D}[S]$ *and* $\mathcal{D}[S \cup \{(x, +1\}]$, i.e., from the set of consistent detectors for both 0^+-restricted and 1^+restricted input samples.

Proposition 2. *Suppose there exists an algorithm as defined in Proposition 1 and another algorithm that, for every input sample $S \subseteq \mathcal{U} \times \{-1\}$ and every element $x \in \mathcal{U}$, samples a detector $d \in \mathcal{D}[S \cup \{(m, +1)\}]$ u.a.r. Furthermore, assume that the 0^+- and 1^+-restricted consistency problems for $\mathcal{D}$ are both "self-reducible" [15]. Then there exists a fully polynomial time randomized approximation scheme (FPRAS) for computing the detector sampling distance $\Delta_{\mathcal{D}}(S, m)$.*

Proof. For self-reducible problems, Jerrum et al. [15] showed that a polynomial time u.a.r. sampler[2] can be used to construct an algorithm that determines the number of solutions within factor $1 + \epsilon$ in polynomial time in both the input size and $1/\epsilon$. Applying this theorem, we obtain FPRASs for computing both the numerator and the denominator of Equation 1. Therefore, we can approximate $\Delta_{\mathcal{D}}$ within factor $1 + \epsilon'$, where $\epsilon' = 2\epsilon + \epsilon^2$. □

For lack of space, we cannot reproduce the precise definition of self-reducibility here, and refer the reader instead to Jerrum et al [15]. Intuitively, self-reducibility means that solutions to the whole problem can be constructed by extending solutions of slightly smaller instances. Self-reducibility seems to be "the rule rather than the exception" [15] for combinatorial problems. For instance, it is easy to show that all detector types that have so far been used in string-based negative selection (summarized in [9]) lead to self-reducible consistency problems.

Hence, via efficient detector sampling, we can approximate the results of detector compression techniques [8]. A natural question is therefore whether efficient detector sampling can be possible when efficient detector compression (which leads to detector counting algorithms) is not. In the upcoming section, we prove that this can indeed be the case.

Theorem 2. *There exists a detector type $\mathcal{D}$ for which (1) computing $\Delta_{\mathcal{D}}$ is #P-hard, and (2) there exist expected polynomial time algorithms for sampling u.a.r. from both $\mathcal{D}[S \cup \{(x, +1)\}]$ and $\mathcal{D}[S]$, implying an FPRAS for $\Delta_{\mathcal{D}}$.*

The classical example for a combinatorial problem where counting the solutions is #P-hard but sampling from the solution space is easy is DNF-satisfiability [15]. However, consistency problems correspond to conjunctions of constraints. Therefore, there appears to be no way to encode DNF-satisfiability in a 0^+- and 1^+-restricted consistency problem (without exponential blowup), which is the main technical challenge that needs to be overcome to prove Theorem 2.

[2] In fact, it is only required to sample approximately u.a.r. from $\mathcal{D}[S]$ and $\mathcal{D}[S \cup \{(x, +1)\}]$. However, in this paper we restrict ourselves to u.a.r. sampling.

4 Proof of the Main Theorem

We first have to prove a technical lemma.

Lemma 1 (Embedding an Additional Object into a Uniform Sampler). *Let X be a finite set of unknown cardinality $|X| > 1$, and suppose that there exists an algorithm $\mathcal{A}$ that generates an element of X u.a.r. in expected polynomial time. Let $x^* \notin X$. Then there exists an algorithm $\mathcal{A}^*$ that generates an element of $X \cup \{x^*\}$ u.a.r. in expected polynomial time.*

Proof. Let n denote the unknown cardinality of X. Our procedure $\mathcal{A}^*$ works as follows: (1) $\mathcal{A}^*$ samples an element $a \in X$ u.a.r. (2) $\mathcal{A}^*$ repeatedly samples a tuple $(x, y) \in X^2$ u.a.r. until $(x, y) \neq (a, a)$. (3) If $x = a$, then $\mathcal{A}^*$ outputs x^*. Otherwise, $\mathcal{A}^*$ outputs a. Now the probability that $\mathcal{A}^*$ outputs x^* is

$$\frac{n-1}{n^2-1} = \frac{1}{n+1} ,$$

and thus the output probability distribution is uniform over $X \cup \{x^*\}$. The lemma now follows by noting that because $|X| \geq 2$, step (2) above terminates after a constant expected number of iterations. □

Now we are prepared to prove Theorem 2.

Proof (Theorem 2). The basic idea is to define a detector type $\mathcal{D}$ whose consistency problem amounts to finding graph colorings. We recall that a k-coloring of a graph $G = (V, E)$ is a mapping $C : V \to [1, k]$, and it is called *valid* if $C(v) \neq C(w)$ for all $\{v, w\} \in E$. Counting the k-colorings of a graph with maximal degree κ is #P-hard for all constants $k, \kappa \geq 3$ [16]. Still, for $k > \kappa(\kappa + 2)$ there exists an algorithm that samples u.a.r. in expected polynomial time from the valid k-colorings of a given graph. Below, we assume that $\kappa \leq 3$ and $k \geq 16$. This includes the #P-hard case $\kappa = 3, k = 16$ as a special case, which will suffice to establish our hardness result. Note that a graph of maximum degree 3 is always 16-colorable, such that the decision version of the graph coloring problem is trivial for these constants. In the following, we denote the maximum degree of a given graph G by $\kappa(G)$.

To make the proof more palatable, we proceed in three steps. First we show that there exists a detector type $\mathcal{D}$ for which determining $|\mathcal{D}[S]|$ is #P-hard, even though we can sample u.a.r. from $\mathcal{D}[S]$ for 0^+-restricted samples S. This is not yet exactly what we need because $|\mathcal{D}[S]|$ is only the denominator of the detector sampling distance $\Delta_{\mathcal{D}}$ (see Equation 1); infeasibility of computing the denominator of a fraction does not imply infeasibility of computing the entire fraction. In the second step, we deal with this technicality. In the third step, we show the feasibility of sampling consistent detectors for both 0^+- and 1^+-restricted samples u.a.r.

Step 1. Our universe $\mathcal{U}$ is the set of graphs with maximum degree ≤ 3 and one labeled edge, which we call the *root edge* ρ:

$$\mathcal{U} = \{(V, E, \rho) : V = [1, L],\ E \subseteq \{e \subseteq V : |e| = 2\},\ \rho \in E,\ \kappa(V, E) \leq 3\} .$$

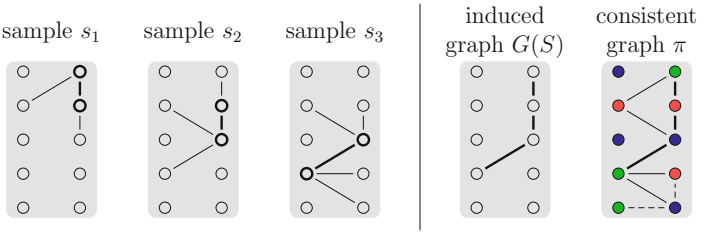


Fig. 2. Illustration of the proof of Theorem 2. The definition of $\mathcal{M}$ ensures that every edge $\{u, v\}$ sharing a node with a root edge in one sample must also occur in every other sample whose root edge contains u or v, otherwise there exists no S-consistent pattern. For example, if any of the non-root edges in sample s_2 were missing, then no pattern could be consistent with $(s_1, -1), (s_2, -1)$ and $(s_3, -1)$. The induced graph $G(S)$ is the union of all root edges. The pattern set $\mathcal{P}$ is the set of all colored graphs, and a graph $\pi \in \mathcal{P}$ is S-consistent if and only if it contains all edges from the samples and its nodes are validly colored with respect to $G(S)$. Edges not occurring in the samples may only occur in π if they do not touch any root edge from S, like the dashed edges in the rightmost graph.

Figure 2 shows three examples s_1, s_2, and s_3 (with roots edges ρ in bold).

We are going to define a detector type $\textsc{gcol} = (\mathcal{P}, \mathcal{M})$ whose pattern set $\mathcal{P}$ and matching function $\mathcal{M}$ will be constructed in such a way that a negative sample $S \subseteq \mathcal{U} \times \{-1\}$ encodes an *induced graph* $G(S)$. Counting S-consistent patterns will be equivalent to counting the valid k-colorings of $G(S)$. We define

$$\mathcal{P} = \{(V, E, C) : V = [1, L],\ E \subseteq \{e \subseteq V : |e| = 2\},\ C : V \to [1, k]\}\ ,$$

which is simply the set of all arbitrarily k-colored graphs with L vertices (the coloring need not be valid). The matching function is defined as follows:

$$\mathcal{M}_{\textsc{gcol}}\left((V, E, C), (V, E', \rho)\right) = \begin{cases} +1 & (1)\ E' \nsubseteq E \\ & \text{or } (2)\ E \setminus E' \text{ contains a } \rho\text{-adjacent edge} \\ & \text{or } (3)\ C \text{ is no valid coloring for } \rho \\ -1 & \text{otherwise} \end{cases}$$

Now consider a negative sample S, and suppose there does not exist an S-consistent pattern $\pi = (V, E, C)$. This can occur if and only if there exist two samples $(V, E_1, \rho_1), (V, E_2, \rho_2) \in S$ and an edge $e \in E_1$ such that e shares a node with ρ_1 but not with ρ_2. In other words, an S-consistent pattern exists if and only if it holds true that once an edge $\{u, v\}$ appears in one sample graph where either u or v belongs to the root edge, it appears in *every* sample graph where u or v belongs to the root edge. Therefore, the existence of an S-consistent pattern can be decided in polynomial time.

Consider a negative sample $S = \{((V, E_1, \rho_1), -1), \ldots, ((V, E_n, \rho_n), -1))\}$ for which at least one consistent graph pattern $\pi = (V, E, C)$ exists. We define the *induced graph* $G(S)$ as the union of all root edges in S, i.e., $G(S) := (V, \cup_i \{\rho_i\})$.

The definition of $\mathcal{M}$ directly ensures that (1) every S-consistent graph $\pi \in \mathcal{P}$ contains $G(S)$ as a subgraph, and (2) C is a valid coloring of $G(S)$. Moreover, it is not hard to show that $G(S)$ has maximum degree ≤ 3 – if that weren't the case, then there would either be no S-consistent pattern or one sample with maximum degree > 3. Edges that did not occur in S and do not share nodes with $G(S)$ may or may not be present in π (dashed edges in the consistent graph π in Figure 2), and nodes that are not in $G(S)$ (like the top left and bottom left nodes in the consistent graph in Figure 2) are assigned an arbitrary color.

Let $\#\chi(G(S))$ be the number of valid k-colorings of $G(S)$. Let ϵ denote the number of edges not adjacent to nodes in $G(S)$ (those which may or may not be present in S-consistent graphs). Then, assuming $|\textsc{gcol}[S]| > 0$, we have

$$|\textsc{gcol}[S]| = \#\chi(G(S)) \cdot 2^{\epsilon} \ .$$

Now suppose we had an algorithm for computing $|\textsc{gcol}(S)|$. Then we could determine the number $\#\chi(G)$ of valid k-colorings for an arbitrary graph G of maximal degree $\kappa = 3$ as follows: Decompose $G = (V, E)$ into sample graphs by creating for each edge $e = \{u, v\} \subseteq V$ a sample graph containing root edge e and its adjacent edges in G. Create a sample S containing all these graphs with negative labels. Then $G(S) = G$, and

$$\#\chi(G) = \frac{|\textsc{gcol}[S]|}{2^{\epsilon}}$$

gives the number of valid k-colorings of G. Because ϵ can be computed in polynomial time from S, computing $|\textsc{gcol}[S]|$ must thus be #P-hard[3] for $k \geq 3$.

Conversely, given an arbitrary negative sample S over $\mathcal{U}$, we can sample from $\textsc{gcol}[S]$ as follows. First check whether $|\textsc{gcol}[S]| = 0$ as discussed above. If this is not the case, we compute the induced graph $G(S) = (V, E)$, and sample a valid coloring of $G(S)$ u.a.r. using Huber's algorithm [14]. Next, consider every edge $\{u, v\} \subseteq V$ that does not occur in S and does not share nodes with root edges from S, and insert $\{u, v\}$ into E with probability $1/2$. The resulting graph is sampled u.a.r. from $\textsc{gcol}[S]$. An example result of this process is depicted as the rightmost graph in Figure 2.

Step 2. So far we proved that computing $|\textsc{gcol}[S]|$ is #P-hard. To show that computing $\Delta_{\textsc{gcol}}$ is also #P-hard, we insert a special pattern $\hat{\pi}$ into $\mathcal{P}$ and a special element $\hat{x}$ into $\mathcal{U}$ such that $\hat{\pi}$ matches only $\hat{x}$ and vice versa. From now on, let $\textsc{gcol}$ denote this augmented pattern class. Suppose we had access to an oracle that computes $\Delta_{\textsc{gcol}}(S, \hat{x})$. We could use this oracle to count the k-colorings of a graph $G = (V, E), V = [1, L]$, as follows: We create a negative sample S with $G(S) = G$. Then

$$\Delta_{\textsc{gcol}}(S, \hat{x}) = \frac{1}{1 + \#\chi(G)\, 2^{\epsilon}} \ .$$

[3] Strictly speaking, only functions to the natural numbers can be #P-hard. Therefore, more formally correctly we should say that every #P-hard function could be computed by a polytime algorithm with single-call access to an oracle for $|\textsc{gcol}[S]|$.

Hence, we could compute $\#\chi(G)$ from $\Delta_{\text{GCOL}}(S, \hat{x})$ by rearranging the above, which implies that computing $\Delta_{\text{GCOL}}(S, \hat{x})$ is #P-hard.

Step 3. It remains to show that for all $S \subseteq \mathcal{U} \times \{-1\}$ and for all $m \in \mathcal{U}$, we can generate elements of both $\text{GCOL}[S]$ and $\text{GCOL}[S \cup \{(m, +1)\}]$ u.a.r. in expected polynomial time. We start with the case where there is no positive sample. If S contains $\hat{x}$, then $\hat{\pi}$ is not S-consistent, and we output a pattern sampled at uniform from $\text{GCOL}[S]$. Otherwise, $\hat{\pi}$ is S-consistent, and we apply Lemma 1 to sample a pattern at uniform from $\text{GCOL}[S] \cup \{\hat{\pi}\}$. Now, consider the case where there is one positive sample $(m, +1)$. If m is a subgraph of the union of all sample graphs and the root of m occurs as a root in S, then there is no S-consistent pattern that matches m. Otherwise[4], m contains at least one edge e not present in S. We iteratively generate S-consistent patterns π u.a.r until we find one that matches m. Each π will match m with probability $\geq 1/2$, because the probability that π contains e is $1/2$. Therefore, after a constant expected number of trials we find the desired π, which is u.a.r. from $\text{GCOL}[S \cup \{(m, +1)\}]$.

5 Outlook

In this paper we presented a novel generic approach for implementing negative selection algorithms (NSAs) efficiently and proved that there exist cases where the new approach is feasible even though the detector compression technique that we put forward previously [9] is not. An obviously desirable next step would be to demonstrate the feasibility of this approach in practice. Sampling-based approximation algorithms, in particular Markov Chain Monte Carlo (MCMC) methods, have proved successful in many areas, e.g. Bayesian network analysis, even in cases where rigorous proofs for convergence in polynomial time (which can be notoriously difficult) are lacking. One appealing feature of MCMC approaches is their typical ease of implementation. This presents an advantage over detector compression, which often relies on rather intricate data structures [8]. To illustrate this, we conclude by defining an MCMC method for a natural detector type, Boolean monomials, and leave proving or disproving its efficient convergence as an open problem.

Open Problem. For $\mathcal{U} = \{0, 1\}^L$, $\mathcal{P} = \{0, 1, *\}^L$, let π match x if π and x are identical at all positions i where $\pi_i \neq *$ (π can be interpreted as a Boolean monomial). Given $S \subseteq \mathcal{U}^L \times \{-1\}$ and $m \in \mathcal{U}^L$, generate an $S \cup \{(m, +1)\}$-consistent pattern π as follows. Initialize $\pi = m$. Then, in each step, do nothing with probability $1/2$ and else, pick a position $i \in [1, L]$ u.a.r. If $\pi_i = *$, set $\pi_i = m_i$. Otherwise, replace π_i by $*$ and check whether the resulting pattern matches any $s \in S$. If yes, undo the change, else continue. This algorithm describes an ergodic Markov chain M whose stationary distribution is uniform over all $S \cup \{(m, +1)\}$-consistent patterns. Prove or disprove: M is rapidly mixing.

[4] For lack of space, we omit the required, but technical special treatment of $|V| < 5$.

References

1. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 202–212. IEEE Computer Society Press (1994)
2. Percus, J.K., Percus, O.E., Perelson, A.S.: Predicting the size of the T-cell receptor and antibody combining region from consideration of efficient self-nonself discrimination. Proceedings of the National Academy of Sciences of the United States of America 90(5), 1691–1695 (1993)
3. Timmis, J., Hone, A., Stibor, T., Clark, E.: Theoretical advances in artificial immune systems. Theoretical Computer Science 403, 11–32 (2008)
4. Stibor, T.: Foundations of r-contiguous matching in negative selection for anomaly detection. Natural Computing 8, 613–641 (2009)
5. Stibor, T.: On the Appropriateness of Negative Selection for Anomaly Detection and Network Intrusion Detection. PhD thesis, Technische Universität Darmstadt (2006)
6. Aickelin, U.: Special issue on artificial immune systems: editorial. Evolutionary Intelligence 1(2), 83–84 (2008)
7. Elberfeld, M., Textor, J.: Efficient Algorithms for String-Based Negative Selection. In: Andrews, P.S., Timmis, J., Owens, N.D.L., Aickelin, U., Hart, E., Hone, A., Tyrrell, A.M. (eds.) ICARIS 2009. LNCS, vol. 5666, pp. 109–121. Springer, Heidelberg (2009)
8. Elberfeld, M., Textor, J.: Negative selection algorithms on strings with efficient training and linear-time classification. Theoretical Computer Science 412, 534–542 (2011)
9. Liśkiewicz, M., Textor, J.: Negative selection algorithms without generating detectors. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2010), pp. 1047–1054. ACM (2010)
10. Chao, D.L., Davenport, M.P., Forrest, S., Perelson, A.S.: A stochastic model of cytotoxic T cell responses. Journal of Theoretical Biology 228, 227–240 (2004)
11. Chao, D.L., Davenport, M.P., Forrest, S., Perelson., A.S.: The effects of thymic selection on the range of T cell cross-reactivity. European Journal of Immunology 35, 3452–3459 (2005)
12. Košmrlj, A., Jha, A.K., Huseby, E.S., Kardar, M., Chakraborty, A.K.: How the thymus designs antigen-specific and self-tolerant T cell receptor sequences. Proceedings of the National Academy of Sciences of the USA 105(43), 16671–16676 (2008)
13. Košmrlj, A., Read, E.L., Qi, Y., Allen, T.M., Altfeld, M., Deeks, S.G., Pereyra, F., Carrington, M., Walker, B.D., Chakraborty, A.K.: Effects of thymic selection of the T-cell repertoire on HLA class I–associated control of HIV infection. Nature 465, 350–354 (2010)
14. Huber, M.: Exact sampling and approximate counting techniques. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 31–40. ACM, New York (1998)
15. Jerrum, M.R., Valiant, L.G., Vazirani, V.V.: Random generation of combinatorial structures from a uniform distribution. Theoretical Computer Science 43, 169–188 (1986)
16. Bubley, R., Dyer, M., Greenhill, C., Jerrum, M.: On approximately counting colourings of small degree graphs. SIAM Journal on Computing 29, 387–400 (1998)

Convergence of the Continuous Time Trajectories of Isotropic Evolution Strategies on Monotonic $\mathcal{C}^2$-composite Functions

Youhei Akimoto, Anne Auger, and Nikolaus Hansen

TAO Team, INRIA Saclay-Ile-de-France, LRI, Paris Sud University, France
{Youhei.Akimoto,Anne.Auger,Nikolaus.Hansen}@lri.fr

Abstract. The *Information-Geometric Optimization (IGO)* has been introduced as a unified framework for stochastic search algorithms. Given a parametrized family of probability distributions on the search space, the IGO turns an arbitrary optimization problem on the search space into an optimization problem on the parameter space of the probability distribution family and defines a natural gradient ascent on this space. From the natural gradients defined over the entire parameter space we obtain continuous time trajectories which are the solutions of an ordinary differential equation (ODE). Via discretization, the IGO naturally defines an iterated gradient ascent algorithm. Depending on the chosen distribution family, the IGO recovers several known algorithms such as the pure rank-μ update CMA-ES. Consequently, the continuous time IGO-trajectory can be viewed as an idealization of the original algorithm.

In this paper we study the continuous time trajectories of the IGO given the family of isotropic Gaussian distributions. These trajectories are a deterministic continuous time model of the underlying evolution strategy in the limit for population size to infinity and change rates to zero. On functions that are the composite of a monotone and a convex-quadratic function, we prove the global convergence of the solution of the ODE towards the global optimum. We extend this result to composites of monotone and twice continuously differentiable functions and prove local convergence towards local optima.

1 Introduction

Evolution Strategies (ESs) are stochastic search algorithms for numerical optimization. In ESs, candidate solutions are sampled using a Gaussian distribution parametrized by a mean vector and a covariance matrix. In state-of-the art ESs, those parameters are iteratively adapted using the ranking of the candidate solutions w.r.t. the objective function. Consequently, ESs are invariant to applying a monotonic transformation to the objective function. Adaptive ES algorithms are successfully applied in practice and there is ample empirical evidence that they converge linearly towards a local optimum of the objective function on a wide class of functions. However, their theoretical analysis even on simple functions is difficult as the state of the algorithm is given by both the mean vector and the covariance matrix that have a stochastic dynamic that needs to be simultaneously controlled. Their linear convergence to local optima is so far only proven for functions

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 42–51, 2012.

that are composite of a monotonic transformation with a convex quadratic function—hence function with a single optimum—for rather simple search algorithms compared to the covariance matrix adaptation evolution strategy (CMA-ES) that is considered as the state-of-the-art ES [1–4]. In this paper, instead of analyzing the exact stochastic dynamic of the algorithms, we consider the deterministic time continuous model underlying adaptive ESs that follows from the Information-Geometric Optimization (IGO) setting recently introduced [5].

The Information-Geometric Optimization is a unified framework for randomized search algorithms. Given a family of probability distributions parametrized by $\theta \in \Theta$, the original objective function, f, is transformed to a fitness function J_θ defined on Θ. The IGO algorithm defined on Θ performs a natural gradient ascent aiming at maximizing J_θ. For the family of Gaussian distributions, the IGO algorithm recovers the pure rank-μ update CMA-ES [6], for the family of Bernoulli distributions, PBIL [7] is recovered. When the step-size for the gradient ascent algorithm (that corresponds to a learning rate in CMA-ES and PBIL) goes to zero, we obtain an ordinary differential equation (ODE) in θ. The set of solutions of this ODE, the IGO-flow, consists of continuous time models of the recovered algorithms in the limit of the population size going to infinity and the step-size (learning rate for ES or PBIL) to zero.

In this paper we analyze the convergence of the IGO-flow for isotropic ESs where the family of distributions is Gaussian with covariance matrix equal to an overall variance times the identity. The underlying algorithms are step-size adaptive ESs that resemble ESs with derandomized adaptation [8] and encompass xNES [9] and the pure rank-μ update CMA-ES with only one variance parameter [6]. Previous works have proposed and analyzed continuous models of ESs that are solutions of ODEs [10,11] using the machinery of stochastic approximation [12,16]. The ODE variable in these studies encodes solely the mean vector of the search distribution and the overall variance is taken to be proportional to $H(\nabla f)$ where H is a smooth function with $H(0) = 0$. Consequently the model analyzed looses invariance to monotonic transformation of the objective function and scale-invariance, both being fundamental properties of virtually all ESs. The technique relies on the Lyapunov function approach and assumes the stability of critical points of the ODE [10,11]. In this paper, our approach also relies on the stability of the critical points of the ODE that we analyze by means of Lyapunov functions. However one difficulty stems from the fact that when convergence occurs, the variance typically converges to zero which is at the boundary of the definition domain Θ. To circumvent this difficulty we extend the standard Lyapunov method to be able to study stability of boundary points.

Applying the extended Lyapunov's method to the IGO-flow in the manifold of isotropic Gaussian distributions, we derive a sufficient condition on the so-called weight function w—parameter of the algorithm and usually chosen by the algorithm designer—so that the IGO-flow converges to the global minimum independently of the starting point on objective functions that are composite of a monotonic function with a convex quadratic function. We will call those functions *monotonic convex-quadratic-composite* in the sequel. We then extend this result to functions that are the composition of a monotonic transformation and a twice continuously differentiable function, called monotonic $\mathcal{C}^2$-composite in the rest of the paper. We prove local convergence to a local optimum of

the function in the sense that starting close enough from a local optimum, with a small enough variance, the IGO-flow converges to this local optimum.

The rest of the paper is organized as follows. In Section 2 we introduce the IGO-flow for the family of isotropic Gaussian distributions, which we call *ES-IGO*-flow. In Section 3 we extend the standard Lyapunov's method for proving stability. In Section 4 we apply the extended method to the ES-IGO-flow and provide convergence results of the ES-IGO-flow on monotonic convex-quadratic-composite functions and on monotonic $\mathcal{C}^2$-composite functions.

Notation. For $A \subset X$, where X is a topological space, we let A^c denote the complement of A in X, A^o the interior of A, $\overline{A}$ the closure of A, $\partial A = \overline{A} \setminus A^o$ the boundary of A. Let $\mathbb{R}$ and $\mathbb{R}^d$ be the sets of real numbers and d-dimensional real vectors, $\mathbb{R}_{\geqslant 0}$ and $\mathbb{R}_+$ denote the sets of non-negative and positive real numbers, respectively. Let $\|x\|$ represent the Euclidean norm of $x \in \mathbb{R}^d$. The open and closed balls in $\mathbb{R}^d$ centered at θ with radius $r > 0$ are denoted by $B(\theta, r)$ and $\overline{B}(\theta, r)$.

Let μ_{Leb} denote the Lebesgue measure on either $\mathbb{R}$ or $\mathbb{R}^d$. Let P_1 and P_d be the probability measures induced by the one-variate and d-variate standard normal distributions, p_1 and p_d the probability density function induced by P_1 and P_d w.r.t. μ_{Leb}. Let p_θ and P_θ represent the probability density function w.r.t. μ_{Leb} and the probability measure induced by the Gaussian distribution $\mathcal{N}(m(\theta), C(\theta))$ parameterized by $\theta \in \Theta$, where the mean vector $m(\theta)$ is in $\mathbb{R}^d$ and the covariance matrix $C(\theta)$ is a positive definite symmetric matrix of dimension d. We sometimes abbreviate $m(\theta(t))$ and $C(\theta(t))$ to $m(t)$ and $C(t)$. Let $\mathrm{vec} : \mathbb{R}^{d\times d} \to \mathbb{R}^{d^2}$ denote the vectorization operator such that $\mathrm{vec} : C \mapsto [C_{1,1}, C_{1,2}, \ldots, C_{1,d}, C_{2,1}, \ldots, C_{d,d}]^{\mathrm{T}}$, where $C_{i,j}$ is the i,j-th element of C. We use both notations: $\theta = [m^{\mathrm{T}}, \mathrm{vec}(C)^{\mathrm{T}}]^{\mathrm{T}}$ and $\theta = (m, C)$.

2 The ES-IGO-Flow

The IGO framework for continuous optimization with the family of Gaussian distributions is as follows. The original objective is to minimize an objective function $f : \mathbb{R}^d \to \mathbb{R}$. This objective function is mapped into a function on Θ. Hereunder, we suppose that f is μ_{Leb}-measurable. Let $w : [0,1] \to \mathbb{R}$ be a bounded, non-increasing weight function. We define the weighted quantile function [5] as

$$W_\theta^f(x) = w\big(P_\theta[y : f(y) \leqslant f(x)]\big) \ . \tag{1}$$

The function $W_\theta^f(x)$ is a preference weight for x according to the P_θ-quantile. The fitness value of θ' given θ is defined as the expectation of the preference W_θ^f over $P_{\theta'}$, $J_\theta(\theta') = \mathbb{E}_{x\sim P_\theta}\big[W_\theta^f(x)\big]$. Note that since $W_\theta^f(x)$ depends on θ so does $J_\theta(\theta')$. The function J_θ is defined on a statistical manifold $(\Theta, \mathcal{I})$ equipped with the Fisher metric $\mathcal{I}$ as a Riemannian metric. The Fisher metric is the natural metric. It is compatible with relative entropy and with KL-divergence and is the only metric that does not depend on the chosen parametrization. Using log-likelihood trick and exchanging the order of differentiation and integration, the "vanilla" gradient of J_θ at $\theta' = \theta$ can be expressed as $\nabla_{\theta'} J_\theta(\theta')|_{\theta'=\theta} = \mathbb{E}_{x\sim P_\theta}\big[W_\theta^f(x) \nabla_\theta \ln(p_\theta(x))\big]$. The natural gradient, that is, the

gradient taken w.r.t. the Fisher metric, is given by the product of the inverse of the Fisher information matrix $\mathcal{I}_\theta$ at θ and the vanilla gradient, namely $\mathcal{I}_\theta^{-1} \nabla_{\theta'} J_\theta(\theta')|_{\theta'=\theta}$. The IGO ordinary differential equation is defined as

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = \mathcal{I}_\theta^{-1} \nabla_{\theta'} J_\theta(\theta')\big|_{\theta'=\theta} \ . \tag{2}$$

Since the right-hand side (RHS) of the above ODE is independent of t the IGO ODE is autonomous. The IGO-flow is the set of solution trajectories of the above ODE (2).

When the parameter θ encodes the mean vector and the covariance matrix of the gaussian distribution in the following way $\theta = [m^\mathrm{T}, \mathrm{vec}(C)^\mathrm{T}]^\mathrm{T}$, the product of the inverse of the Fisher information matrix $\mathcal{I}_\theta^{-1}$ and the gradient of the log-likelihood $\nabla_\theta \ln(p_\theta(x))$ can be written in an explicit form [14] and (2) reduces to

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = \int W_\theta^f(x) \begin{bmatrix} x - m \\ \mathrm{vec}\big((x-m)(x-m)^\mathrm{T} - C\big) \end{bmatrix} P_\theta(\mathrm{d}x) \ . \tag{3}$$

The pure rank-μ update CMA-ES [6] can be considered as an Euler scheme for solving (3) with a Monte-Carlo approximation of the integral. Let $x_1, \ldots, x_n$ be samples independently generated from P_θ. Then, the quantile $P_\theta[y : f(y) \leqslant f(x_i)]$ in (1) is approximated by the number of solutions better than x_i divided by n, i.e., $\big|\{x_j, j = 1, \ldots, n : f(x_j) \leqslant f(x_i)\}\big|/n =: R_i/n$. Then $W_\theta^f(x_i)$ is approximated by $w\big((R_i - 1/2)/n\big)$, where w is the given weight function. The Euler scheme for approximating the solutions of (3) where the integral is approximated by Monte-Carlo leads to

$$\theta^{t+1} = \theta^t + \eta \sum_{i=1}^{n} \frac{w\big((R_i - 1/2)/n\big)}{n} \begin{bmatrix} x_i - m^t \\ \mathrm{vec}\big((x_i - m^t)(x_i - m^t)^\mathrm{T} - C^t\big) \end{bmatrix} \ , \tag{4}$$

where η is the time discretization step-size. This equation is equivalent to the pure rank-μ update CMA-ES when the learning rates η_m and η_C, for the update of m^t and C^t respectively, are set to the same value η, while they have different values in practice ($\eta_m = 1$ and $\eta_C \leqslant 1$). The summation on the RHS in (4) converges to the RHS of (3) with probability one as $\lambda \to \infty$ (Theorem 4 in [5]).

In the following, we study the simplified IGO-flow where the covariance matrix is parameterized by only a single variance parameter v as $C = vI_d$. Under the parameterization $\theta = [m^\mathrm{T}, v]^\mathrm{T}$, (2) reduces to $\frac{\mathrm{d}\theta}{\mathrm{d}t} = \int W_\theta^f(x) \left[\begin{smallmatrix} x-m \\ \|x-m\|^2/d - v \end{smallmatrix} \right] P_\theta(\mathrm{d}x)$. Using the change of variable $z = (x - m)/\sqrt{v}$, the above ODE reads

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = F_\theta(\theta) \ , \quad F_\theta(\theta) = \int W_\theta^f(m + \sqrt{v}z) \begin{bmatrix} \sqrt{v}z \\ v(\|z\|^2/d - 1) \end{bmatrix} P_d(\mathrm{d}z) \tag{5}$$

and we rewrite it by part

$$\frac{\mathrm{d}m}{\mathrm{d}t} = F_m(\theta) \ , \quad F_m(\theta) = \sqrt{v} \int W_\theta^f(m + \sqrt{v}z) z P_d(dz) \tag{6}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = F_v(\theta) \ , \quad F_v(\theta) = v \int W_\theta^f(m + \sqrt{v}z)(\|z\|^2/d - 1) P_d(dz) \ . \tag{7}$$

The domain of this ODE is $\Theta = \{\theta = (m, v) \in \mathbb{R}^d \times \mathbb{R}_+\}$. We call (5) the *ES-IGO* ordinary differential equation. The following proposition shows that for a Lipschitz continuous weight function w, solutions of the ODE (5) exist for any initial condition $\theta(0) \in \Theta$ and are unique.

Proposition 1 (Existence and Uniqueness). *Suppose w is Lipschitz continuous. Then the initial value problem: $\frac{\mathrm{d}\theta}{\mathrm{dt}} = F_\theta(\theta)$, $\theta(0) = \theta_0$, has a unique solution on $[0, \infty)$ for each $\theta_0 \in \Theta$, i.e. there is only one solution $\theta : \mathbb{R}_{\geqslant 0} \to \Theta$ to the initial value problem.*

Proof. We can obtain a lower bound $a(t) > 0$ and an upper bound $b(t) < \infty$ for $v(t)$ for each $t \geqslant 0$ under a bounded w. Similarly, we can have an upper bound $c(t) < \infty$ for $\|m(t)\|$. Then we have that $(m(t), v(t)) \in E(t) = \{x \in \mathbb{R}^d : \|x\| \leqslant c(t)\} \times \{x \in \mathbb{R}_+ : a(t) \leqslant x \leqslant b(t)\}$ and $E(t)$ is compact for each $t \geqslant 0$. Meanwhile, F_θ is locally Lipschitz continuous for a Lipschitz continuous w. Since $E(t)$ is compact, the restriction of F_θ into $E(t)$ is Lipschitz continuous. Applying Theorem 3.2 in [15] that is an extension of the theorem known as Picard-Lindelöf theorem or Cauchy-Lipschitz theorem, we have the existence and uniqueness of the solution on each bounded interval $[0, t]$. Since t is arbitrary, we have the proposition. □

Now that we know that solutions of the ES-IGO ODE exist and are unique, we define the ES-IGO-flow as the mapping $\varphi : \mathbb{R}_{\geqslant 0} \times \Theta \to \Theta$, which maps (t, θ_0) to the solution $\theta(t)$ of (5) with initial condition $\theta(0) = \theta_0$. Note that we can extend the domain of F_θ from $\Theta = \mathbb{R}^d \times \mathbb{R}_+$ to $\overline{\Theta} = \mathbb{R}^d \times \mathbb{R}_{\geqslant 0}$. It is easy to see from (5) that the value of $F_\theta(\theta)$ at $\theta = (m, 0)$ is 0 for any $m \in \mathbb{R}^d$. However, we exclude the boundary $\partial\Theta$ from the domain for reasons that will become clear in the next section. Because the initial variance must be positive and the variance starting from positive region never reach the boundary in finite time, solutions $\varphi(t, \cdot)$ will stay in the domain Θ. However, as we will see, they can converge asymptotically towards points of the boundary.

Since J_θ is *adaptive*, i.e. $J_{\theta_1}(\theta) \neq J_{\theta_2}(\theta)$ for $\theta_1 \neq \theta_2$ in general, it is not trivial to determine whether the solutions to (2) converge to points where $F_\theta(\theta) = 0$[1]. Even knowing that they converge to zeros of $F_\theta(\theta)$ is not helpful at all, because we have $F_\theta(\theta) = 0$ for any θ with variance zero and we are actually interested in convergence to the point $(\mathrm{x}^*, 0)$ where x^* is a local optimum of f.

Remark 1. Because of the invariance property of the natural gradient, the mean vector $m(\theta)$ and the variance $v(\theta)$ obey (6) and (7) under re-parameterization of the Gaussian distributions. Therefore, the trajectories of m and v are also independent of the parameterization. For instance, we obtain the same trajectories $v(\theta)$ for any of the following parameterizations: $\theta_{d+1} = v$, $\theta_{d+1} = \sqrt{v}$, and $\theta_{d+1} = \frac{1}{2} \ln v$, although the trajectories of the parameters θ_{d+1} are of course different. Consequently, the same convergence results for $m(\theta)$ and $v(\theta)$ (see Section 4) will hold under any parameterization. Parameterizations $\theta = (m, v)$ and $\theta = (m, \frac{1}{2} \ln v)$ correspond to the pure rank-μ update CMA-ES and the xNES with only one variance parameter. Thus, the continuous model to be analyzed encompasses both algorithms.

Remark 2. Theory of stochastic approximation says that a stochastic algorithm $\theta^{t+1} = \theta^t + \eta h^t$ follows the solution trajectories of the ODE $\frac{\mathrm{d}\theta}{\mathrm{dt}} = \mathbb{E}[h^t \mid \theta^t = \theta]$ in the limit

[1] If J_θ is not adaptive and defined to be the expectation of the objective function $f(x)$ over P_θ, convergence to the zeros of the RHS of (2) is easily obtained. For example, see Theorem 12 and its proof in [13], where the solution to the system of a similar ODE whose RHS is the vanilla gradient of the expected objective function is derived and the convergence of the solution trajectory to the critical point of the expected function is proven.

for η to zero under several conditions. In our setting, θ encodes m and v and the noisy observation $h^t = \sum_{i=1}^{\lambda} w_{R_i} \mathcal{I}_\theta^{-1} \nabla_\theta \ln p_{\theta^t}(x_i)$, where w_i, $i = 1, \ldots, \lambda$, are predefined weights and R_i is the ranking of x_i. If we define $w(p) = \sum_{i=1}^{\lambda} w_i \binom{\lambda-1}{i-1} p^{i-1}(1-p)^{\lambda-i}$ in (1), then $F_\theta(\theta) = \mathbb{E}[h^t \mid \theta^t = \theta]$ and the ODE agrees with (5). Therefore, (5) can be viewed as the limit behavior of adaptive-ES algorithms not only in the case $\eta \to 0$ and $\lambda \to \infty$ but also in the case $\eta \to 0$ and finite λ. Indeed, it is possible to bound the difference between $\{\theta^t, t \geqslant 0\}$ and the solution $\theta(\cdot)$ of the ODE (5) by extending Lemma 1 in Chapter 9 of [16].The details are omitted due to the space limitation.[2]

3 Extension of Lyapunov Stability Theorem

When convergence occurs, the variance typically converges to zero. Hence the study of the convergence of the solutions of the ODE will be carried out by analyzing the stability of the points $\theta^* = (\mathrm{x}^*, 0)$. However, because points with variance zero are excluded from the domain Θ, we need to extend classical definitions of stability to be able to handle points located on the boundary of Θ.

Definition 1 (Stability). *Consider the following system of differential equation*

$$\dot{\theta} = F(\theta), \quad \theta(0) = \theta_0 \in D, \tag{8}$$

where $F : D \mapsto \mathbb{R}^{d_\theta}$ is a continuous map and $D \subset \mathbb{R}^{d_\theta}$ is open. Then $\theta^* \in \overline{D}$ is called

- *stable in the sense of Lyapunov*[3] *if for any $\varepsilon > 0$ there is $\delta > 0$ such that $\theta_0 \in D \cap \overline{B}(\theta^*, \delta) \implies \theta(t) \in D \cap \overline{B}(\theta^*, \varepsilon)$ for all $t \geqslant 0$, where $t \mapsto \theta(t)$ is any solution of (8);*
- *locally attractive if there is $\delta > 0$ such that $\theta_0 \in D \cap \overline{B}(\theta^*, \delta) \implies \lim_{t\to\infty} \|\theta(t) - \theta^*\| = 0$ for any solution $t \mapsto \theta(t)$ of (8);*
- *globally attractive if $\lim_{t\to\infty} \|\theta(t) - \theta^*\| = 0$ for any $\theta_0 \in D$ and any solution $t \mapsto \theta(t)$ of (8);*
- *locally asymptotically stable if it is stable and locally attractive;*
- *globally asymptotically stable if it is stable and globally attractive.*

We can now understand why we need to exclude points with variance zero from the domain Θ. Indeed, points with variance zero are points from where solutions of the ODE will never move because $F_\theta(\theta) = 0$. Consequently, if we include points $(x, 0)$

[2] When $H(\theta)$ is a (natural) gradient of a function, the stochastic algorithm is called a stochastic gradient method. The theory of stochastic gradient method (e.g., [17]) relates the convergence of the stochastic algorithm with the zeros of $H(\theta)$. However, it is not applicable to our algorithm due to the reason mentioned above Remark 1.

[3] Usually, stability is defined for stationary points. However, it is not the only case that a point is stable in our definition. Let $\theta^* \in \overline{D}$ be a stable point. If $\theta^* \in D$ or F can be prolonged by continuity at θ^* as $\lim_{\theta\to\theta^*} F(\theta) = F(\theta^*)$, then $F(\theta^*) = 0$. That is, θ^* is a stationary point. However, $\lim_{\theta\to\theta^*} F(\theta)$ does not always exist for a stable boundary point $\theta^* \in \partial D$. For example, consider the ODE: $\mathrm{d}\theta_1/\mathrm{d}t = -\theta_1/\sqrt{\theta_1^2 + \theta_2^2}$, $\mathrm{d}\theta_2/\mathrm{d}t = -\theta_2$. The domain is $\mathbb{R} \times \mathbb{R}_+$. Then, $|\theta_1|$ and θ_2 are monotonically decreasing to zero. Hence, $(0, 0)$ is globally asymptotically stable. However, $\lim_{\theta\to(0,0)} F(\theta)$ does not exist.

in Θ, none of these points can be attractive as in a neighborhood we always find $\theta_0 = (x_0, 0)$ such that a solution starting in θ_0 stays there and cannot thus converge to any other point.

A standard technique to prove stability is Lyapunov's method that consists in finding a scalar function $V : \mathbb{R}^{d_\theta} \to \mathbb{R}_{\geqslant 0}$ that is positive except for a candidate stable point θ^* with $V(\theta^*) = 0$, and that is monotonically decreasing along any trajectory of the ODE. Such a function is called *Lyapunov function* (and is analogous to a potential function in dynamical systems). Lyapunov's method does not require the analysis of the solutions of the ODE. The standard Lyapunov's stability theorem gives practical conditions to verify that a function is indeed a Lyapunov function. However, because our candidate stable points are located on $\partial\Theta$, we need to extend this standard theorem.

Lemma 1 (Extended Lyapunov Stability Method). *Consider the autonomous system (8), where $F : D \to \mathbb{R}^{d_\theta}$ is a map and $D \subset \mathbb{R}^{d_\theta}$ is the open domain of θ. Let $\theta^* \in \overline{D}$ be a candidate stable point. Suppose that there is an $R > 0$ such that*
(A1)*: $F(\theta)$ is continuous on $D \cap B(\theta^*, R)$;*
(A2)*: there is a continuously differentiable $V : \mathbb{R}^{d_\theta} \to \mathbb{R}$ such that for some strictly increasing continuous function $\alpha : \mathbb{R}_+ \to \mathbb{R}_+$ satisfying $\lim_{p\to\infty} \alpha(p) = \infty$,*

$$V(\theta^*) = 0, \quad V(\theta) \geqslant \alpha(\|\theta - \theta^*\|) \quad \forall \theta \in D \cap B(\theta^*, R) \setminus \{\theta^*\} \tag{9}$$

$$\textit{and} \qquad \nabla V(\theta)^{\mathrm{T}} F(\theta) < 0 \quad \forall \theta \in D \cap B(\theta^*, R) \setminus \{\theta^*\}; \tag{10}$$

(A3)*: for any r_1 and r_2 such that $0 < r_1 \leqslant r_2 < R$, if a solution $\theta(\cdot)$ to (8) starting from $D_{r_1,r_2} = \{\theta \in D : r_1 \leqslant \|\theta - \theta^*\| \leqslant r_2\}$ stays in D_{r_1,r_2} for $t \in [0, \infty)$, then there is a $T \geqslant 0$ and a* compact *set $E \subset D_{r_1,r_2}$ such that $\theta(t) \in E$ for $t \in [T, \infty)$.*

Then, θ^* is locally asymptotically stable. If (A1) *and* (A2) *hold with D replacing $D \cap B(\theta^*, R)$ and* (A3) *holds with $R = \infty$, then θ^* is globally asymptotically stable.*

Proof. We follow the proof of Theorem 4.1 in [15]. We have from assumptions (A1) and (A2) that there is $\delta < R$ such that θ^* is stable and $V(\theta(t)) \to \tilde{V} \geqslant 0$ for each $\theta_0 \in D \cap B(\theta^*, \delta)$. Moreover, under (A1) and (A2) with D replacing $D \cap B(\theta^*, R)$ we have that $V(\theta(t)) \to \tilde{V} \geqslant 0$ for each $\theta_0 \in D$. Since $\lim_{t\to\infty} V(\theta(t)) \to 0$ implies $\lim_{t\to\infty} \|\theta - \theta^*\| = 0$ by (9), it is enough to show $\tilde{V} = 0$. We show $\tilde{V} = 0$ by contradiction argument. Assume that $\tilde{V} > 0$. Then, we have that for each $\theta_0 \in D$ (or $\in D \cap B(\theta^*, \delta)$ for the case of local asymptotic stability) there are r_1 and r_2 such that $0 < r_1 \leqslant r_2$ ($\leqslant \delta$) and $\theta(t)$ lies in D_{r_1,r_2} for $t \geqslant 0$. Note that D_{r_1,r_2} is not necessarily a compact set. This is different from Theorem 4.1 in [15]. By assumption (A3) we have that there is a compact set E and $T \geqslant 0$ such that $\theta(t) \in E$ for $t \geqslant T$. Since V is continuously differentiable and F is continuous, $\nabla V(\theta)^{\mathrm{T}} F(\theta)$ is continuous. Then, the function $\theta \mapsto V(\theta)^{\mathrm{T}} F(\theta)$ has its maximum $-\beta$ on the compact E and $-\beta < 0$ by (10). This leads to $V(\theta(t)) \leqslant V(\theta(T)) - \beta(t - T) \downarrow -\infty$ as $t \to \infty$. This contradicts the hypothesis that $V > 0$. Hence, $\tilde{V} = 0$ for any $\theta_0 \in D$ (or $\in D \cap B(\theta^*, \delta)$). □

4 Convergence of the ES-IGO-Flow

In this section we study the convergence properties of the ES-IGO-flow $\varphi : (t, \theta_0) \mapsto \theta(t)$, where $\theta(\cdot)$ represents the solution to the ES-IGO ODE (5) with initial value

$\theta(0) = \theta_0$, i.e., $\frac{\mathrm{d}\varphi(t,\theta_0)}{\mathrm{d}t} = F_\theta(\varphi(t,\theta_0))$ and $\varphi(0,\theta_0) = \theta_0$. By the definition of asymptotic stability, the global asymptotic stability of $\theta^* \in \overline{\Theta}$ implies the global convergence, that is, $\lim_{t\to\infty} \varphi(t,\theta_0) = \theta^*$ for all $\theta_0 \in \Theta$. Moreover, the local asymptotic stability of $\theta^* \in \overline{\Theta}$ implies the local convergence, that is, $\exists \delta > 0$ such that $\lim_{t\to\infty} \varphi(t,\theta_0) = \theta^*$ for all $\theta_0 \in \Theta \cap B(\theta^*, \delta)$. We will prove convergence properties of the ES-IGO-flow by applying Lemma 1. In order to prove our result we need to make the following assumption on w:

(B1): w is non-increasing and Lipschitz continuous with $w(0) > w(1)$;

(B2): $\int w(P_1[y : y \leqslant z])(z^2/d - 1/d)P_1(\mathrm{d}z) = \alpha > 0$.

Assumption (B1) is not restrictive. Indeed, the non-increasing and non-constant property of $w(\cdot)$ is a natural requirement and any weight setting in (4) can be expressed, for any given population size n, as a discretization of some Lipschitz continuous weight function. Assumption (B2) is satisfied if and only if the variance v diverges exponentially on a linear function. In fact, $F_v(\theta)$ defined in (7) reduces to $v \int w(P_1[y : y \leqslant z])(z^2/d - 1/d)P_1(\mathrm{d}z)$ when $f(x) = a^{\mathrm{T}}x$ for $\forall a \in \mathbb{R}^d \setminus \{0\}$ and we have that $\dot{v} = \alpha v$ and the solution is $v(t) = v_0 \exp(\alpha t)$. Then, $v(t) \to \infty$ as $t \to \infty$. Assumption (B2) holds, for example, if w is convex and not linear.

Let $\mathcal{G}$ be the set of strictly increasing functions $g : \mathbb{R} \to \mathbb{R}$ that are μ_{Leb}-measurable and $\mathcal{C}^2$ be the set of twice continuously differentiable functions $h : \mathbb{R}^d \to \mathbb{R}$ that are μ_{Leb}-measurable. Under (B1) and (B2), we have the following main theorems.

Theorem 1. *Suppose that the objective function f is a monotonic convex-quadratic-composite function $g \circ h$, where $g \in \mathcal{G}$ and h is a convex quadratic function $x \mapsto (x - \mathrm{x}^*)^{\mathrm{T}} A(x - \mathrm{x}^*)/2$ where A is positive definite and symmetric. Assume that* (B1) *and* (B2) *hold. Then, $\theta^* = (\mathrm{x}^*, 0) \in \overline{\Theta}$ is the globally asymptotically stable point of the ES-IGO. Hence, we have the global convergence of $\varphi(t,\theta_0)$ to θ^*.*

Proof. Since the ES-IGO does not explicitly utilize the function values but uses the quantile $P_\theta[y : f(y) \leqslant f(x)]$ which is equivalent to $P_\theta[y : g^{-1} \circ f(y) \leqslant g^{-1} \circ f(x)]$, without loss of generality we assume $f = h$.

According to Lemma 1, it is enough to show that (A1) and (A2) hold with $D(= \Theta)$ replacing $D \cap B(\theta^*, R)$ and (A3) holds with $R = \infty$. As is mentioned in the proof of Proposition 1, F_θ is locally Lipschitz continuous for a Lipschitz continuous w. Thus, (A1) is satisfied under (B1).

We can choose as a Lyapunov candidate function $V(\theta) = \sum_{i=1}^d (m_i - \mathrm{x}^*{}_i)^2 + d \cdot v = \|m - \mathrm{x}^*\|^2 + \mathrm{Tr}(vI_d)$. All the conditions on V described in (A2) are obvious except for the negativeness of $\nabla V(\theta)^{\mathrm{T}} F_\theta(\theta)$. To show the negativeness, rewrite $F_\theta(\theta)$ as $\int W_\theta^f(m + \sqrt{v}z)F_\theta(\theta, z)P_d(\mathrm{d}z)$. The idea is to show the (strictly) negative correlation between $W_\theta^f(m + \sqrt{v}z)$ and $\nabla V(\theta)^{\mathrm{T}} F_\theta(\theta, z)$ by using an extension of the result in [18, Chapter 1] and apply the inequality $\int W_\theta^f(m + \sqrt{v}z)\nabla V(\theta)^{\mathrm{T}} F_\theta(\theta, z)P_d(\mathrm{d}z) < \int W_\theta^f(m + \sqrt{v}z)P_d(\mathrm{d}z) \int \nabla V(\theta)^{\mathrm{T}} F_\theta(\theta, z)P_d(\mathrm{d}z) = 0$. We use the non-increasing property of w with $w(0) > w(1)$ in (B1) to show the negative correlation.

To prove (A3), we require (B2). Since a continuously differentiable function can be approximated by a linear function at any non-critical point $\bar{\mathrm{x}}$, the natural gradient F_θ is approximated by that on a linear function in a small neighborhood of $(\bar{\mathrm{x}}, 0)$. We use the property $\mu_{\mathrm{Leb}}[x : f(x) = \bar{f}] = 0$ to approximate F_θ. As is mentioned above,

(B2) implies F_v on a linear function is positive. By using the approximation and this property, we can show that $E = D_{r_1,r_2} \cap \{\theta : v \geqslant \bar{v}\}$ satisfies (A3) for some $\bar{v} > 0$. □

We have that for any initial condition $\theta(0) = (m_0, v_0)$, the search distribution P_θ weakly converges to the Dirac measure δ_{x^*} concentrated at the global minimum point x^*. This result is generalized to monotonic $\mathcal{C}^2$-composite functions using a quadratic Taylor approximation. However, global convergence becomes local convergence.

Theorem 2. *Suppose that the objective function f is a monotonic $\mathcal{C}^2$-composite function $g \circ h$, where $g \in \mathcal{G}$ and $h \in \mathcal{C}^2$ has the property that $\mu_{\mathrm{Leb}}[x : h(x) = s] = 0$ for any $s \in \mathbb{R}$. Assume that* (B1) *and* (B2) *hold. Let x^* be a critical point of h, i.e. $\nabla h(\mathrm{x}^*) = 0$, with a positive definite Hessian matrix A. Then, $\theta^* = (\mathrm{x}^*, 0) \in \overline{\Theta}$ is a locally asymptotically stable point of the ES-IGO. Hence, we have the local convergence of $\varphi(t, \theta_0)$ to θ^*. Moreover, if $\bar{\mathrm{x}}$ is not a critical point of $h(\cdot)$, for any $\theta_0 \in \Theta$, $\varphi(t, \theta_0)$ will never converge to $\bar{\theta} = (\bar{\mathrm{x}}, 0)$.*

Proof. As in the proof of Theorem 1, we assume $f = h$ without loss of generality. The proofs of (A1) and (A3) carry over from Theorem 1 because we only used the property $\mu_{\mathrm{Leb}}[x : f(x) = \bar{f}] = 0$. To show (A2), we use the Taylor approximation of the objective function f. Since f is approximated by a quadratic function in a neighborhood of a critical point x^*, we approximate the natural gradient by the corresponding natural gradient on the quadratic function. Then, employing the same Lyapunov candidate function as in the previous theorem we can show (A2). Because of the approximation, we only have local asymptotic stability. The last statement of Theorem 2 is an immediate consequence of the approximation of the natural gradient and (B2). □

We have that starting from a point close enough to a local minimum point x^* with a sufficiently small initial variance, the search distribution weakly converges to δ_{x^*}. It is not guaranteed for the parameter to converge somewhere when the initial mean is not close enough to the local optimum or the initial variance is not small enough. Theorem 2 also states that the convergence $(m(t), v(t)) \to (\bar{\mathrm{x}}, 0)$ does not happen for $\bar{\mathrm{x}}$ such that $\nabla h(\bar{\mathrm{x}}) \neq 0$. That is, the continuous time ES-IGO does not prematurely converge on a slope of the landscape of f.

5 Conclusion

In this paper we have proven the local convergence of the continuous time model associated to step-size adaptive ESs towards local minima on monotonic $\mathcal{C}^2$-composite functions. In the case of monotonic convex-quadratic-composite functions we have proven the global convergence, i.e. convergence independently of the initial condition (provided the initial step-size is strictly positive) towards the unique minimum. Our analysis relies on investigating the stability of critical points associated to the underlying ODE that follows from the Information Geometric Optimization setting. We use a classical method for the analysis of stability of critical points, based on Lyapunov functions. We have however extended the method to be able to handle convergence towards solutions at the boundary of the ODE definition domain. We believe that our approach is general

enough to handle more difficult cases like the CMA-ES with a more general covariance matrix. We want to emphasize that the model we have analyzed is the correct model for step-size *adaptive* ESs as the ODE encodes both the mean vector *and* step-size and preserves fundamental invariance properties of the algorithm.

Acknowledgments. This work was partially supported by the ANR-2010-COSI-002 grant (SIMINOLE) of the French National Research Agency and the ANR COSINUS project ANR-08-COSI-007-12.

References

1. Auger, A.: Convergence results for the (1, λ)-SA-ES using the theory of φ-irreducible Markov chains. Theoretical Computer Science 334(1-3), 35–69 (2005)
2. Jägersküpper, J.: Probabilistic runtime analysis of $(1+, \lambda)$, ES using isotropic mutations. In: Proceedings of the 2006 Genetic and Evolutionary Computation Conference, GECCO 2006, pp. 461–468. ACM (2006)
3. Jägersküpper, J.: How the $(1 + 1)$ ES using isotropic mutations minimizes positive definite quadratic forms. Theoretical Computer Science 361(1), 38–56 (2006)
4. Jägersküpper, J.: Algorithmic analysis of a basic evolutionary algorithm for continuous optimization. Theoretical Computer Science 379(3), 329–347 (2007)
5. Arnold, L., Auger, A., Hansen, N., Ollivier, Y.: Information-geometric optimization algorithms: a unifying picture via invariance principles. arXiv:1106.3708v1 (2011)
6. Hansen, N., Muller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation 11(1), 1–18 (2003)
7. Baluja, S., Caruana, R.: Removing the genetics from the standard genetic algorithm. In: Proceedings of the 12th International Conference on Machine Learning (1995)
8. Ostermeier, A., Gawelczyk, A., Hansen, N.: A derandomized approach to self-adaptation of evolution strategies. Evolutionary Computation 2(4), 369–380 (1994)
9. Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., Schmidhuber, J.: Exponential natural evolution strategies. In: Proceedings of Genetic and Evolutionary Computation Conference, pp. 393–400. ACM (2010)
10. Yin, G.G., Rudolph, G., Schwefel, H.P.: Establishing connections between evolutionary algorithms and stochastic approximation. Informatica 1, 93–116 (1995)
11. Yin, G.G., Rudolph, G., Schwefel, H.P.: Analyzing the (1, λ) evolution strategy via stochastic approximation methods. Evolutionary Computation 3(4), 473–489 (1996)
12. Kushner, H.J., Yin, G.G.: Stochastic approximation and recursive algorithms and applications, 2nd edn. Springer (2003)
13. Malagò, L., Matteucci, M., Pistone, G.: Towards the geometry of estimation of distribution algorithms based on the exponential family. In: Proceedings of Foundations of Genetic Algorithms (FOGA 2011), pp. 230–242. ACM (2011)
14. Akimoto, Y., Nagata, Y., Ono, I., Kobayashi, S.: Theoretical foundation for CMA-ES from information geometry perspective. Algorithmica, Online First (2011)
15. Khalil, H.K.: Nonlinear systems. Prentice-Hall, Inc. (2002)
16. Borkar, V.S.: Stochastic approximation: a dynamical systems viewpoint. Cambridge University Press (2008)
17. Bonnabel, S.: Stochastic gradient descent on Riemannian manifolds. arXiv:1111.5280v2 (2011)
18. Thorisson, H.: Coupling, stationarity, and regeneration. Springer (2000)

A Parameterized Runtime Analysis of Simple Evolutionary Algorithms for Makespan Scheduling

Andrew M. Sutton and Frank Neumann

School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia

Abstract. We consider simple multi-start evolutionary algorithms applied to the classical NP-hard combinatorial optimization problem of Makespan Scheduling on two machines. We study the dependence of the runtime of this type of algorithm on three different key hardness parameters. By doing this, we provide further structural insights into the behavior of evolutionary algorithms for this classical problem.

1 Introduction

Evolutionary algorithms and other types of bio-inspired computing techniques have been extensively used for a wide range of combinatorial optimization problems. Understanding the behavior of evolutionary algorithms on NP-hard combinatorial optimization problems from a theoretical point of view is still a challenging task. Results on the runtime of evolutionary algorithms for different combinatorial optimization problems have been obtained during the last ten years. We refer the interested reader to the textbook of Neumann and Witt [10] for an overview on this area of research. One of the first runtime analyses of evolutionary algorithms for NP-hard combinatorial optimization problems has been carried out by Witt [14] by considering the Makespan Scheduling problem. Witt has studied the approximation and average-case behavior of simple evolutionary algorithms for this problem. Gunia [7] later extended this work to the case of multiple machines. Other recent works have studied the multiple machine case in terms of convergence to solutions corresponding to Nash equilibria in multiplayer non-cooperative games [4,6].

We consider the analysis of evolutionary algorithms in the context of fixed-parameter tractability by expressing their runtime as a function of both problem size and an additional hardness parameter that attempts to isolate the exponential complexity of the instance. This approach, which is widely used in the classical analysis of algorithms and problem hardness [3], has recently been introduced into the analysis of evolutionary algorithms. It facilitates the understanding of which features in an instance of a given problem makes the problem hard to solve. Parameterized runtime results have been obtained in the context of evolutionary computation for the vertex cover problem [9], the computation of maximum leaf spanning trees [8], the MAX-2-SAT problem [12], and the Euclidean TSP [13].

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 52–61, 2012.

Our goal is to provide further insights into the optimization process of evolutionary algorithms for the Makespan Scheduling problem by carrying out parameterized runtime analyses. We show that multi-start variants of two simple evolutionary algorithms are fixed-parameter evolutionary algorithms for a parameterization of Makespan Scheduling that takes into account the value of the optimal schedule above its theoretical lower bound. We then study their runtime in dependence of the critical path size of an optimal schedule. Finally, we investigate a parameterization that considers the machine load discrepancy in an optimal schedule. We show that, with a minor modification to the mutation procedure, the resulting multi-start variant of RLS is a Monte Carlo fixed-parameter tractable algorithm for Makespan Scheduling. This indicates that instances with large discrepancies will be easier to solve by randomized local search.

2 Preliminaries

We investigate the classical NP-hard Makespan Scheduling problem on two identical machines. In this problem, we have a set of n jobs where each job j requires a nonzero integral processing time p_j on either machine. We define the *load* of a machine to be the sum of processing times of the jobs that are assigned to it. The makespan of a schedule is the maximum load over both machines. The objective is to find an assignment that minimizes the makespan.

An arbitrary schedule can be represented as a binary length-n decision vector where the j-th component specifies to which machine job j is assigned in a schedule. For a given instance of Makespan Scheduling, the makespan of a schedule corresponding to a binary decision vector $x \in \{0,1\}^n$ is captured by the pseudo-Boolean function

$$f : \{0,1\}^n \to \mathbb{N} := x \mapsto \max \left\{ \sum_{j=1}^{n} x_j p_j, \sum_{j=1}^{n} (1 - x_j) p_j \right\}.$$

We will denote $P = \sum_{j=1}^{n} p_j$. Thus $P/2 \leq f(x) \leq P$. Without loss of generality, we will assume that the processing times are sorted in nonincreasing order, i.e., $p_1 \geq \cdots \geq p_n$. We denote $f^* = \min_{x \in \{0,1\}^n} f(x)$ as the value of the optimal makespan for an instance.

We will carry out parameterized runtime analyses of evolutionary algorithms for Makespan Scheduling. Let Σ be a finite alphabet. A *parameterized problem* over Σ is a pair (L, κ) where $L \subseteq \Sigma^*$ is a language over Σ and $\kappa : \Sigma^* \to \mathbb{N}$ is a map called a *parameterization* of Σ. Letting $n = |x|$ and $k = \kappa(x)$, a parameterized problem (L, κ) is *fixed-parameter tractable* if there is an algorithm that decides $x \in L$ in time bounded by $g(k) \cdot poly(n)$ where g is an arbitrary recursive function that depends only on k. We call such an algorithm an fpt-algorithm. The class of parameterized problems (L, κ) that can be decided by an fpt-algorithm is called FPT. A *Monte Carlo fpt-algorithm* for (L, κ) is a randomized fpt-algorithm with runtime bounded by $g(k) \cdot poly(n)$ that will accept an input $x \in \Sigma^*$ with probability at least $1/2$ if $x \in L$, otherwise it accepts with

probability zero. An XP-algorithm for a parameterized problem (L, κ) is an algorithm that runs in worst-case time $n^{g(k)}$. We define a *Monte Carlo* XP*-algorithm* analogously.

We consider two classical mutation-only evolutionary algorithms, the (1+1) EA and RLS. In particular, we will analyze repeating runs of length $\ell(n) = O(poly(n))$ and take the best solution found during any run. A run of length $\ell(n)$ for the (1+1) EA and RLS is explicitly defined in Algorithms 1 and 2, respectively. In each case, we will investigate the probability that a single run solves the parameterized problem. Observing that each run is an independent Bernoulli trial, it will be straightforward to bound the failure probability after a prescribed number of runs. We will make use of the following technical lemma.

Lemma 1. *Let h be a positive function. The probability that an arbitrary (but nonempty) set of $k < n$ bits is never changed during a run of length $\ell(n) = n \cdot h(n)$ is bounded by $\Omega\big(e^{-k \cdot h(n)}\big)$ for the (1+1) EA, and $\Omega\big(e^{-(k \log k) \cdot h(n)}\big)$ for RLS.*

Proof. For the (1+1) EA, the probability that none of the specified k bits are mutated during a single iteration is $(1 - 1/n)^k$. After $\ell(n)$ iterations, the probability is $(1-1/n)^{kn \cdot h(n)} = \Omega(e^{-k \cdot h(n)})$. For RLS, the probability that none of the specified k bits are changed in a single iteration is $(1 - k/n)$. Here, we must also consider the rate k grows as a function of n. If $k = o(n)$, the bound is obviously the same for the (1+1) EA, otherwise, $k = \Theta(n)$. In the case that $c_1 n \leq k \leq c_2 n$ for some constants $0 < c_1 \leq c_2 < 1$, then we have $(1-k/n)^n \geq (1-c_2)^{k/c_1} = e^{-k\epsilon}$ where ϵ is a positive constant. Finally, in the case that $k \sim n$, since k is at most $n-1$, it must hold that $(1-k/n)^n \geq e^{-n \log n}$ and since in this case $n = (1+o(1))k$, the asymptotic bound holds. □

Algorithm 1. A single run of the (1+1) EA

```
input  : A run length ℓ(n)
output: A candidate decision vector x
Choose x uniformly at random from {0,1}^n;
for i ← 1 to ℓ(n) do
    x' ← x;
    Flip each bit of x' independently with probability 1/n;
    if f(x') ≤ f(x) then x ← x'
end
```

3 Parameterized Analysis for Optimal Makespan Value

The *standard parameterization* of a combinatorial optimization problem is (assuming minimization), given an instance and a parameter k, is the value of the optimal solution *at most* k? Fernau [5] has shown that the standard parameterization of Makespan Scheduling[1] is fixed-parameter tractable. The proof relies on the following straightforward kernelization technique. If $k < P/2$, the

[1] Makespan Scheduling is referred to as Minimum Partition in Fernau's work.

Algorithm 2. A single run of RLS

input : A run length $\ell(n)$
output: A candidate decision vector x
Choose x uniformly at random from $\{0,1\}^n$;
for $i \leftarrow 1$ **to** $\ell(n)$ **do**
  $x' \leftarrow x$;
  Choose j uniformly at random from $\{1,\ldots,n\}$;
  $x'_j \leftarrow (1 - x'_j)$;
  if $f(x') \leq f(x)$ **then** $x \leftarrow x'$
end

answer is always "no" since clearly f is bounded below by $P/2$. On the other hand, if $k \geq P/2$, it follows that $2k \geq P \geq n$, the rightmost inequality coming from the fact that the processing times are positive integers. Hence there are at most 2^{2k} schedules which can be search exhaustively in time bounded by $O(4^k)$.

To provide stronger insights into the difficulty of Makespan Scheduling as a function of the value of the optimal makespan, we will consider a more detailed parameterization that captures the difference between the makespan of an optimal schedule and the theoretical lower bound. In particular, we show that if the optimal schedule has a makespan much larger than $P/2+P/n$, the problem is easier to solve by the (1+1) EA and RLS using a multi-start approach. We show that the multi-start variants of both the (1+1) EA and RLS are Monte Carlo fpt-algorithms for Makespan Scheduling by showing they are capable of simulating a polynomial-time approximation scheme (PTAS).

We will hereafter assume that $p_1 \leq P/2$, otherwise it is easy to show that RLS always runs in expected polynomial time simply by collecting all the smaller jobs onto the other machine. A move could result in an improving solution if it shifts a job from the fuller machine to the emptier machine. We follow Witt [14] and define the *critical job size* $s(x)$ with respect to a decision vector x as the processing time of the smallest job on the fuller machine. If $f(x) > (P + s(x))/2$, then it is possible to construct an improving schedule by moving at least one job from the fuller machine to the emptier machine. The optimal solution parameterization is, given an instance of Makespan Scheduling and an integer k, is $f^* \leq P/2 + P/k$?

Lemma 2 (due to Witt [14]). *Let x be the current search point. Suppose the critical job size is bounded above by s^* for all following search points of value greater than $L + s^*/2$ where $L \geq P/2$. Then for any $\gamma > 1$ and $0 < \delta < 1$, both the (1+1) EA and RLS can compute a decision vector with makespan at most $L + s^*/2 + \delta P/2$ in at most $\lceil en\ln(\gamma/\delta)\rceil$ steps with probability at least $1 - \gamma^{-1}$.*

Lemma 3. *Given some $1 \leq k \leq n$, let x' be a decision vector such that the contribution of jobs $1,\ldots,k$ is minimal. The probability that after a run of length $\lceil en\ln(2k)\rceil$ the (1+1) EA or RLS has discovered a schedule with makespan at most $P/2 + P/k$ is bounded below by $\Omega(e^{-k\lceil e\ln(2k)\rceil})$ for the (1+1) EA, and $\Omega(e^{-(k\log k)\lceil e\ln(2k)\rceil})$ for RLS.*

Proof. As long as no move involves the first k bits, the critical job size s^* is bounded above by p_k. Furthermore, since $kp_k \leq p_1 + \cdots + p_k \leq P$, it follows that p_k is at most P/k. By Lemma 2, by setting L to $P/2$, s^* to P/k, $\gamma = 2$, and δ to $1/k$, the probability that we reach a solution $\hat{x}$ where

$$f(\hat{x}) \leq L + P/(2k) + (1/k)(P/2) = P/2 + P/k$$

in $\lceil en\ln(2k)\rceil$ steps is at least $1/2$, as long as none of the first k jobs are moved.

Thus, if q denotes the probability that none of the first k bits are mutated during a run of length $\lceil en\ln(2k)\rceil$, then the solution is reached with probability at least $q/2$. The proof is completed by appealing to Lemma 1 for the lower bound on q and using the fact that $\lceil en\ln(2k)\rceil \leq n\lceil e\ln(2k)\rceil$.

Theorem 1. *The multi-start (1+1) EA (RLS) using runs of length $\ell(n) = \lceil en\ln(2k)\rceil$ is a Monte Carlo fpt-algorithm for the optimal makespan parameterization of* Makespan Scheduling.

Proof. Consider an arbitrary instance of Makespan Scheduling. If $f^* > P/2 + P/k$ the proof is complete since the output of the algorithm in this case is irrelevant. Thus we suppose that $f^* \leq P/2 + P/k$.
The probability that a random initial solution to any run contains the first k jobs properly fixed is at least 2^{-k+1}. Given such a solution, let $q(n)$ denote the probability that, after a run of length $\lceil en\ln(2k)\rceil$, the algorithm has found a schedule $\hat{x}$ where $f(\hat{x}) \leq P/2 + P/k$. The probability that t consecutive runs are all unsuccessful is at most $(1 - q(n)/2^{k-1})^t$. Setting $t = \lceil 2^{k-1}q(n)^{-1}\rceil$ gives a failure probability of at most $1/e$. Since each run consists of $O(n\log k)$ evaluations, the total runtime is $O(tn\log k)$. Due to Lemma 3, $q(n)$ is bounded by a function depending only on k for both the (1+1) EA and RLS. Thus by setting $g(k) = 2^{k-1}q(n)^{-1}$, the total runtime is bounded by $O(g(k)\cdot n\log k)$ and the success probability is at least $1 - 1/e > 1/2$.

4 Parameterized Analysis for Critical Path Size

In general machine scheduling problems, the *critical path* of a schedule is a set of consecutive jobs in which the first job starts at time zero, the completion time of the last job is the makespan of the schedule, and the completion time of each job is equal to the starting time of the next [11]. For the two-machine Makespan Scheduling problem, we define the critical path of a schedule as the set of jobs scheduled on the fuller machine. Formally, the critical path of a schedule x is the set $\mathcal{C}(x) \subseteq [n]$ such that for all $i, j, \in \mathcal{C}(x)$, $x_i = x_j$ and $\sum_{i\in\mathcal{C}(x)} p_i = f(x)$. In the ambiguous case (when the machines balance) we define the critical path as the smallest such set with ties in cardinality broken arbitrarily. We define the *critical path size* of a schedule x as $|\mathcal{C}(x)|$. The *critical path size parameterization* of Makespan Scheduling is, given an integer k, is there a schedule with critical path size at most k?

Lemma 4. *Consider an instance of* Makespan Scheduling *such that there exists a schedule z with $|\mathcal{C}(z)| \leq k$. Suppose x' corresponds to a schedule such that for all $i, j \in \mathcal{C}(z)$, $x'_i = x'_j$. We call a run of the (1+1) EA (RLS) a* success *if it discovers a schedule with critical path size at most k. Then starting with x' as the initial decision vector, for any constant $c > 1$, the success probability of a run of the (1+1) EA of length $\lceil cen\,(\ln n + \ln p_1 + 1)\rceil$ is bounded by $\Omega\big((enp_1)^{-cek}\big)$. Moreover, the success probability of a run of RLS of length $\lceil cn\,(\ln n + 1)\rceil$ starting from x' is bounded by $\Omega\big((en)^{-ck\log k}\big)$.*

Proof. Without loss of generality, suppose that for all $i, j \in \mathcal{C}(z)$, $x'_i = x'_j = 0$. If, for any $\ell \in [n]$, $x'_\ell = 0 \implies \ell \in \mathcal{C}(z)$, then the proof is complete. Otherwise, machine zero (i.e., the machine that corresponds to a zero bit in the decision vector) obviously must have the highest load since it contains every job in $\mathcal{C}(z)$. Let $S(x) = \{i : i \notin \mathcal{C}(z) \wedge x_i = 0\}$ be the set of jobs on machine zero that do not belong to $\mathcal{C}(z)$.
During a run of either the (1+1) EA or RLS, as long as none of the jobs in $\mathcal{C}(z)$ are not moved off machine zero, any move that reduces the number of jobs not in $\mathcal{C}(z)$ on machine zero is accepted. Furthermore, as long as the jobs in $\mathcal{C}(z)$ remain on machine zero, its load is at least the load of machine one. Thus, no moves which increase the number of jobs on machine zero are accepted.

For the (1+1) EA, let $d(x) = \sum_{i \in S(x)} p_i$. Suppose that no jobs from $\mathcal{C}(z)$ are moved off machine zero during a run of the (1+1) EA. In this case, any mutation involving an element of $S(x)$ is accepted and decreases the makespan (and the d value) by its processing time. Such a move occurs with probability at least $1/(en)$. By the multiplicative drift theorem [2], the expected number of steps until the d value has reached zero conditioned on the event that no bits corresponding to $\mathcal{C}(z)$ are flipped is at most $en\,(1 + \ln d(x')) \leq en\,(\ln n + \ln p_1 + 1)$ since $d(x') \leq np_1$. Consider a run of the (1+1) EA of length $t = cen(\ln n + \ln p_1 + 1)$. By the Markov inequality, the success probability of such a run conditioned on the event that no bit in $\mathcal{C}(z)$ is flipped is at least $1 - 1/c = \Omega(1)$. Hence the bound on the success probability is $\Omega\big((enp_1)^{-cek}\big)$ by Lemma 1.

For RLS, we set the run length to $\ell(n) = \lceil cn(\ln n + 1)\rceil$. The probability that RLS takes fewer than $\ell(n)$ steps to move the remaining $|S(x)|$ jobs conditioned on the event that no jobs in $\mathcal{C}(z)$ are moved is at least $1 - (ne)^{-c+1} = 1 - o(1)$. This result comes from the classical coupon collector analysis (see Theorem 1.23 in Chapter 1 of [1]). The bound on the success probability of such a run of RLS then follows directly from Lemma 1. □

Theorem 2. *For any constant $c > 1$, a multi-start (1+1) EA procedure using a run length of $\ell(n) = \lceil cen(\ln n + \ln p_1 + 1)\rceil$ solves the critical path size parameterization in at most $O\big(2^k (enp_1)^{cek} \cdot n(\log n + \log p_1)\big)$ evaluations with probability at least $1/2$. Moreover, a multi-start RLS procedure using a run length of $\ell(n) = \lceil cn\,(\ln n + 1)\rceil$ solves the critical path size parameterization in at most $O\big(2^k (en)^{ck\log k} \cdot n \log n\big)$ evaluations with probability at least $1/2$.*

Proof. Consider an arbitrary instance of Makespan Scheduling. If there is no schedule z such that $|\mathcal{C}(z)| \leq k$, the proof is complete. Otherwise, suppose there exists such a schedule.
With probability at least $2 \cdot 2^{-k}$, the initial schedule x' of a run of the (1+1) EA (RLS) has $x'_i = x'_j$ for all $i, j \in \mathcal{C}(z)$. Let $q(n)$ denote the probability that a (1+1) EA run of length $\lceil cen(\ln n + \ln p_1 + 1) \rceil$ starting from x' generates a schedule with critical path size at most k. By Lemma 4, $q(n) = \Omega\big((enp_1)^{-cek}\big)$.

The probability that t consecutive runs of the required size of the (1+1) EA all fail to find such a schedule is at most $(1 - q(n)/2^{k-1})^t$. Hence, after $2^{k-1}q(n)^{-1} = O\big(2^k(enp_1)^{cek}\big)$ such runs of the (1+1) EA, the failure probability is at most $1/e$ and the parameterization is solved with probability $1 - 1/e > 1/2$. Since each run of the (1+1) EA costs $O(n(\log n + \log p_1))$ evaluations, we have the claimed runtime. The proof for RLS is analogous. □

It immediately follows from Theorem 2 that the multi-start RLS is a Monte Carlo XP-algorithm for the critical path size parameterization of Makespan Scheduling. We must, however, be slightly more careful in the case of the multi-start (1+1) EA since p_1 can be exponential in n. In this case, it follows that the multi-start (1+1) EA is a Monte Carlo XP-algorithm for inputs where all processing times are polynomially bounded in n.

5 A Monte Carlo fpt-Algorithm for Discrepancy

Following the terminology of Witt [14] we define the absolute difference in load across machines the *discrepancy* of a schedule, i.e., $\Delta(x) = 2f(x) - P$. Denoting as $\Delta^* = 2f^* - P$ the discrepancy of the optimal solution of an instance, we consider the following parameterized problem (for notational convenience, we set $p_{n+1} = 0$). Given an instance of Makespan Scheduling and an integer k, is $p_k \geq \Delta^* \geq p_{k+1}$?

We will consider two evolutionary algorithms, called k-biased (1+1) EA and k-biased-RLS which differ from the (1+1) EA and RLS by using a slightly modified mutation operator. We then consider the efficiency of these variants for solving the discrepancy parameterization. For the k-biased (1+1) EA, the mutation step in line 4 of Algorithm 1 is replaced with the following lines of code.

for $j \leftarrow 1$ **to** k **do** flip x'_j with probability $1/(kn)$;
for $j \leftarrow k+1$ **to** n **do** flip x'_j with probability $1/n$;

For the k-biased-RLS, the mutation step in lines 4 and 5 of Algorithm 2 are replaced with the following lines of code.

if $r < 1/n$ **then** choose j uniformly at random from $\{1, \ldots, k\}$;
else choose j uniformly at random from $\{k+1, \ldots, n\}$;
$x'_j \leftarrow (1 - x'_j)$;

These biased mutation operators have a smaller probability of flipping the bits on the first k positions compared to the ones presented in Section 2.

Lemma 5. *Let h be a positive function. The probability that the k-biased (1+1) EA (k-biased-RLS) does not change the first k bits during a run of length $\ell(n) = n \cdot h(n)$ is bounded by $\Omega(e^{-h(n)})$.*

Proof. For the k-biased (1+1) EA, the probability that none of k bits are selected for mutation in a single step is $(1 - 1/(kn))^k$. After $\ell(n)$ steps the probability that none of the first k bits have changed is at least $(1 - 1/(kn))^{kn \cdot h(n)}$. For k-biased-RLS, the probability that any of the first k bits are selected for mutation is $1/n$. After $\ell(n)$ steps, the first k bits have not changed with probability at least $(1-1/n)^{n \cdot h(n)}$. In both cases, the asymptotic bound follows from $(1-1/x)^{x \cdot f(x)} = \Omega(e^{-f(x)})$. □

Lemma 6. *Let k be such that $p_{k+1} \leq \Delta^*$ where $p_{n+1} = 0$. Let x' be a decision vector such that the contribution of jobs $1, \ldots, k$ to the makespan is minimal. We call a run of the k-biased (1+1) EA (k-biased-RLS) a* success *if it discovers an optimal schedule. Then starting with x' as the initial decision vector, the success probability for a run of the k-biased (1+1) EA of length $2en(\ln n + \ln p_1 + 1)$ is bounded below by $\Omega\big((np_1)^{-2e}\big)$. Moreover, the success probability for a run of k-biased-RLS of length $2n(\ln n + 1)$ is bounded below by $\Omega\big(n^{-2}\big)$.*

Proof. We assume $\Delta(x') > \Delta^* \geq 0$ since otherwise x' is already optimal. In this case there is a machine with a higher load. Let $S = \{k+1, k+2, \ldots, n\}$. We first show that as long as x' is not optimal and there are jobs from S on the fuller machine, moving any such job to the emptier machine results in a strictly improving move. Suppose not. Then there is a job $j > k$ on the fuller machine and $\Delta(x') \leq p_j$, otherwise moving p_j results in an improvement. But by definition, we have $p_j \leq \Delta^*$ which contradicts the non-optimality of x'. It follows that if the first k jobs already contribute minimally to the makespan, as long as no mutation involves the first k bits, the optimal schedule can be found by moving all jobs from S on to the emptier machine.

For the k-biased (1+1) EA, let $d(x) = \Delta(x) - \Delta^*$. The probability that a mutation removes a job in S from the fuller machine is at least

$$(1 - 1/(kn))^k \, (1 - 1/n)^{n-k-|S|} \, |S|^{-1} \geq (1 - 1/n)^{n-|S|} \, |S|^{-1} \geq 1/(en).$$

The expected time until the d value has reduced to zero conditioned on the event that no bits of index at most k are flipped follows from the multiplicative drift theorem of Doerr et al. [2] and is at most $t = en\,(1 + \ln d(x'))$. By the Markov inequality, the probability that this occurs after $2t$ steps (again, conditioned on the event that no bits with index at most k are flipped) is at least $1/2 = \Omega(1)$. The bound on the success probability of a run of length $2t$ follows from Lemma 5.

For k-biased-RLS, suppose there are i jobs from S on the fuller machine. The probability that k-biased-RLS moves one of these jobs to the emptier machine is at least

$$(1 - 1/n) \cdot i/(n-k) = \frac{n-1}{n} \cdot \frac{i}{n-k}$$

for $n > 1$. The expectation until all jobs from S are moved off the fuller machine conditioned on the event that no jobs in $[n] \setminus S$ are moved is at most

$$\frac{n}{n-1} \cdot (n-k) \sum_{i=1}^{n-k} 1/i \leq n(\ln n + 1)$$

since $k \geq 1$. By the Markov inequality, the probability that this occurs in a run of $2n(\ln n + 1)$ steps is at least $1/2 = \Omega(1)$. The final bound on the success probability comes from Lemma 5. □

We now prove that the k-biased (1+1) EA (on inputs with polynomially bounded processing times) and k-biased-RLS (for general processing times) are Monte Carlo fpt-algorithms for this parameterization. At this point, it might be tempting to assume that we require instance-specific knowledge in order to choose the appropriate value for k. Instead, we are interested in the following question. For a given and fixed k, is there a class of Makespan Scheduling instances for which k-biased-RLS and the k-biased (1+1) EA are efficient? We now prove that such a class must include instances where $p_k \geq \Delta^* \geq p_{k+1}$.

Theorem 3. *A multi-start k-biased-RLS procedure that uses a run length of $\ell(n) = \lceil 2n(\ln n + 1) \rceil$ is a Monte Carlo fpt-algorithm for the discrepancy parameterization of* Makespan Scheduling. *In particular, if the instance is a yes instance (that is, $p_k \geq \Delta^* \geq p_{k+1}$), it solves the problem after $O(2^k n^3 \log n)$ steps with probability $1 - 1/e$.*
Similarly, the multi-start (1+1) EA is a Monte Carlo fpt-algorithm for the discrepancy parameterization for inputs where the processing times are polynomially bounded in n.

Proof. Consider an arbitrary instance of Makespan Scheduling. If it is not the case that $p_k \geq \Delta^* \geq p_{k+1}$, the proof is complete since, in this case, the output of the algorithm is arbitrary. Thus we can assume the bounds on Δ^*. A single run of k-biased-RLS starts with the first k jobs contributing minimally to the makespan with probability at least 2^{-k+1}. Let $q(n)$ denote the probability that a k-biased-RLS run of length $\lceil 2n(\ln n + 1) \rceil$ is successful. The failure probability for t consecutive runs is at most $(1 - q(n)/2^{k-1})^t$. Setting $t = \lceil 2^{k-1} q(n)^{-1} \rceil$ gives a failure probability of at most $1/e$. By Lemma 6, $q(n) = \Omega(n^{-2})$. Thus, the probability that the algorithm solves the discrepancy parameterization of Makespan Scheduling in $t = O(2^k n^2)$ runs of length $O(n \log n)$ evaluations each is at least $1 - 1/e > 1/2$.
The proof for the multi-start k-biased (1+1) EA is identical, except we set $\ell(n) = \lceil 2en(\ln n + \ln p_1 + 1) \rceil$ and apply Lemma 6 to get $q(n) = \Omega\big((np_1)^{-2e}\big)$. Thus after $O\big(2^k (np_1)^{2e} n(\log n + \log p_1)\big)$ steps, the algorithm has solved the discrepancy parameterization with probability at least $1 - 1/e$.

6 Conclusion

The parameterized analysis of evolutionary algorithms allows for a deeper understanding of which structural parameters of an instance of a combinatorial

optimization problem makes it easy or hard to solve. With this paper, we have contributed to the parameterized runtime analysis of evolutionary algorithms. We studied the Makespan Scheduling problem previously analyzed by Witt from a worst case and average case perspective. Our results provide further insights into the behaviour of evolutionary algorithms for this classical problem. We have shown that multi-start variants of the (1+1) EA and RLS are Monte Carlo fpt-algorithms for a parameterization which considers the value of the optimal solution above its lower bound. We have performed a runtime analysis in dependence of the critical path size of an optimal solution, and shown that a multi-start variant of RLS is a Monte Carlo fpt-algorithm for a parameterization that considers the discrepancy in load across machines.

References

1. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific Publishing Company (2011)
2. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. In: Pelikan, M., Branke, J. (eds.) GECCO, pp. 1449–1456. ACM (2010)
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
4. Even-Dar, E., Kesselman, A., Mansour, Y.: Convergence time to Nash equilibrium in load balancing. ACM Transactions on Algorithms 3(3) (2007)
5. Fernau, H.: Parameterized Algorithmics: A Graph Theoretic Approach. Habilitationsschrift (English), Universität Tübingen (2005)
6. Goldberg, P.W.: Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game. In: Chaudhuri, S., Kutten, S. (eds.) PODC, pp. 131–140. ACM (2004)
7. Gunia, C.: On the analysis of the approximation capability of simple evolutionary algorithms for scheduling problems. In: Beyer, H.G., O'Reilly, U.M. (eds.) GECCO, pp. 571–578. ACM (2005)
8. Kratsch, S., Lehre, P.K., Neumann, F., Oliveto, P.S.: Fixed Parameter Evolutionary Algorithms and Maximum Leaf Spanning Trees: A Matter of Mutation. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part I. LNCS, vol. 6238, pp. 204–213. Springer, Heidelberg (2010)
9. Kratsch, S., Neumann, F.: Fixed-parameter evolutionary algorithms and the vertex cover problem. In: Rothlauf, F. (ed.) GECCO, pp. 293–300. ACM (2009)
10. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity. Springer (2010)
11. Pinedo, M.: Scheduling: theory, algorithms, and systems. Springer (2012)
12. Sutton, A.M., Day, J., Neumann, F.: A parameterized runtime analysis of evolutionary algorithms for MAX-2-SAT. In: GECCO. ACM (to appear, 2012)
13. Sutton, A.M., Neumann, F.: A parameterized runtime analysis of evolutionary algorithms for the Euclidean traveling salesperson problem. In: AAAI. AAAI Press (to appear, 2012)
14. Witt, C.: Worst-Case and Average-Case Approximations by Simple Randomized Search Heuristics. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 44–56. Springer, Heidelberg (2005)

On Algorithm-Dependent Boundary Case Identification for Problem Classes*

Chao Qian, Yang Yu, and Zhi-Hua Zhou

National Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210046, China
{qianc,yuy,zhouzh}@lamda.nju.edu.cn

Abstract. Running time analysis of metaheuristic search algorithms has attracted a lot of attention. When studying a metaheuristic algorithm over a problem class, a natural question is what are the easiest and the hardest cases of the problem class. The answer can be helpful for simplifying the analysis of an algorithm over a problem class as well as understanding the strength and weakness of an algorithm. This algorithm-dependent boundary case identification problem is investigated in this paper. We derive a general theorem for the identification, and apply it to a case that the (1+1)-EA with mutation probability less than 0.5 is used over the problem class of pseudo-Boolean functions with a unique global optimum.

1 Introduction

Metaheuristic search algorithms such as simulated annealing (SA) [11], particle swarm optimization (PSO) [10], evolutionary algorithms (EA) [2], etc., have been widely and successfully applied to real-world optimization problems. An advantage of metaheuristic algorithms is their problem independence, i.e., they can be applied to a very large range of optimization problems. A natural theoretical question is, therefore, how well the metaheuristic algorithms perform on classes of problems.

A commonly used quality measure of a metaheuristic algorithm is its expected running time, i.e., the expected number of steps that it takes to find an optimum. Several approaches, e.g., drift analysis [7] and convergence-based approach [17], have been developed for running time analysis of metaheuristic algorithms. The running time of several metaheuristic algorithms has been studied on some simple pseudo-Boolean problems, e.g., [4,7,5], and later, on some combinatorial optimization problems, e.g., [1,12]. In most of these studies, the analysis was over restricted problem classes where the problem cases have similar structures. While, for large problem classes, a large variety of structures of problem cases can obstruct the analysis.

One possible way to simplify the analysis over a problem class is to characterize the class by its easiest and hardest cases, and then analyze on these boundary cases. By the well-known no free lunch theorem [14], we know that no single problem is intrinsically harder than another, until an algorithm is involved into the consideration. Therefore,

* This research was supported by the National Science Foundation of China (60903103, 61105043)

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 62–71, 2012.

this paper studies the algorithm-dependent boundary case identification problem. Given a metaheuristic algorithm, the identification of the boundary cases of a problem class can not only help to study the performance of the algorithm over the problem class, but also provide concrete cases to reveal the strength and weakness of the metaheuristic algorithm.

For this purpose, we derive a general theorem for algorithm-dependent boundary case identification, which gives a sufficient condition for identifying the easiest and hardest cases of a problem class for an algorithm. We then prove that in the pseudo-Boolean function class with a unique global optimum, the OneMax and the Trap problem are the easiest and the hardest case for the (1+1)-EA with mutation probability less than 0.5, respectively.

There are a few previous studies concerning problem classes. The running time of EAs on linear pseudo-Boolean function class, which is a relatively small problem class, was analyzed in [5,8,9,3]. Yu and Zhou [17] provided a general idea on why EAs can fail over a complex problem class; Fournier and Teytaud [6] provided a general lower bound for the performance of EAs over problem classes with VC-dimension measured complexity. However, these studies did not concern the boundary problem cases. Recently, Doerr et al. [3] used the lower bound of running time of the (1+1)-EA with mutation probability $\frac{1}{n}$ on the OneMax problem as that on the pseudo-Boolean function class with a unique global optimum by proving that the OneMax problem is the easiest case for the (1+1)-EA in this class, comparing to which this paper derives a more general result for the easiest case as well as the hardest case. Note that, the easiest case derived in this paper has also been proved by Witt [13], but we give a different and more compact proof.

The rest of this paper is organized as follows. Section 2 introduces some preliminaries. Section 3 presents the main theorem, which is then used to identify the boundary cases in the pseudo-Boolean function class for the (1+1)-EA in Section 4. Section 5 concludes.

2 Preliminaries

Most metaheuristic search algorithms generate solutions only based on their maintained solutions, but not the historical ones, therefore, they can be modeled and analyzed as Markov chains, e.g., [7,17]. In this paper, we only consider the algorithms that can be modeled by Markov chains. A Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ modeling the metaheuristic algorithm is constructed by taking the algorithm's state space $\mathcal{X}$ as the chain's state space, i.e. $\xi_t \in \mathcal{X}$. Let $\mathcal{X}^* \subset \mathcal{X}$ denote the set of all optimal states. The goal of the algorithm is to reach $\mathcal{X}^*$ from an arbitrary initial state. Thus, the process of an algorithm seeking $\mathcal{X}^*$ can be analyzed by studying the corresponding Markov chain.

A Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ is a random process, where, for all $t \geq 0$, ξ_t is defined in the state space $\mathcal{X}$ and ξ_{t+1} depends only on ξ_t. Let $\mathcal{X}^* \subset \mathcal{X}$ be the target space. A Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ is said to be absorbing, if $\forall t \geq 0 : P(\xi_{t+1} \in \mathcal{X}^* | \xi_t \in \mathcal{X}^*) = 1$.

Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, and $\xi_{\tilde{t}} = x$ for arbitrary $\tilde{t} \geq 0$, we define $\tau_{\tilde{t}}$ as a random variable such that $\tau_{\tilde{t}} = \min\{t | \xi_{\tilde{t}+t} \in \mathcal{X}^*, t \geq 0\}$. That is, $\tau_{\tilde{t}}$ is the number of steps needed to reach the target space for the first time from $\tilde{t}$. The mathematical expectation of $\tau_{\tilde{t}}$, $\mathbb{E}[\![\tau_{\tilde{t}} | \xi_{\tilde{t}} = x]\!] = \sum_{i=0}^{\infty} iP(\tau_{\tilde{t}} = i)$, is called the *conditional first hitting*

time (CFHT) of the Markov chain from $\tilde{t}$ and $\xi_{\tilde{t}} = x$. If $\xi_{\tilde{t}}$ is drawn from a distribution $\pi_{\tilde{t}}$, the expectation of the CFHT over $\pi_{\tilde{t}}$, $\mathbb{E}[\![\tau_{\tilde{t}}|\xi_{\tilde{t}} \sim \pi_{\tilde{t}}]\!] = \sum_{x\in\mathcal{X}} \pi_{\tilde{t}}(x)\mathbb{E}[\![\tau_{\tilde{t}}|\xi_{\tilde{t}} = x]\!]$, is called the *distribution-conditional first hitting time* (DCFHT) of the Markov chain from $\tilde{t}$ and $\xi_{\tilde{t}} \sim \pi_{\tilde{t}}$. If $\tilde{t} = 0$, $\mathbb{E}[\![\tau_0|\xi_0 \sim \pi_0]\!]$ is also called the expected running time of the corresponding algorithm.

Switch analysis is a recently proposed approach [16] that compares two Markov chains for their first hitting time. By modeling EAs as Markov chains, it has been used to derive running time bounds of EAs [16] and investigate if one EA runs faster than another EA for a problem [15,16].

Theorem 1 (Switch Analysis [16]). *Given two absorbing Markov chains $\{\xi_t\}_{t=0}^{+\infty}(\xi_t \in \mathcal{X})$ and $\{\xi'_t\}_{t=0}^{+\infty}(\xi'_t \in \mathcal{Y})$, let $\mathcal{X}^*$ and $\mathcal{Y}^*$ denote the optimal state space of ξ_t and ξ'_t, respectively, let τ and τ' denote the hitting events of ξ_t and ξ'_t, respectively, let π_t denote the distribution of ξ_t. Let $\{\rho_t\}_{t=0}^{+\infty}$ be a series of numbers whose sum converges to ρ. If there exists a mapping $\phi : \mathcal{X} \to \mathcal{Y}$, $\phi(x) \in \mathcal{Y}^*$ if and only if $x \in \mathcal{X}^*$; and it satisfies that $\mathbb{E}[\![\tau_0|\xi_0 \sim \pi_0]\!]$ is finite, and for all $t \geq 0$,*

$$\begin{aligned}&\sum\nolimits_{x\in\mathcal{X},y\in\mathcal{Y}} \pi_t(x)P(\xi_{t+1} \in \phi^{-1}(y)|\xi_t = x)\mathbb{E}[\![\tau'_{t+1}|\xi'_{t+1} = y]\!] \\ &\leq (\geq) \sum\nolimits_{y_1,y_2\in\mathcal{Y}} \pi'_t(y_1)P(\xi'_{t+1} = y_2|\xi'_t = y_1)\mathbb{E}[\![\tau'_{t+1}|\xi'_{t+1} = y_2]\!] + \rho_t,\end{aligned} \tag{1}$$

where $\phi^{-1}(y) = \{x \in \mathcal{X}|\phi(x) = y\}$ and $\pi'_t(y) = \pi_t(\phi^{-1}(y))$, it will hold that

$$\mathbb{E}[\![\tau_0|\xi_0 \sim \pi_0]\!] \leq (\geq)\mathbb{E}[\![\tau'_0|\xi'_0 \sim \pi'_0]\!] + \rho.$$

3 A Theorem for Boundary Problem Identification

We assume that the studied problem class is homogeneous, as in Definition 1, which means that all the problem cases in the class have the same solution space and the same optimal solutions when fixing the problem dimensionality. At the first glance, the requirement of the same optimal solutions is a strong restriction. However, since most metaheuristic algorithms do not rely on the meaning of the solution, in the analysis we can commonly switch the optimal solutions. For example, the solution 10011 in a binary space can be shifted as 11111 if we switch the meaning of 1 and 0 for the 2nd and the 3rd bits.

Definition 1 (Homogeneous Problem Class). *A problem class is homogeneous if, for each problem dimensionality, all the problem cases have the same solution space and the same optimal solutions.*

To characterize the algorithm-dependent structure of the problem cases, we partition the state space according to the CFHT of a given algorithm, as in Definition 2. We also define a jumping probability in Definition 3.

Definition 2 (CFHT-Partition). *For a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ with state space $\mathcal{X}$, the CFHT-Partition at time t is a partition of $\mathcal{X}$ into non-empty spaces $\{\mathcal{X}_0^t, \mathcal{X}_1^t, \ldots, \mathcal{X}_m^t\}$ such that $\forall x, y \in \mathcal{X}_i^t, \mathbb{E}[\![\tau_{t+1}|\xi_{t+1} = x]\!] = \mathbb{E}[\![\tau_{t+1}|\xi_{t+1} = y]\!]$ and $\mathbb{E}[\![\tau_{t+1}|\xi_{t+1} \in \mathcal{X}_m^t]\!] > \ldots > \mathbb{E}[\![\tau_{t+1}|\xi_{t+1} \in \mathcal{X}_1^t]\!] > \mathbb{E}[\![\tau_{t+1}|\xi_{t+1} \in \mathcal{X}_0^t]\!] = 0$.*

Definition 3. *For a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ with the state space $\mathcal{X}$, $P_\xi^t(x, \mathcal{X}')$ is the probability of jumping from state x to state space $\mathcal{X}' \subseteq \mathcal{X}$ in one step at time t.*

We then derive Theorem 2, which is a general sufficient condition for identifying the easiest and the hardest problem cases. Note that, the easiest (hardest) problem case of a problem class for an algorithm means that the expected running time of the algorithm on the problem case is the smallest (largest).

Theorem 2. *Given a homogeneous problem class $\mathcal{F}$ and an algorithm $\mathcal{A}$, with dimensionality n, let $\{\xi'_t\}_{t=0}^{+\infty}$ model $\mathcal{A}$ running on a problem $f^* \in \mathcal{F}_n$, of which $\{\mathcal{X}_0^t, \mathcal{X}_1^t, \ldots, \mathcal{X}_m^t\}$ is the CFHT-Partition at time t. If for all problem $f \in \mathcal{F}_n - \{f^*\}$, for all $t \geq 0$, and for all $x \in \mathcal{X} - \mathcal{X}_0^t$, denoting $\{\xi_t\}_{t=0}^{+\infty}$ as the chain modeling $\mathcal{A}$ running on f, there exists an integer $k \in [0, m]$,*

$$\forall j \leq k, P_\xi^t(x, \mathcal{X}_j^t) \leq (\geq) P_{\xi'}^t(x, \mathcal{X}_j^t), \quad \forall j > k, P_\xi^t(x, \mathcal{X}_j^t) \geq (\leq) P_{\xi'}^t(x, \mathcal{X}_j^t), \tag{2}$$

then f^* is the easiest (hardest) case in $\mathcal{F}_n$ for the algorithm $\mathcal{A}$.

Proof. We use the switch analysis approach to show the easiest problem case identification of this theorem by proving that the expected running time of the algorithm $\mathcal{A}$ on the problem f^* is at most as large as that on any other problem. The hardest case identification can be proved similarly. Note that both Markov chains $\{\xi_t\}_{t=0}^{+\infty}$ and $\{\xi'_t\}_{t=0}^{+\infty}$ can be transformed to be absorbing by letting them always stay at the optimal state once an optimal state has been found, and this transformation does not affect their running time by the definition of CFHT/DCFHT.

The two chains $\{\xi_t\}_{t=0}^{+\infty}$ and $\{\xi'_t\}_{t=0}^{+\infty}$ have the same state space $\mathcal{X}$ and the same optimal state space $\mathcal{X}^*$, since the studied problem class is homogeneous. For the clearness of the proof, we denote the state space and the optimal space of $\{\xi'_t\}_{t=0}^{+\infty}$ by $\mathcal{Y}$ and $\mathcal{Y}^*$, respectively. Obviously, $\mathcal{Y} = \mathcal{X}$ and $\mathcal{Y}^* = \mathcal{X}^*$. Then, we construct the mapping $\phi : \mathcal{X} \to \mathcal{Y}$ as that $\forall x \in \mathcal{X} : \phi(x) = x$. It is obvious that $\phi(x) \in \mathcal{Y}^*$ iff $x \in \mathcal{X}^*$.

Then, we investigate Eq. 1 in switch analysis. For an optimal state $x \in \mathcal{X}^* = \mathcal{X}_0^t$, since $\phi(x) = x$ and both Markov chains are absorbing, we have

$$\begin{aligned} &\sum\nolimits_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\![\tau'_{t+1} \mid \xi'_{t+1} = y]\!] \\ &= \sum\nolimits_{y \in \mathcal{Y}} P(\xi'_{t+1} = y \mid \xi'_t = \phi(x)) \mathbb{E}[\![\tau'_{t+1} \mid \xi'_{t+1} = y]\!] = 0. \end{aligned} \tag{3}$$

For a non-optimal state $x \in \mathcal{X}_i^t$ $(i \geq 1)$, since $\phi(x) = x$, we have

$$\sum_{y \in \mathcal{Y}} P(\xi'_{t+1} = y | \xi'_t = \phi(x)) \mathbb{E}[\![\tau'_{t+1} | \xi'_{t+1} = y]\!] = \sum_{j=0}^{m} P_{\xi'}^t(x, \mathcal{X}_j^t) \mathbb{E}[\![\tau'_{t+1} | \xi'_{t+1} \in \mathcal{X}_j^t]\!];$$

$$\sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\![\tau'_{t+1} | \xi'_{t+1} = y]\!] = \sum_{j=0}^{m} P_\xi^t(x, \mathcal{X}_j^t) \mathbb{E}[\![\tau'_{t+1} | \xi'_{t+1} \in \mathcal{X}_j^t]\!].$$

By comparing the above two equalities, since the condition Eq. 2 holds, and furthermore, $\mathbb{E}[\![\tau'_{t+1} | \xi'_{t+1} \in \mathcal{X}_j^t]\!]$ increases with j, we have

$$\sum_{y\in\mathcal{Y}} P(\xi_{t+1}\in\phi^{-1}(y)|\xi_t=x)\mathbb{E}[\![\tau'_{t+1}|\xi'_{t+1}=y]\!] \geq \sum_{y\in\mathcal{Y}} P(\xi'_{t+1}=y|\xi'_t=\phi(x))\mathbb{E}[\![\tau'_{t+1}|\xi'_{t+1}=y]\!]. \quad (4)$$

Then, by combining Eq. 3 with Eq. 4, we have for all $t \geq 0$,

$$\begin{aligned}
&\sum_{x\in\mathcal{X},y\in\mathcal{Y}} \pi_t(x)P(\xi_{t+1}\in\phi^{-1}(y)|\xi_t=x)\mathbb{E}[\![\tau'_{t+1}|\xi'_{t+1}=y]\!] \\
&\geq \sum_{x\in\mathcal{X},y\in\mathcal{Y}} \pi_t(x)P(\xi'_{t+1}=y|\xi'_t=\phi(x))\mathbb{E}[\![\tau'_{t+1}|\xi'_{t+1}=y]\!] \\
&= \sum_{y_1,y_2\in\mathcal{Y}} \pi'_t(y_1)P(\xi'_{t+1}=y_2|\xi'_t=y_1)\mathbb{E}[\![\tau'_{t+1}|\xi'_{t+1}=y_2]\!]. \quad (\text{by } \pi'_t(y)=\pi_t(\phi^{-1}(y)))
\end{aligned}$$

Thus, Eq. 1 holds with $\rho_t = 0$. By switch analysis, $\mathbb{E}[\![\tau_0|\xi_0\sim\pi_0]\!] \geq \mathbb{E}[\![\tau'_0|\xi'_0\sim\pi'_0]\!]$. Since $\pi_0 = \pi'_0$, we have $\mathbb{E}[\![\tau_0|\xi_0\sim\pi_0]\!] \geq \mathbb{E}[\![\tau'_0|\xi'_0\sim\pi_0]\!]$. The two sides of this inequality are the expected running time of $\mathcal{A}$ on a problem $f\in\mathcal{F}_n-\{f^*\}$ and that on the problem f^*, respectively. This inequality holds for each $f\in\mathcal{F}_n-\{f^*\}$ and f^*. Thus, f^* is the easiest problem in $\mathcal{F}_n$ for the algorithm $\mathcal{A}$. □

4 Pseudo-Boolean Function Class and (1+1)-EA

In this section, we use the proved theorem to identify the easiest and the hardest function in the pseudo-Boolean function class with a unique global optimum for the (1+1)-EA with mutation probability less than 0.5.

The pseudo-Boolean function class in Definition 4 is a large function class which only requires the solution space to be $\{0,1\}^n$ and the objective space to be $\mathbb{R}$. The pseudo-Boolean function class with a unique global optimum is a subset of the pseudo-Boolean function class, where each function has a unique global optimum. Here, we consider maximization problems since minimizing f is equivalent to maximizing $-f$. The OneMax problem in Definition 5 is to maximize the number of 1 bits of a solution. The Trap problem in Definition 6 is to maximize the number of 0 bits of a solution except the global optimum 1^n.

Definition 4 (Pseudo-Boolean Function Class). *A function in the pseudo-Boolean function class has the form:* $f:\{0,1\}^n\to\mathbb{R}$.

Definition 5 (OneMax Problem). *OneMax Problem of size* n *is to find an* n *bits binary string* x^* *such that* $x^*=\arg\max_{x\in\{0,1\}^n}\{f(x)|f(x)=\sum_{i=1}^n x_i\}$.

Definition 6 (Trap Problem). *Trap Problem of size* n *is to find an* n *bits binary string* x^* *such that* $x^*=\arg\max_{x\in\{0,1\}^n}\{f(x)|f(x)=\sum_{i=1}^n(1-x_i)+(n+1)\prod_{i=1}^n x_i\}$.

The (1+1)-EA [4] in Algorithm 1 is a randomized heuristic algorithm for maximizing pseudo-Boolean functions, which is often involved in theoretical analysis of EAs.

Algorithm 1 ((1+1)-EA). *Given solution length* n *and objective function* f*, (1+1)-EA consists of the following steps:*

1. $x :=$ *randomly selected from* $\{0,1\}^n$.
2. Repeat until the termination condition is met
3. $x' :=$ *flip each bit of* x *with probability* p*;*
4. if $f(x') \geq f(x)$
5. $x := x'$*;*
where $p \in (0,1)$ *is the mutation probability.*

For a pseudo-Boolean function with a unique global optimum, we assume without loss of generality that the optimal solution is 1^n. This is because the (1+1)-EA treats the bits 0 and 1 symmetrically, and thus the 0 bits in an optimal solution can be interpreted as 1 bits without affecting the behavior of the (1+1)-EA. The optimization time of the (1+1)-EA for maximizing a pseudo-Boolean function is computed as the number of iterations until a global optimum has been found for the first time.

Theorem 3. *In the pseudo-Boolean function class with a unique global optimum, the OneMax and the Trap problem are the easiest and the hardest problem case for the (1+1)-EA with* $0 < p < 0.5$*, respectively.*

Before proving Theorem 3, we first prove the order of the CFHT of the (1+1)-EA on the OneMax problem in Lemma 1 as well as that on the Trap problem in Lemma 2. Since the bits of the OneMax problem are independent and their weights are same, it is not hard to see that the CFHT $\mathbb{E}[\![\tau'_t | \xi'_t = x]\!]$ of the (1+1)-EA on the OneMax problem only depends on the number of 1 bits of the solution x, i.e., $\|x\|$. Thus, we denote $\mathbb{E}(j)$ as the CFHT $\mathbb{E}[\![\tau'_t | \xi'_t = x]\!]$ with $\|x\| = n - j$. Then, it is obvious that $\mathbb{E}(0) = 0$, which implies the optimal solution.

Lemma 1. *For* $0 < p < 0.5$*, it holds that* $\mathbb{E}(0) < \mathbb{E}(1) < \mathbb{E}(2) < \ldots < \mathbb{E}(n)$.

Proof. We prove $\forall\, 0 \leq j < n : \mathbb{E}(j) < \mathbb{E}(j+1)$ inductively on j.

(a) Initialization is to prove $\mathbb{E}(0) < \mathbb{E}(1)$. Since $\mathbb{E}(1) = 1 + p(1-p)^{n-1}\mathbb{E}(0) + (1 - p(1-p)^{n-1})\mathbb{E}(1)$, we have $\mathbb{E}(1) = 1/(p(1-p)^{n-1}) > 0 = \mathbb{E}(0)$.

(b) Inductive Hypothesis assumes that

$$\forall\, 0 \leq j < K (K \leq n-1) : \mathbb{E}(j) < \mathbb{E}(j+1).$$

Then, we consider $j = K$. Let x and x' be a solution with $K+1$ number of 0 bits and that with K number of 0 bits, respectively. Then, we have $\mathbb{E}(K+1) = \mathbb{E}[\![\tau'_t | \xi'_t = x]\!]$ and $\mathbb{E}(K) = \mathbb{E}[\![\tau'_t | \xi'_t = x']\!]$. For a Boolean string of length $n-1$ with K number of 0 bits, we denote P_i $(0 \leq i \leq n-1)$ as the probability that the number of 0 bits changes to be i after bit-wise mutation on this string with mutation probability p.

For the solution x, we divide the mutation on x into two parts: mutation on one 0 bit and mutation on the $n-1$ remaining bits. The $n-1$ remaining bits contain K number of 0 bits since $n - \|x\| = K + 1$. Then, by considering the mutation and selection behavior of the (1+1)-EA on the OneMax problem, we have

$$\begin{aligned}\mathbb{E}(K+1) = 1 + p \cdot \Big(\sum\nolimits_{i=0}^{K+1} P_i \mathbb{E}(i) + \sum\nolimits_{i=K+2}^{n-1} P_i \mathbb{E}(K+1)\Big) \\ + (1-p) \cdot \Big(\sum\nolimits_{i=0}^{K} P_i \mathbb{E}(i+1) + \sum\nolimits_{i=K+1}^{n-1} P_i \mathbb{E}(K+1)\Big),\end{aligned}$$

where the term p is the probability that the 0 bit in the first mutation part is flipped.

For the solution x', we also divide the mutation on x' into two parts: mutation on one 1 bit and mutation on the $n-1$ remaining bits. The $n-1$ remaining bits also contain K number of 0 bits since $n - \|x'\| = K$. Then, we have

$$\begin{aligned}\mathbb{E}(K) = 1 + p \cdot (\sum\nolimits_{i=0}^{K-1} P_i \mathbb{E}(i+1) + \sum\nolimits_{i=K}^{n-1} P_i \mathbb{E}(K)) \\ + (1-p) \cdot (\sum\nolimits_{i=0}^{K} P_i \mathbb{E}(i) + \sum\nolimits_{i=K+1}^{n-1} P_i \mathbb{E}(K)),\end{aligned}$$

where the term p is the probability that the 1 bit in the first mutation part is flipped.

From the above two equalities, we have

$$\begin{aligned}&\mathbb{E}(K+1) - \mathbb{E}(K) \\ &= p \cdot \big(\sum\nolimits_{i=0}^{K-1} P_i(\mathbb{E}(i) - \mathbb{E}(i+1)) + \sum\nolimits_{i=K+1}^{n-1} P_i(\mathbb{E}(K+1) - \mathbb{E}(K)) \big) \\ &+ (1-p) \cdot \big(\sum\nolimits_{i=0}^{K} P_i(\mathbb{E}(i+1) - \mathbb{E}(i)) + \sum\nolimits_{i=K+1}^{n-1} P_i(\mathbb{E}(K+1) - \mathbb{E}(K)) \big) \\ &= (1-2p) \cdot \big(\sum\nolimits_{i=0}^{K-1} P_i(\mathbb{E}(i+1) - \mathbb{E}(i)) \big) \\ &+ \big((1-p)P_K + \sum\nolimits_{i=K+1}^{n-1} P_i \big) \cdot \big(\mathbb{E}(K+1) - \mathbb{E}(K) \big) \\ &> \big((1-p)P_K + \sum\nolimits_{i=K+1}^{n-1} P_i \big) \cdot \big(\mathbb{E}(K+1) - \mathbb{E}(K) \big),\end{aligned}$$

where the inequality is by $0 < p < 0.5$ and inductive hypothesis.

Since $(1-p)P_K + \sum_{i=K+1}^{n-1} P_i < 1$, we have $\mathbb{E}(K+1) > \mathbb{E}(K)$. □

For the Trap problem, it is not hard to see that the CFHT $\mathbb{E}[\![\tau'_t | \xi'_t = x]\!]$ of the (1+1)-EA on the Trap problem also only depends on the number of 1 bits of the solution x. Thus, we denote $\mathbb{E}'(j)$ as the CFHT $\mathbb{E}[\![\tau'_t | \xi'_t = x]\!]$ with $\|x\| = n - j$. Then, it is obvious that $\mathbb{E}'(0) = 0$, which implies the optimal solution.

Lemma 2. *For $0 < p < 0.5$, it holds that $\mathbb{E}'(0) < \mathbb{E}'(1) < \mathbb{E}'(2) < \ldots < \mathbb{E}'(n)$.*

Proof. First, $\mathbb{E}'(0) < \mathbb{E}'(1)$ trivially holds, since $\mathbb{E}'(0) = 0$ and $\mathbb{E}'(1) > 0$. Then, we prove $\forall\, 0 < j < n : \mathbb{E}'(j) < \mathbb{E}'(j+1)$ inductively on j.

(a) Initialization is to prove $\mathbb{E}'(n-1) < \mathbb{E}'(n)$. For $\mathbb{E}'(n)$, since only the offspring 0^n or 1^n will be accepted, we have $\mathbb{E}'(n) = 1 + p^n \mathbb{E}'(0) + (1-p^n)\mathbb{E}'(n)$, then, $\mathbb{E}'(n) = 1/p^n$. For $\mathbb{E}'(n-1)$, since the accepted offsprings are 0^n, the solutions with $n-1$ number of 0 bits and 1^n, we have $\mathbb{E}'(n-1) = 1 + p^{n-1}(1-p)\mathbb{E}'(0) + p(1-p)^{n-1}\mathbb{E}'(n) + (1 - p^{n-1}(1-p) - p(1-p)^{n-1})\mathbb{E}'(n-1)$, then, $\mathbb{E}'(n-1) = (1 + (1-p)^{n-1}/p^{n-1})/(p^{n-1}(1-p) + p(1-p)^{n-1})$. Thus, we have

$$\frac{\mathbb{E}'(n)}{\mathbb{E}'(n-1)} = \frac{p^{n-1}(1-p) + p(1-p)^{n-1}}{p^n + (1-p)^{n-1}p} > 1,$$

where the inequality is by $0 < p < 0.5$.

(b) Inductive Hypothesis assumes that

$$\forall\, K < j \leq n-1 (K \geq 1) : \mathbb{E}'(j) < \mathbb{E}'(j+1).$$

Then, we consider $j = K$. When comparing $\mathbb{E}'(K+1)$ with $\mathbb{E}'(K)$, we use the same analysis method as that in the proof of Lemma 1. By additionally considering the selection behavior of the (1+1)-EA on the Trap problem which is different from that on the OneMax problem, we can get

$$\begin{aligned}\mathbb{E}'(K+1) = 1 + p \cdot (P_0\mathbb{E}'(0) + \sum\nolimits_{i=1}^{K} P_i\mathbb{E}'(K+1) + \sum\nolimits_{i=K+1}^{n-1} P_i\mathbb{E}'(i)) \\ + (1-p) \cdot (\sum\nolimits_{i=0}^{K-1} P_i\mathbb{E}'(K+1) + \sum\nolimits_{i=K}^{n-1} P_i\mathbb{E}'(i+1)),\end{aligned}$$

and

$$\begin{aligned}\mathbb{E}'(K) = 1 + p \cdot (\sum\nolimits_{i=0}^{K-2} P_i\mathbb{E}'(K) + \sum\nolimits_{i=K-1}^{n-1} P_i\mathbb{E}'(i+1)) \\ + (1-p) \cdot (P_0\mathbb{E}'(0) + \sum\nolimits_{i=1}^{K-1} P_i\mathbb{E}'(K) + \sum\nolimits_{i=K}^{n-1} P_i\mathbb{E}'(i)).\end{aligned}$$

From the above two equalities, we have

$$\begin{aligned}&\mathbb{E}'(K+1) - \mathbb{E}'(K) = p \cdot \big(P_0(\mathbb{E}'(0) - \mathbb{E}'(K)) + \sum\nolimits_{i=1}^{K-1} P_i(\mathbb{E}'(K+1) - \mathbb{E}'(K)) \\ &+ \sum\nolimits_{i=K+1}^{n-1} P_i(\mathbb{E}'(i) - \mathbb{E}'(i+1))\big) + (1-p) \cdot \big(P_0(\mathbb{E}'(K+1) - \mathbb{E}'(0)) \\ &+ \sum\nolimits_{i=1}^{K} P_i(\mathbb{E}'(K+1) - \mathbb{E}'(K)) + \sum\nolimits_{i=K+1}^{n-1} P_i(\mathbb{E}'(i+1) - \mathbb{E}'(i))\big) \\ &= P_0 \cdot \big((1-p)\mathbb{E}'(K+1) - p\mathbb{E}'(K)\big) + \big(\sum\nolimits_{i=1}^{K-1} P_i + (1-p)P_K\big) \\ &\cdot \big(\mathbb{E}'(K+1) - \mathbb{E}'(K)\big) + (1-2p) \cdot \big(\sum\nolimits_{i=K+1}^{n-1} P_i(\mathbb{E}'(i+1) - \mathbb{E}'(i))\big) \\ &> \big(\sum\nolimits_{i=1}^{K-1} P_i + (1-p)P_K + pP_0\big) \cdot \big(\mathbb{E}'(K+1) - \mathbb{E}'(K)\big),\end{aligned}$$

where the inequality is by $0 < p < 0.5$ and inductive hypothesis.

Since $\sum_{i=1}^{K-1} P_i + (1-p)P_K + pP_0 < 1$, we have $\mathbb{E}'(K+1) > \mathbb{E}'(K)$. □

Proof of Theorem 3. The pseudo-Boolean function class with a unique global optimum is homogeneous, since for each dimensionality n, the solution space and the optimal solution for any function are $\{0,1\}^n$ and 1^n, respectively. By the behavior of the (1+1)-EA, it is easy to see that the (1+1)-EA can be modeled as a Markov chain.

Let the OneMax problem correspond to f^* in Theorem 2. Then for the parameter m and $\mathcal{X}_i^t$ in Theorem 2, we have $m = n$ and $\mathcal{X}_i^t = \{x | \|x\| = n - i\}$ $(0 \leq i \leq n)$ by Lemma 1. For any non-optimal solution $x \in \mathcal{X}_k^t$ $(k > 0)$, we denote $P(j)$ $(0 \leq j \leq n)$ as the probability that the offspring generated by bit-wise mutation on x has j number of 0 bits. For $\{\xi'_t\}_{t=0}^{+\infty}$, since only the offspring solution with no more 0 bits will be accepted, we have

$$\begin{aligned}&\forall\, 0 \leq j \leq k-1 : P_{\xi'}^t(x, \mathcal{X}_j^t) = P(j); \quad P_{\xi'}^t(x, \mathcal{X}_k^t) = \sum\nolimits_{j=k}^{n} P(j); \\ &\forall\, k+1 \leq j \leq n : P_{\xi'}^t(x, \mathcal{X}_j^t) = 0.\end{aligned}$$

For $\{\xi_t\}_{t=0}^{+\infty}$, since the offspring solution with less 0 bits may be rejected and that with more 0 bits may be accepted, we have

$$P_\xi^t(x, \mathcal{X}_0^t) = P(0); \qquad \forall\, 1 \le j \le k-1 : P_\xi^t(x, \mathcal{X}_j^t) \le P(j);$$
$$\forall\, k+1 \le j \le n : P_\xi^t(x, \mathcal{X}_j^t) \ge 0.$$

Thus, if $P_{\xi'}^t(x, \mathcal{X}_k^t) \ge P_\xi^t(x, \mathcal{X}_k^t)$, we have

$$\forall\, 0 \le j \le k : P_\xi^t(x, \mathcal{X}_j^t) \le P_{\xi'}^t(x, \mathcal{X}_j^t),\ \forall\, k+1 \le j \le n : P_\xi^t(x, \mathcal{X}_j^t) \ge P_{\xi'}^t(x, \mathcal{X}_j^t);$$

otherwise, we have

$$\forall\, 0 \le j \le k-1 : P_\xi^t(x, \mathcal{X}_j^t) \le P_{\xi'}^t(x, \mathcal{X}_j^t),\ \forall\, k \le j \le n : P_\xi^t(x, \mathcal{X}_j^t) \ge P_{\xi'}^t(x, \mathcal{X}_j^t).$$

Note that the above two formulas hold for all $t \ge 0$, since the (1+1)-EA uses time-invariant operators. Therefore, by Theorem 2, we get that the OneMax problem is the easiest in the pseudo-Boolean function class with a unique global optimum for the (1+1)-EA with $p < 0.5$.

Let the Trap problem correspond to f^*. By Lemma 2, we have $m = n$ and $\mathcal{X}_i^t = \{x | \|x\| = n - i\}$ $(1 \le i \le n)$. For any non-optimal solution $x \in \mathcal{X}_k^t$ $(k > 0)$, we also denote $P(j)$ $(0 \le j \le n)$ as the probability that the offspring generated by bit-wise mutation on x has j number of 0 bits. For $\{\xi'_t\}_{t=0}^{+\infty}$, since only the optimal solution and the offspring solutions with no less 0 bits will be accepted, we have

$$P_{\xi'}^t(x, \mathcal{X}_0^t) = P(0); \qquad \forall\, 1 \le j \le k-1 : P_{\xi'}^t(x, \mathcal{X}_j^t) = 0;$$
$$P_{\xi'}^t(x, \mathcal{X}_k^t) = \sum\nolimits_{j=1}^{k} P(j); \quad \forall\, k+1 \le j \le n : P_{\xi'}^t(x, \mathcal{X}_j^t) = P(j).$$

For $\{\xi_t\}_{t=0}^{+\infty}$, since the offspring solution with less 0 bits may be accepted and that with more 0 bits may be rejected, we have

$$P_\xi^t(x, \mathcal{X}_0^t) = P(0); \qquad \forall\, 1 \le j \le k-1 : P_\xi^t(x, \mathcal{X}_j^t) \ge 0;$$
$$\forall\, k+1 \le j \le n : P_\xi^t(x, \mathcal{X}_j^t) \le P(j).$$

Thus, if $P_{\xi'}^t(x, \mathcal{X}_k^t) \ge P_\xi^t(x, \mathcal{X}_k^t)$, we have

$$\forall\, 0 \le j \le k-1 : P_\xi^t(x, \mathcal{X}_j^t) \ge P_{\xi'}^t(x, \mathcal{X}_j^t),\ \forall\, k \le j \le n : P_\xi^t(x, \mathcal{X}_j^t) \le P_{\xi'}^t(x, \mathcal{X}_j^t);$$

otherwise, we have

$$\forall\, 0 \le j \le k : P_\xi^t(x, \mathcal{X}_j^t) \ge P_{\xi'}^t(x, \mathcal{X}_j^t),\ \forall\, k+1 \le j \le n : P_\xi^t(x, \mathcal{X}_j^t) \le P_{\xi'}^t(x, \mathcal{X}_j^t).$$

By Theorem 2, we get that the Trap problem is the hardest in the pseudo-Boolean function class with a unique global optimum for the (1+1)-EA with $p < 0.5$. □

5 Conclusion

In this paper, we derive a theorem to identify the easiest and the hardest problem cases of a problem class for an algorithm. Using the theorem, we prove that the OneMax and the Trap problem are the easiest and the hardest function in the pseudo-Boolean function class with a unique global optimum for the (1+1)-EA with mutation probability less than 0.5, respectively, which much extends the previous knowledge [3]. In the future, we will apply this theorem for more problem classes and more algorithms.

References

1. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific, Singapore (2011)
2. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford (1996)
3. Doerr, B., Johannsen, D., Winzen, C.: Drift analysis and linear functions revisited. In: Proceedings of the 2010 IEEE Congress on Evolutionary Computation, Barcelona, Spain, pp. 1–8 (2010)
4. Droste, S., Jansen, T., Wegener, I.: A rigorous complexity analysis of the (1+1) evolutionary algorithm for linear functions with Boolean inputs. Evolutionary Computation 6(2), 185–196 (1998)
5. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science 276(1-2), 51–81 (2002)
6. Fournier, H., Teytaud, O.: Lower bounds for comparison based evolution strategies using VC-dimension and sign patterns. Algorithmica 59(3), 387–408 (2011)
7. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artificial Intelligence 127(1), 57–85 (2001)
8. He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. Natural Computing 3(1), 21–35 (2004)
9. Jägersküpper, J.: A blend of Markov-chain and drift analysis. In: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature, Dortmund, Germany, pp. 41–51 (2008)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942–1948 (1995)
11. Kirkpatrick, S.: Optimization by simulated annealing: Quantitative studies. Journal of Statistical Physics 34(5), 975–986 (1984)
12. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity. Springer, Berlin (2010)
13. Witt, C.: Optimizing linear functions with randomized search heuristics-The robustness of mutation. In: Proceedings of the 29th Symposium on Theoretical Aspects of Computer Science, Paris, France, pp. 420–431 (2012)
14. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)
15. Yu, Y., Qian, C., Zhou, Z.-H.: Towards Analyzing Recombination Operators in Evolutionary Search. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 144–153. Springer, Heidelberg (2010)
16. Yu, Y., Qian, C., Zhou, Z.-H.: Towards analyzing crossover operators in evolutionary search via general Markov chain switching theorem. CORR abs/1111.0907 (2011)
17. Yu, Y., Zhou, Z.-H.: A new approach to estimating the expected first hitting time of evolutionary algorithms. Artificial Intelligence 172(15), 1809–1832 (2008)

Cumulative Step-Size Adaptation on Linear Functions

Alexandre Chotard, Anne Auger, and Nikolaus Hansen

TAO team, INRIA Saclay-Ile-de-France, LRI, Paris-Sud University, France
firstname.lastname@lri.fr

Abstract. The CSA-ES is an Evolution Strategy with Cumulative Step size Adaptation, where the step size is adapted measuring the length of a so-called cumulative path. The cumulative path is a combination of the previous steps realized by the algorithm, where the importance of each step decreases with time. This article studies the CSA-ES on composites of strictly increasing functions with affine linear functions through the investigation of its underlying Markov chains. Rigorous results on the change and the variation of the step size are derived with and without cumulation. The step-size diverges geometrically fast in most cases. Furthermore, the influence of the cumulation parameter is studied.

Keywords: CSA, cumulative path, evolution path, evolution strategies, step-size adaptation.

1 Introduction

Evolution strategies (ESs) are continuous stochastic optimization algorithms searching for the minimum of a real valued function $f : \mathbb{R}^n \to \mathbb{R}$. In the $(1, \lambda)$-ES, in each iteration, λ new children are generated from a single parent point $\boldsymbol{X} \in \mathbb{R}^n$ by adding a random Gaussian vector to the parent,

$$\boldsymbol{X} \in \mathbb{R}^n \mapsto \boldsymbol{X} + \sigma \boldsymbol{\mathcal{N}}(\mathbf{0}, \boldsymbol{C}) \ .$$

Here, $\sigma \in \mathbb{R}_+^*$ is called step-size and $\boldsymbol{C}$ is a covariance matrix. The best of the λ children, i.e. the one with the lowest f-value, becomes the parent of the next iteration. To achieve reasonably fast convergence, step size and covariance matrix have to be adapted throughout the iterations of the algorithm. In this paper, $\boldsymbol{C}$ is the identity and we investigate the so-called Cumulative Step-size Adaptation (CSA), which is used to adapt the step-size in the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [12,10]. In CSA, a cumulative path is introduced, which is a combination of all steps the algorithm has made, where the importance of a step decreases exponentially with time. Arnold and Beyer studied the behavior of CSA on sphere, cigar and ridge functions [1,2,3,7] and on dynamical optimization problems where the optimum moves randomly [5] or linearly [6]. Arnold also studied the behaviour of a $(1, \lambda)$-ES on linear functions with linear constraint [4].

In this paper, we study the behaviour of the $(1, \lambda)$-CSA-ES on composites of strictly increasing functions with affine linear functions, e.g. $f : \boldsymbol{x} \mapsto \exp(x_2 - 2)$. Because the CSA-ES is invariant under translation, under change of an orthonormal basis (rotation and reflection), and under strictly increasing transformations of the f-value, we

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 72–81, 2012.

investigate, w.l.o.g., $f : \boldsymbol{x} \mapsto x_1$. Linear functions model the situation when the current parent is far (here infinitely far) from the optimum of a smooth function. To be far from the optimum means that the distance to the optimum is large, *relative to the step-size* σ. This situation is undesirable and threatens premature convergence. The situation should be handled well, by increasing step widths, by any search algorithm (and is not handled well by the $(1, 2)$-σSA-ES [9]). Solving linear functions is also very useful to prove convergence independently of the initial state on more general function classes.

In Section 2 we introduce the $(1, \lambda)$-CSA-ES, and some of its characteristics on linear functions. In Sections 3 and 4 we study $\ln(\sigma_t)$ without and with cumulation, respectively. Section 5 presents an analysis of the variance of the logarithm of the step-size and in Section 6 we summarize our results.

Notations. In this paper, we denote t the iteration or time index, n the search space dimension, $\mathcal{N}(0, 1)$ a standard normal distribution, i.e. a normal distribution with mean zero and standard deviation 1. The multivariate normal distribution with mean vector zero and covariance matrix identity will be denoted $\boldsymbol{\mathcal{N}}(\mathbf{0}, I_n)$, the i^{th} order statistic of λ standard normal distributions $\mathcal{N}_{i:\lambda}$, and $\Psi_{i:\lambda}$ its distribution. If $\boldsymbol{x} = (x_1, \cdots, x_n) \in \mathbb{R}^n$ is a vector, then $[x]_i$ will be its value on the i^{th} dimension, that is $[x]_i = x_i$. A random variable $\boldsymbol{X}$ distributed according to a law $\mathcal{L}$ will be denoted $\boldsymbol{X} \sim \mathcal{L}$.

2 The $(1, \lambda)$-CSA-ES

We denote with $\boldsymbol{X}_t$ the parent at the t^{th} iteration. From the parent point $\boldsymbol{X}_t$, λ children are generated: $\boldsymbol{Y}_{t,i} = \boldsymbol{X}_t + \sigma_t \boldsymbol{\xi}_{t,i}$ with $i \in [[1, \lambda]]$, and $\boldsymbol{\xi}_{t,i} \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, I_n)$, $(\boldsymbol{\xi}_{t,i})_{i \in [[1,\lambda]]}$ i.i.d. Due to the $(1, \lambda)$ selection scheme, from these children, the one minimizing the function f is selected: $\boldsymbol{X}_{t+1} = \text{argmin}\{f(\boldsymbol{Y}), \boldsymbol{Y} \in \{\boldsymbol{Y}_{t,1}, ..., \boldsymbol{Y}_{t,\lambda}\}\}$. This latter equation implicitly defines the random variable $\boldsymbol{\xi}_t^\star$ as

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \sigma_t \boldsymbol{\xi}_t^\star \ . \tag{1}$$

In order to adapt the step-size, the cumulative path is defined as

$$\boldsymbol{p}_{t+1} = (1 - c)\boldsymbol{p}_t + \sqrt{c(2 - c)}\, \boldsymbol{\xi}_t^\star \tag{2}$$

with $0 < c \leq 1$. The constant $1/c$ represents the life span of the information contained in $\boldsymbol{p}_t$, as after $1/c$ generations $\boldsymbol{p}_t$ is multiplied by a factor that approaches $1/e \approx 0.37$ for $c \to 0$ from below (indeed $(1-c)^{1/c} \leq \exp(-1)$). The typical value for c is between $1/\sqrt{n}$ and $1/n$. We will consider that $\boldsymbol{p}_0 \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, I_n)$ as it makes the algorithm easier to analyze.

The normalization constant $\sqrt{c(2 - c)}$ in front of $\boldsymbol{\xi}_t^\star$ in Eq. (2) is chosen so that under random selection and if $\boldsymbol{p}_t$ is distributed according to $\boldsymbol{\mathcal{N}}(\mathbf{0}, I_n)$ then also $\boldsymbol{p}_{t+1}$ follows $\boldsymbol{\mathcal{N}}(\mathbf{0}, I_n)$. Hence the length of the path can be compared to the expected length of $\|\boldsymbol{\mathcal{N}}(\mathbf{0}, I_n)\|$ representing the expected length under random selection.

The step-size update rule increases the step-size if the length of the path is larger than the length under random selection and decreases it if the length is shorter than under random selection:

$$\sigma_{t+1} = \sigma_t \exp\left(\frac{c}{d_\sigma}\left(\frac{\|\boldsymbol{p}_{t+1}\|}{E(\|\boldsymbol{\mathcal{N}}(\mathbf{0}, I_n)\|)} - 1\right)\right)$$

where the damping parameter d_σ determines how much the step-size can change and is set to $d_\sigma = 1$. A simplification of the update considers the squared length of the path [5]:

$$\sigma_{t+1} = \sigma_t \exp\left(\frac{c}{2d_\sigma}\left(\frac{\|\boldsymbol{p}_{t+1}\|^2}{n} - 1\right)\right). \tag{3}$$

This rule is easier to analyse and we will use it throughout the paper.

Preliminary results on linear functions. Selection on the linear function, $f(\boldsymbol{x}) = [\boldsymbol{x}]_1$, is determined by $[\boldsymbol{X}_t]_1 + \sigma_t [\boldsymbol{\xi}_t^\star]_1 \leq [\boldsymbol{X}_t]_1 + \sigma_t \left[\boldsymbol{\xi}_{t,i}\right]_1$ for all i which is equivalent to $[\boldsymbol{\xi}_t^\star]_1 \leq \left[\boldsymbol{\xi}_{t,i}\right]_1$ for all i where by definition $\left[\boldsymbol{\xi}_{t,i}\right]_1$ is distributed according to $\mathcal{N}(0,1)$. Therefore the first coordinate of the selected step is distributed according to $\mathcal{N}_{1:\lambda}$ and all others coordinates are distributed according to $\mathcal{N}(0,1)$, i.e. selection does not bias the distribution along the coordinates $2, \ldots, n$. Overall we have the following result.

Lemma 1. *On the linear function $f(\boldsymbol{x}) = x_1$, the selected steps $(\boldsymbol{\xi}_t^\star)_{t\in\mathbb{N}}$ of the $(1,\lambda)$-ES are i.i.d. and distributed according to the vector $\boldsymbol{\xi} := (\mathcal{N}_{1:\lambda}, \mathcal{N}_2, \ldots, \mathcal{N}_n)$ where $\mathcal{N}_i \sim \mathcal{N}(0,1)$ for $i \geq 2$.*

Because the selected steps $\boldsymbol{\xi}_t^\star$ are i.i.d. the path defined in Eq. 2 is an autonomous Markov chain, that we will denote $\boldsymbol{\mathcal{P}} = (\boldsymbol{p}_t)_{t\in\mathbb{N}}$. Note that if the distribution of the selected step depended on $(\boldsymbol{X}_t, \sigma_t)$ as it is generally the case on non-linear functions, then the path alone would not be a Markov Chain, however $(\boldsymbol{X}_t, \sigma_t, \boldsymbol{p}_t)$ would be an autonomous Markov Chain. In order to study whether the $(1,\lambda)$-CSA-ES diverges geometrically, we investigate the log of the step-size change, whose formula can be immediately deduced from Eq. 3:

$$\ln\left(\frac{\sigma_{t+1}}{\sigma_t}\right) = \frac{c}{2d_\sigma}\left(\frac{\|\boldsymbol{p}_{t+1}\|^2}{n} - 1\right) \tag{4}$$

By summing up this equation from 0 to $t-1$ we obtain

$$\frac{1}{t}\ln\left(\frac{\sigma_t}{\sigma_0}\right) = \frac{c}{2d_\sigma}\left(\frac{1}{t}\sum_{k=1}^{t}\frac{\|\boldsymbol{p}_k\|^2}{n} - 1\right) \quad . \tag{5}$$

We are interested to know whether $\frac{1}{t}\ln(\sigma_t/\sigma_0)$ converges to a constant. In case this constant is positive this will prove that the $(1,\lambda)$-CSA-ES diverges geometrically. We recognize thanks to (5) that this quantity is equal to the sum of t terms divided by t that suggests the use of the law of large numbers to prove convergence of (5). We will start by investigating the case without cumulation $c = 1$ (Section 3) and then the case with cumulation (Section 4).

3 Divergence Rate of $(1, \lambda)$-CSA-ES without Cumulation

In this section we study the $(1, \lambda)$-CSA-ES without cumulation, i.e. $c = 1$. In this case, the path always equals to the selected step, i.e. for all t, we have $\boldsymbol{p}_{t+1} = \boldsymbol{\xi}_t^\star$. We have proven in Lemma 1 that $\boldsymbol{\xi}_t^\star$ are i.i.d. according to $\boldsymbol{\xi}$. This allows us to use the standard law of large numbers to find the limit of $\frac{1}{t}\ln(\sigma_t/\sigma_0)$ as well as compute the expected log-step-size change.

Proposition 1. *Let* $\Delta_\sigma := \frac{1}{2d_\sigma n}\left(\mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right) - 1\right)$. *On linear functions, the* $(1, \lambda)$*-CSA-ES without cumulation satisfies (i) almost surely* $\lim_{t\to\infty} \frac{1}{t}\ln\left(\sigma_t/\sigma_0\right) = \Delta_\sigma$, *and (ii) for all* $t \in \mathbb{N}$, $\mathbb{E}(\ln(\sigma_{t+1}/\sigma_t)) = \Delta_\sigma$.

Proof. We have identified in Lemma 1 that the first coordinate of $\boldsymbol{\xi}_t^\star$ is distributed according to $\mathcal{N}_{1:\lambda}$ and the other coordinates according to $\mathcal{N}(0,1)$, hence $\mathbb{E}\left(\|\boldsymbol{\xi}_t^\star\|^2\right) = \mathbb{E}\left(\left[\boldsymbol{\xi}_t^\star\right]_1{}^2\right) + \sum_{i=2}^n \mathbb{E}\left(\left[\boldsymbol{\xi}_t^\star\right]_i^2\right) = \mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right) + n - 1$. Therefore $\mathbb{E}\left(\|\boldsymbol{\xi}_t^\star\|^2\right)/n - 1 = \left(\mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right) - 1\right)/n$. By applying this to Eq. (4), we deduce that $\mathbb{E}(\ln(\sigma_{t+1}/\sigma_t) = 1/(2d_\sigma n)(\mathbb{E}(\mathcal{N}_{1:\lambda}^2) - 1)$. Furthermore, as $\mathbb{E}(\mathcal{N}_{1:\lambda}^2) \leq \mathbb{E}((\lambda\mathcal{N}(0,1))^2) = \lambda^2 < \infty$, we have $\mathbb{E}(\|\boldsymbol{\xi}_t^\star\|^2) < \infty$. The sequence $(\|\boldsymbol{\xi}_t^\star\|^2)_{t\in\mathbb{N}}$ being i.i.d according to Lemma 1, and being integrable as we just showed, we can apply the strong law of large numbers on Eq. (5). We obtain

$$\frac{1}{t}\ln\left(\frac{\sigma_t}{\sigma_0}\right) = \frac{1}{2d_\sigma}\left(\frac{1}{t}\sum_{k=0}^{t-1}\frac{\|\boldsymbol{\xi}_k^\star\|^2}{n} - 1\right)$$
$$\xrightarrow[t\to\infty]{a.s.} \frac{1}{2d_\sigma}\left(\frac{\mathbb{E}\left(\|\boldsymbol{\xi}^\star\|^2\right)}{n} - 1\right) = \frac{1}{2d_\sigma n}\left(\mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right) - 1\right)$$

□

The proposition reveals that the sign of $\left(\mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right) - 1\right)$ determines whether the step-size diverges to infinity. In the following, we show that $\mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right)$ increases in λ for $\lambda \geq 2$ and that the $(1, \lambda)$-ES diverges for $\lambda \geq 3$. For $\lambda = 1$ and $\lambda = 2$, the step-size follows a random walk on the log-scale.

Lemma 2. *Let* $(\mathcal{N}_i)_{i\in[[1,\lambda]]}$ *be independent random variables, distributed according to* $\mathcal{N}(0,1)$, *and* $\mathcal{N}_{i:\lambda}$ *the* i^{th} *order statistic of* $(\mathcal{N}_i)_{i\in[[1,\lambda]]}$. *Then* $\mathbb{E}\left(\mathcal{N}_{1:1}^2\right) = \mathbb{E}\left(\mathcal{N}_{1:2}^2\right) = 1$. *In addition, for all* $\lambda \geq 2$, $\mathbb{E}\left(\mathcal{N}_{1:\lambda+1}^2\right) > \mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right)$.

Proof. (see [8] for the full proof) The idea of the proof is to use the symmetry of the normal distribution to show that for two random variables $U \sim \Psi_{1:\lambda+1}$ and $V \sim \Psi_{1:\lambda}$, for every event E_1 where $U^2 < V^2$, there exists another event E_2 counterbalancing the effect of E_1, i.e $\int_{E_2}(u^2 - v^2) f_{U,V}(u,v)\,\mathrm{d}u\,\mathrm{d}v = \int_{E_1}(v^2 - u^2) f_{U,V}(u,v)\,\mathrm{d}u\,\mathrm{d}v$, with $f_{U,V}$ the joint density of the couple (U, V). We then have $\mathbb{E}\left(\mathcal{N}_{1:\lambda+1}^2\right) \geq \mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right)$. As there is a non-negligible set of events E_3, distinct of E_1 and E_2, where $U^2 > V^2$, we have $\mathbb{E}(\mathcal{N}_{1:\lambda+1}^2) > \mathbb{E}(\mathcal{N}_{1:\lambda}^2)$.
For $\lambda = 1$, $\mathcal{N}_{1:1} \sim \mathcal{N}(0,1)$ so $\mathbb{E}(\mathcal{N}_{1:1}^2) = 1$. For $\lambda = 2$ we have $\mathbb{E}(\mathcal{N}_{1:2}^2 + \mathcal{N}_{2:2}^2) = 2\mathbb{E}(\mathcal{N}(0,1)^2) = 2$, and since the normal distribution is symmetric $\mathbb{E}(\mathcal{N}_{1:2}^2) = \mathbb{E}(\mathcal{N}_{2:2}^2)$, hence $\mathbb{E}(\mathcal{N}_{1:2}^2) = 1$. □

We can now link Proposition 1 and Lemma 2 into the following theorem:

Theorem 1. *On linear functions, for $\lambda \geq 3$, the step-size of the $(1, \lambda)$-CSA-ES without cumulation ($c = 1$) diverges geometrically almost surely and in expectation at the rate $1/(2d_\sigma n)(\mathbb{E}(\mathcal{N}_{1:\lambda}^2) - 1)$, i.e.*

$$\frac{1}{t} \ln \left(\frac{\sigma_t}{\sigma_0} \right) \xrightarrow[t\to\infty]{a.s.} \mathbb{E} \left(\ln \left(\frac{\sigma_{t+1}}{\sigma_t} \right) \right) = \frac{1}{2d_\sigma n} \left(\mathbb{E} \left(\mathcal{N}_{1:\lambda}^2 \right) - 1 \right) \quad . \tag{6}$$

For $\lambda = 1$ and $\lambda = 2$, without cumulation, the logarithm of the step-size does an additive unbiased random walk i.e. $\ln \sigma_{t+1} = \ln \sigma_t + W_t$ where $E[W_t] = 0$. More precisely $W_t \sim 1/(2d_\sigma)(\chi_n^2/n - 1)$ for $\lambda = 1$, and $W_t \sim 1/(2d_\sigma)((\mathcal{N}_{1:2}^2 + \chi_{n-1}^2)/n - 1)$ for $\lambda = 2$, where χ_k^2 stands for the chi-squared distribution with k degree of freedom.

Proof. For $\lambda > 2$, from Lemma 2 we know that $\mathbb{E}(\mathcal{N}_{1:\lambda}^2) > \mathbb{E}(\mathcal{N}_{1:2}^2) = 1$. Therefore $\mathbb{E}(\mathcal{N}_{1:\lambda}^2) - 1 > 0$, hence Eq. (6) is strictly positive, and with Proposition 1 we get that the step-size diverges geometrically almost surely at the rate $1/(2d_\sigma)(\mathbb{E}(\mathcal{N}_{1:\lambda}^2) - 1)$. With Eq. 4 we have $\ln(\sigma_{t+1}) = \ln(\sigma_t) + W_t$, with $W_t = 1/(2d_\sigma)(\|\boldsymbol{\xi}_t^\star\|^2/n - 1)$. For $\lambda = 1$ and $\lambda = 2$, according to Lemma 2, $\mathbb{E}(W_t) = 0$. Hence $\ln(\sigma_t)$ does an additive unbiased random walk. Furthermore $\|\boldsymbol{\xi}\|^2 = \mathcal{N}_{1:\lambda}^2 + \chi_{n-1}^2$, so for $\lambda = 1$, since $\mathcal{N}_{1:1} = \mathcal{N}(0, 1)$, $\|\boldsymbol{\xi}\|^2 = \chi_n^2$. □

In [8] we extend this result on the step-size to $|[\boldsymbol{X}_t]_1|$, which diverges geometrically almost surely at the same rate.

4 Divergence Rate of $(1, \lambda)$-CSA-ES with Cumulation

We are now investigating the $(1, \lambda)$-CSA-ES with cumulation, i.e. $0 < c < 1$. The path $\boldsymbol{\mathcal{P}}$ is then a Markov chain and contrary to the case where $c = 1$ we cannot apply a LLN for independent variables to Eq. (5) in order to prove the almost sure geometric divergence. However LLN for Markov chains exist as well, provided the Markov chain satisfies some stability properties: in particular, if the Markov chain $\boldsymbol{\mathcal{P}}$ is φ-irreducible, that is, there exists a measure φ such that every Borel set A of $\mathbb{R}^n$ with $\varphi(A) > 0$ has a positive probability to be reached in a finite number of steps by $\boldsymbol{\mathcal{P}}$ starting from any $\boldsymbol{p}_0 \in \mathbb{R}^n$. In addition, the chain $\boldsymbol{\mathcal{P}}$ needs to be (i) positive, that is the chain admits an invariant probability measure π, i.e., for any borelian A, $\pi(A) = \int_{\mathbb{R}^n} P(x, A)\pi(A)$ with $P(x, A)$ being the probability to transition in one time step from x into A, and (ii) Harris recurrent which means for any borelian A such that $\varphi(A) > 0$, the chain $\boldsymbol{\mathcal{P}}$ visits A an infinite number of times with probability one. Under those conditions, $\boldsymbol{\mathcal{P}}$ satisfies a LLN, more precisely:

Lemma 3. *[11, 17.0.1] Suppose that $\boldsymbol{\mathcal{P}}$ is a positive Harris chain with invariant probability measure π, and let g be a π-integrable function such that $\pi(|g|) = \int_{\mathbb{R}^n} |g(x)|\pi(dx) < \infty$. Then $1/t \sum_{k=1}^t g(\boldsymbol{p}_k) \xrightarrow[t\to\infty]{a.s.} \pi(g)$.*

The path $\boldsymbol{\mathcal{P}}$ satisfies the conditions of Lemma 3 and exhibits an invariant measure [8]. By a recurrence on Eq. (2) we see that the path follows the following equation

$$\boldsymbol{p}_t = (1-c)^t \boldsymbol{p}_0 + \sqrt{c(2-c)} \sum_{k=0}^{t-1} (1-c)^k \underbrace{\boldsymbol{\xi}^\star_{t-1-k}}_{\text{i.i.d.}} \quad . \tag{7}$$

For $i \neq 1$, $[\boldsymbol{\xi}^\star_t]_i \sim \mathcal{N}(0,1)$ and, as also $[\boldsymbol{p}_0]_i \sim \mathcal{N}(0,1)$, by recurrence $[\boldsymbol{p_t}]_i \sim \mathcal{N}(0,1)$ for all $t \in \mathbb{N}$. For $i = 1$ with cumulation ($c < 1$), the influence of $[\boldsymbol{p}_0]_1$ vanishes with $(1-c)^t$. Furthermore, as from Lemma 1 the sequence $([\boldsymbol{\xi}^\star_t]_1])_{t\in\mathbb{N}}$ is independent, we get by applying the Kolgomorov's three series theorem that the series $\sum_{k=0}^{t-1}(1-c)^k \left[\boldsymbol{\xi}^\star_{t-1-k}\right]_1$ converges almost surely. Therefore, the first component of the path becomes distributed as the random variable $[\boldsymbol{p}_\infty]_1 = \sqrt{c(2-c)}\sum_{k=0}^{\infty}(1-c)^k[\boldsymbol{\xi}^\star_k]_1$ (by re-indexing the variable $\boldsymbol{\xi}^\star_{t-1-k}$ in $\boldsymbol{\xi}^\star_k$, as the sequence $(\boldsymbol{\xi}^\star_t)_{t\in\mathbb{N}}$ is i.i.d.).

We now obtain geometric divergence of the step-size and get an explicit estimate of the expression of the divergence rate.

Theorem 2. *The step-size of the* $(1,\lambda)$*-CSA-ES with* $\lambda \geq 2$ *diverges geometrically fast if* $c < 1$ *or* $\lambda \geq 3$*. Almost surely and in expectation we have for* $0 < c \leq 1$*,*

$$\frac{1}{t}\ln\left(\frac{\sigma_t}{\sigma_0}\right) \underset{t\to\infty}{\longrightarrow} \frac{1}{2d_\sigma n} \underbrace{\left(2(1-c)\,\mathbb{E}\left(\mathcal{N}_{1:\lambda}\right)^2 + c\left(\mathbb{E}\left(\mathcal{N}^2_{1:\lambda}\right) - 1\right)\right)}_{>0 \text{ for } \lambda\geq 3 \text{ and for } \lambda=2 \text{ and } c<1} \quad . \tag{8}$$

Proof. For proving almost sure convergence of $\ln(\sigma_t/\sigma_0)/t$ we need to use the LLN for Markov chain. We refer to [8] for the proof that $\boldsymbol{\mathcal{P}}$ satisfies the right assumptions. We now focus on the convergence in expectation. From Eq. (4) we have $\mathbb{E}(\ln(\sigma_{t+1}/\sigma_t)) = c/(2d_\sigma)(\mathbb{E}(\|\boldsymbol{p}_{t+1}\|^2)/n - 1)$, so $\mathbb{E}(\|\boldsymbol{p}_{t+1}\|^2) = \mathbb{E}(\sum_{i=1}^n \left[\boldsymbol{p}_{t+1}\right]_i^2)$ is the term we have to analyse. From Eq. (7) and its conclusions we get that for $\mathrm{J} \neq 1$ $[\boldsymbol{p}_t]_j \sim \mathcal{N}(0,1)$, so $\mathbb{E}(\sum_{j=1}^n \left[\boldsymbol{p}_{t+1}\right]_j^2) = \mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^2) + (n-1)$. When t goes to infinity, the influence of $[\boldsymbol{p}_0]_1$ in this equation goes to 0 with $(1-c)^{t+1}$, so we can remove it when taking the limit:

$$\lim_{t\to\infty} \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^2\right) = \lim_{t\to\infty} \mathbb{E}\left(\left(\sqrt{c(2-c)}\sum_{i=0}^{t}(1-c)^i \left[\boldsymbol{\xi}^\star_{t-i}\right]_1\right)^2\right) \tag{9}$$

We will now develop the sum with the square, such that we have either a product $\left[\boldsymbol{\xi}^\star_{t-i}\right]_1 \left[\boldsymbol{\xi}^\star_{t-j}\right]_1$ with $i \neq j$, or $\left[\boldsymbol{\xi}^\star_{t-j}\right]_1^2$. This way, we can separate the variables by using Lemma 1 with the independence of $\boldsymbol{\xi}^\star_i$ over time. To do so, we use the development formula $(\sum_{i=1}^n a_n)^2 = 2\sum_{i=1}^n \sum_{j=i+1}^n a_i a_j + \sum_{i=1}^n a_i^2$. We take the limit of $\mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^2)$ and find that it is equal to

$$\lim_{t\to\infty} c(2-c)\left(2\sum_{i=0}^{t}\sum_{j=i+1}^{t}(1-c)^{i+j} \underbrace{\mathbb{E}\left(\left[\boldsymbol{\xi}^\star_{t-i}\right]_1 \left[\boldsymbol{\xi}^\star_{t-j}\right]_1\right)}_{=\mathbb{E}\left[\boldsymbol{\xi}^\star_{t-i}\right]_1 \mathbb{E}\left[\boldsymbol{\xi}^\star_{t-j}\right]_1 = \mathbb{E}[\mathcal{N}_{1:\lambda}]^2} + \sum_{i=0}^{t}(1-c)^{2i} \underbrace{\mathbb{E}\left(\left[\boldsymbol{\xi}^\star_{t-i}\right]_1^2\right)}_{=\mathbb{E}[\mathcal{N}^2_{1:\lambda}]}\right) \tag{10}$$

Now the expected value does not depend on i or j, so what is left is to calculate $\sum_{i=0}^{t}\sum_{j=i+1}^{t}(1-c)^{i+j}$ and $\sum_{i=0}^{t}(1-c)^{2i}$. We have $\sum_{i=0}^{t}\sum_{j=i+1}^{t}(1-c)^{i+j} = \sum_{i=0}^{t}(1-c)^{2i+1}\frac{1-(1-c)^{t-i}}{1-(1-c)}$ and when we separates this sum in two, the right hand side goes to 0 for $t \to \infty$. Therefore, the left hand side converges to $\lim_{t\to\infty}\sum_{i=0}^{t}(1-c)^{2i+1}/c$, which is equal to $\lim_{t\to\infty}(1-c)/c\sum_{i=0}^{t}(1-c)^{2i}$. And $\sum_{i=0}^{t}(1-c)^{2i}$ is equal to $(1-(1-c)^{2t+2})/(1-(1-c)^2)$, which converges to $1/(c(2-c))$. So, by inserting this in Eq. (10) we get that $\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^2\right) \underset{t\to\infty}{\longrightarrow} 2\frac{1-c}{c}\mathbb{E}\left(\mathcal{N}_{1:\lambda}\right)^2 + \mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right)$, which gives us the right hand side of Eq. (8).

By summing $\mathbb{E}(\ln(\sigma_{i+1}/\sigma_i))$ for $i = 0, \ldots, t-1$ and dividing by t we have the Cesaro mean $1/t\mathbb{E}(\ln(\sigma_t/\sigma_0))$ that converges to the same value that $\mathbb{E}(\ln(\sigma_{t+1}/\sigma_t))$ converges to when t goes to infinity. Therefore we have in expectation Eq. (8).

According to Lemma 2, for $\lambda = 2$, $\mathbb{E}(\mathcal{N}_{1:2}^2) = 1$, so the RHS of Eq. (8) is equal to $(1-c)/(d_\sigma n)\mathbb{E}(\mathcal{N}_{1:2})^2$. The expected value of $\mathcal{N}_{1:2}$ is strictly negative, so the previous expression is strictly positive. Furthermore, according to Lemma 2, $\mathbb{E}(\mathcal{N}_{1:\lambda}^2)$ increases with λ, as does $\mathbb{E}(\mathcal{N}_{1:2})^2$. Therefore we have geometric divergence for $\lambda \geq 2$. □

From Eq. (1) we see that the behavior of the step-size and of $(\boldsymbol{X}_t)_{t\in\mathbb{N}}$ are directly related. Geometric divergence of the step-size, as shown in Theorem 2, means that also the movements in search space and the improvements on affine linear functions f increase geometrically fast. Therefore, as we showed in Theorem 2 geometric divergence for the step-size when $\lambda \geq 2$ and $c < 1$, or when $\lambda \geq 3$, we expect geometric divergence on the first dimension of $(\boldsymbol{X}_t)_{t\in\mathbb{N}}$ (the first dimension being the only dimension with selection pressure). Analyzing $(\boldsymbol{X}_t)_{t\in\mathbb{N}}$ with cumulation requires to study a double Markov chain, which is left to possible future research.

5 Study of the Variations of $\ln(\sigma_{t+1}/\sigma_t)$

The proof of Theorem 2 shows that the step size increase converges to the right hand side of Eq. (8), for $t \to \infty$. When the dimension increases this increment goes to zero, which also suggests that it becomes more likely that σ_{t+1} is smaller than σ_t. To analyze this behavior, we study the variance of $\ln(\sigma_{t+1}/\sigma_t)$ as a function of c and the dimension.

Theorem 3. *The variance of* $\ln(\sigma_{t+1}/\sigma_t)$ *equals to*

$$\mathrm{Var}\left(\ln\left(\frac{\sigma_{t+1}}{\sigma_t}\right)\right) = \frac{c^2}{4d_\sigma^2 n^2}\left(\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^4\right) - \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^2\right)^2 + 2(n-1)\right) \quad . \quad (11)$$

Furthermore, $\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^2\right) \underset{t\to\infty}{\longrightarrow} \mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right) + \frac{2-2c}{c}\mathbb{E}\left(\mathcal{N}_{1:\lambda}\right)^2$ *and with* $a = 1-c$

$$\lim_{t\to\infty}\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^4\right) = \frac{(1-a^2)^2}{1-a^4}\left(k_4 + k_{31} + k_{22} + k_{211} + k_{1111}\right) \quad , \quad (12)$$

where $k_4 = \mathbb{E}\left(\mathcal{N}_{1:\lambda}^4\right)$, $k_{31} = 4\frac{a\left(1+a+2a^2\right)}{1-a^3}\mathbb{E}\left(\mathcal{N}_{1:\lambda}^3\right)\mathbb{E}\left(\mathcal{N}_{1:\lambda}\right)$, $k_{22} = 6\frac{a^2}{1-a^2}\mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right)^2$, $k_{211} = 12\frac{a^3(1+2a+3a^2)}{(1-a^2)(1-a^3)}\mathbb{E}\left(\mathcal{N}_{1:\lambda}^2\right)\mathbb{E}\left(\mathcal{N}_{1:\lambda}\right)^2$ *and* $k_{1111} = 24\frac{a^6}{(1-a)(1-a^2)(1-a^3)}\mathbb{E}\left(\mathcal{N}_{1:\lambda}\right)^4$.

Proof.

$$\mathrm{Var}\left(\ln\left(\frac{\sigma_{t+1}}{\sigma_t}\right)\right) = \mathrm{Var}\left(\frac{c}{2d_\sigma}\left(\frac{\|\boldsymbol{p}_{t+1}\|^2}{n} - 1\right)\right) = \frac{c^2}{4d_\sigma^2 n^2} \underbrace{\mathrm{Var}\left(\|\boldsymbol{p}_{t+1}\|^2\right)}_{\mathbb{E}\left(\|\boldsymbol{p}_{t+1}\|^4\right) - \mathbb{E}\left(\|\boldsymbol{p}_{t+1}\|^2\right)^2} \tag{13}$$

The first part of $\mathrm{Var}(\|\boldsymbol{p}_{t+1}\|^2)$, $\mathbb{E}(\|\boldsymbol{p}_{t+1}\|^4)$, is equal to $\mathbb{E}((\sum_{i=1}^n \left[\boldsymbol{p}_{t+1}\right]_i^2)^2)$. We develop it along the dimensions such that we can use the independence of $[\boldsymbol{p}_{t+1}]_i$ with $[\boldsymbol{p}_{t+1}]_j$ for $i \neq j$, to get $\mathbb{E}(2\sum_{i=1}^n \sum_{j=i+1}^n \left[\boldsymbol{p}_{t+1}\right]_i^2 \left[\boldsymbol{p}_{t+1}\right]_j^2 + \sum_{i=1}^n \left[\boldsymbol{p}_{t+1}\right]_i^4)$. For $i \neq 1$ $\left[\boldsymbol{p}_{t+1}\right]_i$ is distributed according to a standard normal distribution, so $\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_i^2\right) = 1$ and $\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_i^4\right) = 3$.

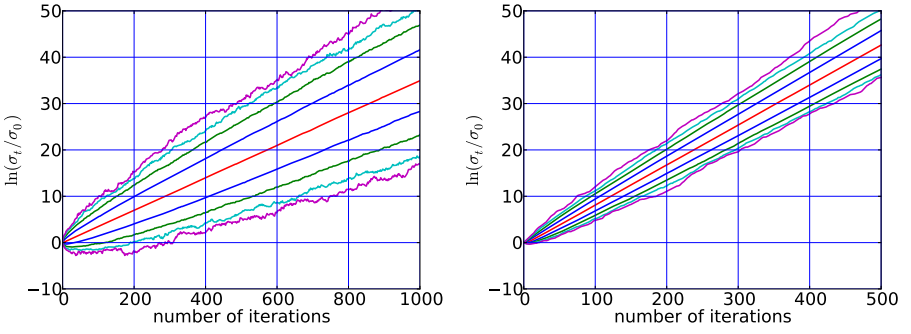
$$\begin{aligned}
\mathbb{E}\left(\|\boldsymbol{p}_{t+1}\|^4\right) &= 2\sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_i^2\right)\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_j^2\right) + \sum_{i=1}^n \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_i^4\right) \\
&= \left(2\sum_{i=2}^n \sum_{j=i+1}^n 1\right) + 2\sum_{j=2}^n \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^2\right) + \left(\sum_{i=2}^n 3\right) + \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^4\right) \\
&= \left(2\sum_{i=2}^n (n-i)\right) + 2(n-1)\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^2\right) + 3(n-1) + \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^4\right) \\
&= \mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^4\right) + 2(n-1)\mathbb{E}\left(\left[\boldsymbol{p}_{t+1}\right]_1^2\right) + (n-1)(n+1)
\end{aligned}$$

The other part left is $\mathbb{E}(\|\boldsymbol{p}_{t+1}\|^2)^2$, which we develop along the dimensions to get $\mathbb{E}(\sum_{i=1}^n \left[\boldsymbol{p}_{t+1}\right]_i^2)^2 = (\mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^2) + (n-1))^2$, which equals to $\mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^2)^2 + 2(n-1)\mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^2) + (n-1)^2$. So by subtracting both parts we get $\mathbb{E}(\|\boldsymbol{p}_{t+1}\|^4) - \mathbb{E}(\|\boldsymbol{p}_{t+1}\|^2)^2 = \mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^4) - \mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^2)^2 + 2(n-1)$, which we insert into Eq. (13) to get Eq. (11).

The development of $\mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^2)$ is the same than the one done in the proof of Theorem 2. We refer to [8] for the development of $\mathbb{E}(\left[\boldsymbol{p}_{t+1}\right]_1^4)$, since limits of space in the paper prevents us to present it here. □

Figure 1 shows the time evolution of $\ln(\sigma_t/\sigma_0)$ for 5001 runs and $c = 1$ (left) and $c = 1/\sqrt{n}$ (right). By comparing Figure 1a and Figure 1b we observe smaller variations of $\ln(\sigma_t/\sigma_0)$ with the smaller value of c.

Figure 2 shows the relative standard deviation of $\ln(\sigma_{t+1}/\sigma_t)$ (i.e. the standard deviation divided by its expected value). Lowering c, as shown in the left, decreases the relative standard deviation. To get a value below one, c must be smaller for larger dimension. In agreement with Theorem 3, In Figure 2, right, the relative standard deviation increases like $\sqrt{n}$ with the dimension for constant c (three increasing curves). A careful study [8] of the variance equation of Theorem 3 shows that for the choice of $c = 1/(1+n^\alpha)$, if $\alpha > 1/3$ the relative standard deviation converges to 0 with



(a) Without cumulation ($c = 1$) **(b)** With cumulation ($c = 1/\sqrt{20}$)

Fig. 1. $\ln(\sigma_t/\sigma_0)$ against t. The different curves represent the quantiles of a set of $5.10^3 + 1$ samples, more precisely the 10^i-quantile and the $1 - 10^{-i}$-quantile for i from 1 to 4; and the median. We have $n = 20$ and $\lambda = 8$.

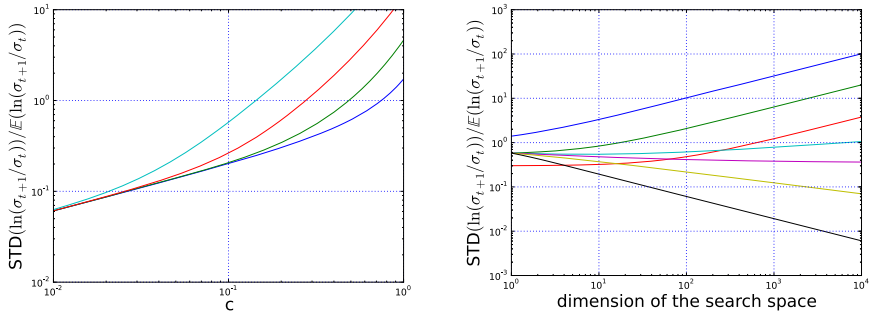


Fig. 2. Standard deviation of $\ln(\sigma_{t+1}/\sigma_t)$ relatively to its expectation. Here $\lambda = 8$. The curves were plotted using Eq. (11) and Eq. (12). On the left, curves for (right to left) $n = 2$, 20, 200 and 2000. On the right, different curves for (top to bottom) $c = 1$, 0.5, 0.2, $1/(1 + n^{1/4})$, $1/(1 + n^{1/3})$, $1/(1 + n^{1/2})$ and $1/(1 + n)$.

$\sqrt{(n^{2\alpha} + n)/n^{3\alpha}}$. Taking $\alpha = 1/3$ is a critical value where the relative standard deviation converges to $1/(\sqrt{2}\mathbb{E}(\mathcal{N}_{1:\lambda})^2)$. On the other hand, lower values of α makes the relative standard deviation diverge with $n^{(1-3\alpha)/2}$.

6 Summary

We investigate throughout this paper the $(1, \lambda)$-CSA-ES on affine linear functions composed with strictly increasing transformations. We find, in Theorem 2, the limit distribution for $\ln(\sigma_t/\sigma_0)/t$ and rigorously prove the desired behaviour of σ with $\lambda \geq 3$ for any c, and with $\lambda = 2$ and cumulation $(0 < c < 1)$: the step-size diverges geometrically fast. In contrast, without cumulation ($c = 1$) and with $\lambda = 2$, a random walk on $\ln(\sigma)$ occurs, like for the $(1, 2)$-σSA-ES [9] (and also for the same symmetry reason). We derive an expression for the variance of the step-size increment. On linear functions when

$c = 1/n^\alpha$, for $\alpha \geq 0$ ($\alpha = 0$ meaning c constant) and for $n \to \infty$ the standard deviation is about $\sqrt{(n^{2\alpha} + n)/n^{3\alpha}}$ times larger than the step-size increment. From this follows that keeping $c < 1/n^{1/3}$ ensures that the standard deviation of $\ln(\sigma_{t+1}/\sigma_t)$ becomes negligible compared to $\ln(\sigma_{t+1}/\sigma_t)$ when the dimensions goes to infinity. That means, the signal to noise ratio goes to zero, giving the algorithm strong stability. The result confirms that even the largest default cumulation parameter $c = 1/\sqrt{n}$ is a stable choice.

Acknowledgments. This work was partially supported by the ANR-2010-COSI-002 grant (SIMINOLE) of the French National Research Agency and the ANR COSINUS project ANR-08-COSI-007-12.

References

1. Arnold, D.V., Beyer, H.-G.: Performance analysis of evolutionary optimization with cumulative step length adaptation. IEEE Transactions on Automatic Control 49(4), 617–622 (2004)
2. Arnold, D.V., Beyer, H.-G.: On the behaviour of evolution strategies optimising cigar functions. Evolutionary Computation 18(4), 661–682 (2010)
3. Arnold, D.V.: Cumulative Step Length Adaptation on Ridge Functions. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN IX. LNCS, vol. 4193, pp. 11–20. Springer, Heidelberg (2006)
4. Arnold, D.V.: On the behaviour of the (1,λ)-es for a simple constrained problem. In: Foundations of Genetic Algorithms FOGA 2011, pp. 15–24. ACM (2011)
5. Arnold, D.V., Beyer, H.-G.: Random Dynamics Optimum Tracking with Evolution Strategies. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN VII. LNCS, vol. 2439, pp. 3–12. Springer, Heidelberg (2002)
6. Arnold, D.V., Beyer, H.G.: Optimum tracking with evolution strategies. Evolutionary Computation 14(3), 291–308 (2006)
7. Arnold, D.V., Beyer, H.G.: Evolution strategies with cumulative step length adaptation on the noisy parabolic ridge. Natural Computing 7(4), 555–587 (2008)
8. Chotard, A., Auger, A., Hansen, N.: Cumulative step-size adaptation on linear functions: Technical report (2012), http://hal.inria.fr/hal-00704903
9. Hansen, N.: An analysis of mutative σ-self-adaptation on linear fitness functions. Evolutionary Computation 14(3), 255–275 (2006)
10. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: International Conference on Evolutionary Computation, pp. 312–317 (1996)
11. Meyn, S.P., Tweedie, R.L.: Markov chains and stochastic stability, 2nd edn. Cambridge University Press (1993)
12. Ostermeier, A., Gawelczyk, A., Hansen, N.: Step-size Adaptation Based on Non-local Use of Selection Information. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN III. LNCS, vol. 866, pp. 189–198. Springer, Heidelberg (1994)

On the Behaviour of the $(1, \lambda)$-σSA-ES for a Constrained Linear Problem

Dirk V. Arnold

Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada B3H 4R2
dirk@cs.dal.ca

Abstract. This paper analyses the behaviour of the $(1, \lambda)$-σSA-ES with deterministic two-point rule when applied to a linear problem with a single linear constraint. Equations that describe the single-step behaviour of the strategy are derived and then used to predict the strategy's multi-step behaviour. The findings suggest that mutative self-adaptation will result in convergence of the $(1, \lambda)$-ES to non-stationary points if the angle between the gradient vector of the objective function and the normal vector of the constraint plane is small. Comparisons with the behaviour of evolution strategies that employ other step size adaptation mechanisms are drawn.

1 Introduction

Step size adaptation mechanisms and constraint handling techniques are important components of evolutionary algorithms (EAs) for constrained real valued optimisation. Most step size adaptation mechanisms have been devised with unconstrained optimisation in mind. Conversely, constraint handling techniques are often designed without much thought to their impact on step size adaptation. Schwefel [16] as early as the 1970s showed that a commonly employed step size adaptation mechanism may result in convergence to non-stationary points in an environment as simple as a linear problem with a single linear constraint.

An understanding of the interaction between step size adaptation mechanisms and constraint handling techniques is crucial for the design of EAs for constrained real valued optimisation. The *Handbook of Evolutionary Computation* [5, page B2.4:**11**f] lists a small number of studies that consider the behaviour of evolution strategies applied to simple constrained problems. Rechenberg [14] studies the performance of the $(1+1)$-ES[1] for the axis-aligned corridor model. Schwefel [16] considers the performance of the $(1, \lambda)$-ES in the same environment. Beyer [6] analyses the performance of the $(1 + 1)$-ES for a constrained, discus-like function. All of those have in common that the constraint planes are oriented such that their normal vectors are perpendicular to the gradient vector of the objective function. In contrast, Schwefel's work [16] suggests that convergence to

[1] See [9] for an explanation of the $(\mu/\rho \stackrel{+}{,} \lambda)$ terminology.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 82–91, 2012.

non-stationary points may occur in situations where the angle between those vectors, which we refer to as the constraint angle, is small. Studying the behaviour of EAs applied to a linear problem with a linear constraint of general orientation is fundamental as owing to Taylor's theorem, any smooth problem will appear increasingly linear as the step size of the strategy decreases. Arnold and Brauer [3] derive analytical results for the $(1 + 1)$-ES with success probability based step size adaptation and provide a quantitative confirmation of Schwefel's findings. More recent work [2, 1] analyses the behaviour of the $(1, \lambda)$-ES with cumulative step size adaptation for the constrained linear problem and compares two constraint handling techniques. It is found that convergence to non-stationary points in the face of small constraint angles is not unique to success probability based step size adaptation mechanisms.

The goal of this paper is to study the behaviour of the $(1, \lambda)$-σSA-ES, i.e., the $(1, \lambda)$-ES that employs mutative self-adaptation [16, 13] for step size control, when applied to a linear problem with a single linear constraint of general orientation. We assume that constraints are handled by resampling infeasible offspring candidate solutions. The work complements prior research that analyses the behaviour of mutative self-adaptation in unconstrained settings, including that by Hansen [10] who considers unconstrained linear problems, Beyer [7, 8] who considers spherically symmetric functions, and Meyer-Nieberg and Beyer [12] and Arnold and MacLeod [4] who consider ridge functions.

The remainder of this paper is organised as follows. Section 2 briefly describes the problem and the evolution strategy considered. Section 3 derives equations describing the single-step behaviour of the strategy. Section 4 considers multiple time steps and employs the balance criterion proposed by Lunacek and Whitley [11] in order to predict whether the strategy converges to a non-stationary point of the objective function. Section 5 concludes with a brief discussion of the findings and contrasts them with corresponding results for other step size adaptation mechanisms.

2 Problem and Algorithm

As in [3, 2, 1], throughout this paper we consider the problem of maximising[2] a linear function $f : \mathbb{R}^n \to \mathbb{R}$, $n \geq 2$, with a single linear constraint. We assume that the gradient vector of the objective function forms an acute angle with the normal vector of the constraint plane. Without loss of generality, we choose a Euclidean coordinate system with its origin located on the constraint plane, and with its axes oriented such that the x_1-axis coincides with the gradient direction ∇f, and the x_2-axis lies in the two-dimensional plane spanned by the gradient vector and the normal vector of the constraint plane. The angle between those two vectors is referred to as the constraint angle and denoted by θ as illustrated in Fig. 1. Constraint angles of interest are in the open interval $(0, \pi/2)$. The unit normal vector of the constraint plane expressed in the chosen

[2] Strictly speaking, the task is one of amelioration rather than maximisation, as a finite maximum does not exist. We do not make that distinction here.

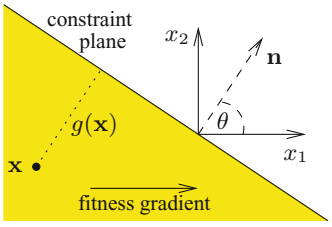


Fig. 1. Linear objective function with a single linear constraint. The subspace spanned by the x_1- and x_2-axes is shown. The shaded area is the feasible region. The parental candidate solution $\mathbf{x}$ of the $(1, \lambda)$-ES is at a distance $g(\mathbf{x})$ from the constraint plane.

coordinate system is $\mathbf{n} = \langle \cos\theta, \sin\theta, 0, \ldots, 0 \rangle$. The signed distance of a point $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in \mathbb{R}^n$ from the constraint plane is thus $g(\mathbf{x}) = -\mathbf{n} \cdot \mathbf{x} = -x_1 \cos\theta - x_2 \sin\theta$, resulting in the optimisation problem

$$\begin{aligned} &\text{maximise} \quad f(\mathbf{x}) = x_1 \\ &\text{subject to} \quad g(\mathbf{x}) = -x_1 \cos\theta - x_2 \sin\theta \geq 0 \ . \end{aligned}$$

Notice that due to the choice of coordinate system, variables $x_3, x_4, \ldots, x_n$ enter neither the objective function nor the constraint inequality.

Assuming a feasible initial candidate solution $\mathbf{x} \in \mathbb{R}^n$ and initial step size parameter $\sigma > 0$, the $(1, \lambda)$-σSA-ES generates a sequence of further candidate solutions by iterating the following three steps [16]:

1. Generate λ feasible offspring candidate solutions $\mathbf{y}^{(i)} = \mathbf{x} + \sigma\mathbf{z}^{(i)}$, $i = 1, \ldots, \lambda$, where the $\mathbf{z}^{(i)} \in \mathbb{R}^n$ are vectors with components drawn independently from normal distributions with mean zero and offspring dependent standard deviation ξ_i.
2. Evaluate $f(\mathbf{x}^{(i)})$ for $i = 1 \ldots, \lambda$ and let $(1; \lambda)$ denote index of the offspring candidate solution with the largest objective function value.
3. Replace the parental candidate solution and update the step size parameter according to

$$\begin{aligned} \mathbf{x} &\leftarrow \mathbf{y}^{(1;\lambda)} \\ \sigma &\leftarrow \sigma\xi_{1;\lambda} \ . \end{aligned}$$

Vectors $\mathbf{z}^{(i)}$ are referred to as mutation vectors, step size parameter σ is referred to as the mutation strength, and the ξ_i are referred to as step size modifiers. Notice that Step 1 may require generating more than λ offspring as infeasible candidate solutions are rejected immediately. However, for the problem under consideration on average no more than 2λ offspring need to be sampled per iteration.

The expected length of mutation vector $\mathbf{z}^{(i)}$ is proportional to step size modifier ξ_i. The underlying proposition of mutative self-adaptation is that if offspring candidate solutions are generated with differing expected lengths of their mutation vectors, then selection of appropriate step sizes becomes a by-product of evolution. Common choices for the distribution of the ξ_i include [15]:

log-normal: $\xi_i = \exp(\tau \mathcal{N}(0,1))$ where $\mathcal{N}(0,1)$ denotes a standard normally distributed random variate sampled anew for each i

two-point: $\xi_i = \beta > 1$ with probability one half and $\xi_i = 1/\beta$ otherwise

deterministic two-point: $\xi_i = \beta > 1$ if $1 \leq i \leq \lambda/2$ and $\xi_i = 1/\beta$ otherwise.

Constants τ (for log-normal) and β (for two-point and deterministic two-point) need to be chosen large enough to result in meaningful differences between the distributions of the offspring they control while being small enough not to render step size control excessively noisy. Rechenberg [15] recommends $\beta = 1.3$ for the two-point rule. As shown in the context of spherically symmetric functions, the log-normal and two-point operators can be made to behave very similarly if the parameters τ and β are chosen appropriately [8]. For simplicity, in this paper only the deterministic two-point operator is considered.

If the $(1, \lambda)$-ES is run on the constrained linear problem described above and the mutation strength σ is held constant, then the distance of the parental candidate solution from the constraint plane will assume a time-invariant distribution. Larger mutation strengths will result in faster progress. If the mutation strength is not fixed but instead allowed to vary under the control of some step size adaptation mechanism, then step sizes will either increase or decrease indefinitely. Decreasing step sizes result in convergence to a non-stationary point; increasing step sizes result in continually accelerating progress and are thus desirable.

3 Single-Step Behaviour

Let $\delta = g(\mathbf{x})/\sigma$ denote the normalised distance of the parental candidate solution $\mathbf{x}$ from the constraint plane. As infeasible offspring are resampled, the probability distribution of the z_1- and z_2-components of mutation vectors of feasible offspring candidate solutions generated with step size modifier ξ is a truncated normal distribution with joint density

$$p_{1,2}(x, y \,|\, \xi) = \begin{cases} \dfrac{1}{2\pi\xi^2 \Phi(\delta/\xi)} \mathrm{e}^{-\frac{1}{2}(x^2+y^2)/\xi^2} & \text{if } \delta \geq x\cos\theta + y\sin\theta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. The normalising term $\Phi(\delta/\xi)$ equals the probability that a randomly generated offspring candidate solution is feasible. The marginal density of the z_1-component is

$$\begin{aligned} p_1(x \,|\, \xi) &= \int_{-\infty}^{\infty} p_{1,2}(x, y \,|\, \xi)\, \mathrm{d}y \\ &= \frac{1}{\sqrt{2\pi}\xi \Phi(\delta/\xi)} \mathrm{e}^{-\frac{1}{2}(x/\xi)^2} \Phi\left(\frac{\delta - x\cos\theta}{\xi \sin\theta}\right) . \end{aligned} \tag{2}$$

We write $P_1(x \,|\, \xi, \delta)$ for the corresponding cumulative distribution function. The density of the z_2-component conditional on the value of the z_1-component is

$$p_2(y \,|\, z_1 = x, \xi) = \frac{p_{1,2}(x, y \,|\, \xi)}{p_1(x \,|\, \xi)} . \tag{3}$$

Integration of the probability density yields

$$P_2(y \,|\, z_1 = x, \xi) = \begin{cases} \dfrac{\Phi(y/\xi)}{\Phi((\delta - x\cos\theta)/(\xi \sin\theta))} & \text{if } y < \dfrac{\delta - x\cos\theta}{\sin\theta} \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

for the conditional cumulative distribution function of the z_2-component.

An important quantity to consider is the probability P_+ that the offspring candidate solution that is selected to replace the parent is one generated with step size modifier $\xi = \beta$ (as opposed to $\xi = 1/\beta$). That probability is of course also the probability that the step size of the strategy increases in the present step. As selection is based purely on the z_1-components of the mutation vectors, the cumulative distribution function of the z_1-component of the best of the $\lambda/2$ offspring candidate solutions generated with step size modifier ξ is

$$Q_\xi(x) = P_1^{\lambda/2}(x \,|\, \xi) \ .$$

The corresponding probability density function is

$$q_\xi(x) = \frac{\mathrm{d}}{\mathrm{d}x} Q_\xi(x) = \frac{\lambda}{2} p_1(x \,|\, \xi) P_1^{\lambda/2-1}(x \,|\, \xi) \ .$$

Probability P_+ is obtained by integrating the probability that the best offspring candidate solution generated with step size modifier β is superior to the best one generated with step size modifier $1/\beta$ and thus equals

$$P_+ = \int_{-\infty}^{\infty} q_\beta(x) Q_{1/\beta}(x) \,\mathrm{d}x \ . \tag{5}$$

Figure 2 illustrates how this probability depends on the normalised parental distance from the constraint plane and on the magnitude of the step size modifier. The plots have been generated from Eq. (5) and are for $\beta = 1.3$ in the left hand graph and for $\theta = \pi/8$ in the right hand one. If the parental candidate solution is far from the constraint plane, then the probability of generating an infeasible candidate solution that needs to be resampled is small and P_+ is independent of the constraint angle and exceeds one half. With decreasing distance from the constraint plane, P_+ decreases. Depending on the values of λ and θ it may either decrease below one half or remain above. Larger values of λ generally result in larger values of P_+. The curves in the right hand graph are monotonic and start at a value of one half for $\beta = 1$, suggesting that the choice of the step size modifier does not impact whether P_+ exceeds one half or not.

4 Multi-Step Behaviour

The results derived up to this point depend on the normalised distance δ of the parental candidate solution from the constraint plane. Assuming for now that

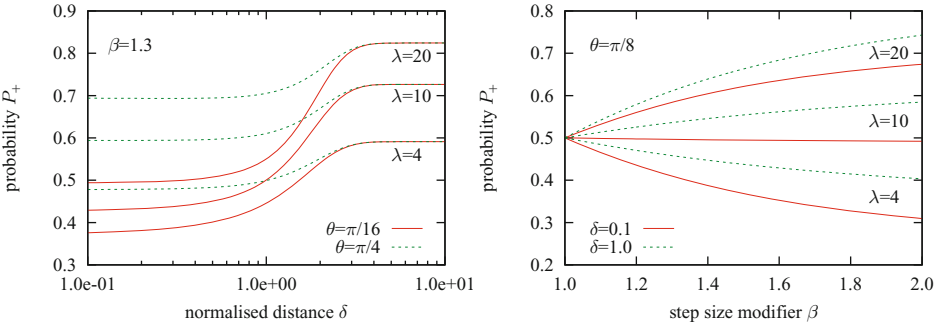


Fig. 2. Probability P_+ that a candidate solution generated with mutation strength modifier $\xi = \beta$ is selected as the next parental candidate solution plotted against the normalised parental distance δ from the constraint plane and against the magnitude of the step size modifier β

the mutation strength is fixed, if the strategy is iterated the normalised distance from the constraint plane evolves according to

$$\delta^{(t+1)} = \delta^{(t)} - z_1^{(1;\lambda)} \cos\theta - z_2^{(1;\lambda)} \sin\theta \tag{6}$$

where superscripts on δ denote time and those on z_1 and z_2 indicate the offspring candidate solution selected to replace the parent. The cumulative distribution function of $\delta^{(t+1)}$ conditional on $\delta^{(t)} = \delta$ is obtained using Eq. (6) by integrating the probability that $\delta^{(t+1)} < y$, yielding

$$\begin{aligned} P_\delta^{(t+1)}(y \,|\, \delta^{(t)} = \delta) &= \text{Prob}\left[\delta^{(t+1)} < y \,\middle|\, \delta^{(t)} = \delta\right] \\ &= \int_{-\infty}^{\infty} q_\beta(x) Q_{1/\beta}(x) \left[1 - P_2\left(\frac{\delta - y - x\cos\theta}{\sin\theta} \,\middle|\, z_1 = x, \xi = \beta\right)\right] \mathrm{d}x \\ &\quad + \int_{-\infty}^{\infty} q_{1/\beta}(x) Q_\beta(x) \left[1 - P_2\left(\frac{\delta - y - x\cos\theta}{\sin\theta} \,\middle|\, z_1 = x, \xi = 1/\beta\right)\right] \mathrm{d}x \end{aligned}$$

where conditional probability $P_2(\cdot|\cdot)$ is given in Eq. (4). Computing the derivative with respect to y yields conditional probability density $p_\delta^{(t+1)}(y \,|\, \delta^{(t)} = \delta)$.

For fixed mutation strength σ, the distance δ of the parental candidate solution from the constraint plane assumes a time-invariant limit distribution the density of which satisfies the evolution equation

$$p_\delta(y) = \int_0^\infty p_\delta(x) p_\delta(y|x) \,\mathrm{d}x \tag{7}$$

where the conditional density is that derived above. An approximation to the stationary limit distribution can be derived using the approach pursued by Beyer [8] in a different context: Expand the unknown distribution at time step t into a Gram-Charlier series with unknown cumulants. Then determine cumulants at

time step $t+1$ using Eq. (7). Considering cumulants up to the kth and imposing equality constraints on the cumulants yields a system of k equations in the k unknown cumulants. In the simplest case, only a single cumulant (the mean) is considered, and the equality constraint is

$$\mathrm{E}\left[\delta^{(t+1)} \,\middle|\, \delta^{(t)} = \delta\right] = \delta \tag{8}$$

which can be solved for the approximate average distance δ of the parental candidate solution from the constraint plane.

The expected distance from the constraint plane after a time step conditional on that distance before the time step is

$$\begin{aligned}\mathrm{E}\left[\delta^{(t+1)} \,\middle|\, \delta^{(t)} = \delta\right] &= \int_0^\infty y p_\delta^{(t+1)}(y \mid \delta^{(t)} = \delta)\,\mathrm{d}y \\ &= \frac{1}{\sin\theta}\int_{-\infty}^{\infty} q_\beta(x) Q_{1/\beta}(x) \int_0^\infty y p_2\left(\frac{\delta - y - x\cos\theta}{\sin\theta}\,\middle|\, z_1 = x, \xi = \beta\right)\mathrm{d}y\,\mathrm{d}x \\ &\quad + \frac{1}{\sin\theta}\int_{-\infty}^{\infty} q_{1/\beta}(x) Q_\beta(x) \int_0^\infty y p_2\left(\frac{\delta - y - x\cos\theta}{\sin\theta}\,\middle|\, z_1 = x, \xi = 1/\beta\right)\mathrm{d}y\,\mathrm{d}x\end{aligned}$$

where the conditional density $p_2(\cdot|\cdot)$ is given in Eq. (3). Solving the inner integrals yields expression

$$\begin{aligned}\mathrm{E}\left[\delta^{(t+1)} \,\middle|\, \delta^{(t)} = \delta\right] &= \int_{-\infty}^{\infty} \gamma_\beta(x) q_\beta(x) Q_{1/\beta}(x)\,\mathrm{d}x \\ &\quad + \int_{-\infty}^{\infty} \gamma_{1/\beta}(x) q_{1/\beta}(x) Q_\beta(x)\,\mathrm{d}x\end{aligned} \tag{9}$$

with

$$\gamma_\xi(x) = \delta - x\cos\theta + \frac{\xi\sin\theta}{\sqrt{2\pi}} \frac{\exp(-((\delta - x\cos\theta)/(\xi\sin\theta))^2/2)}{\Phi((\delta - x\cos\theta)/(\xi\sin\theta))}$$

for the expected distance from the constraint plane after a time step conditional on that distance before the time step.

Figure 3 illustrates how the average normalised distance from the constraint plane and the probability P_+ that an offspring candidate solution generated with the larger step size modifier replaces the parent depend on the constraint angle. The curves have been obtained by using Eq. (9) in Eq. (8) for $\beta = 1.3$, solving for δ, and using the result in Eq. (5). The dots mark measurements made in runs of the $(1, \lambda)$-ES with fixed step size. The average distance at which the constraint plane is tracked decreases with decreasing constraint angle and with increasing λ. The probability that an offspring candidate solution generated with step size modifier $\xi = \beta$ is selected to replace the parental candidate solution decreases with decreasing constraint angle and with decreasing λ. It exceeds one half for large constraint angles, but is below one half for small ones. The accuracy of the predictions made based on the simple stationarity requirement that considers the mean of the distribution only appears visually good for small constraint angles.

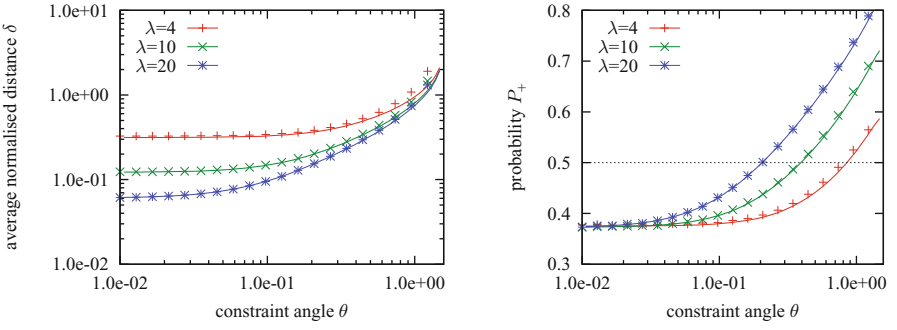


Fig. 3. Average normalised distance δ of the parental candidate solution from the constraint plane and probability P_+ of a candidate solution generated with step size modifier $\xi = \beta$ being selected to replace the parent plotted against constraint angle θ

If the mutation strength is not fixed but instead under the control of mutative self-adaptation, then, depending on the number of offspring λ generated per time step and the constraint angle θ, the strategy will either systematically reduce its step size and converge to a non-stationary point, or it will increase the step size and diverge. Clearly, for the constrained linear problem, which does not have a finite optimum, the latter is desirable. In order to establish whether convergence or divergence occurs, we employ the simple balance criterion proposed by Lunacek and Whitley [11] in the context of ridge functions. Specifically, we consider the probability P_+ that the mutation strength increases in the strategy's stationary state for fixed step size. If that probability exceeds one half, then divergence will occur; if it is below one half, then the strategy will converge.

Figure 4 illustrates how the minimum number of offspring required to avoid convergence to a non-stationary point depends on the constraint angle. The solid lines in both plots have been obtained by using Eq. (8) with $\beta = 1.3$ to determine the stationary δ, using Eq. (5) to obtain the corresponding P_+, and then determining the smallest λ such that P_+ exceeds one half. The points in the left hand graph mark measurements made in runs of the $(1, \lambda)$-σSA-ES. For each combination of λ and θ values considered, 100 runs of the strategy (initialised with the parental candidate solution on the constraint plane and $\sigma = 1$) were conducted until the mutation strength reached a value of either 10^{-20} (which is taken to be indicative of convergence) or 10^{20} (which is taken to be indicative of divergence). If at least 90 of the 100 runs yielded the same result, the location was marked with $\times$ (indicating convergence) or $+$ (indicating divergence). The quality of the predictions made on the basis of the simple stationarity and balance criteria is excellent.

The right hand graph in Fig. 4 contrasts results for the $(1, \lambda)$-σSA-ES with corresponding results for the $(1 + 1)$-ES with 1/5th-success rule [3] and the $(1, \lambda)$-ES with cumulative step size adaptation [2]. The latter curves correspond to, from top to bottom, values of the cumulation parameter of $c = 0.5$, 0.05, and 0.005. In contrast to the $(1+1)$-ES, the $(1, \lambda)$-σSA-ES is capable of avoiding

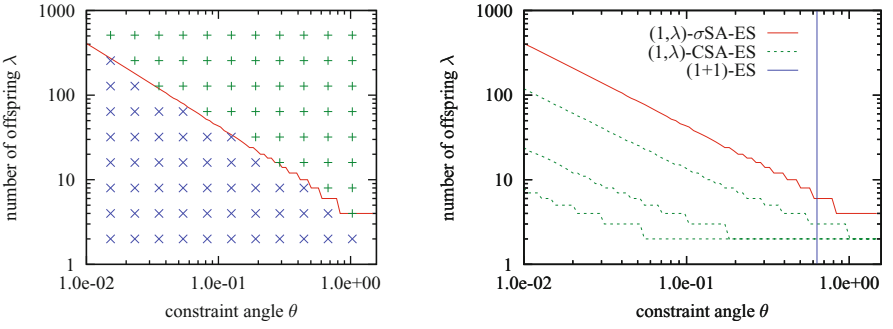


Fig. 4. Number of offspring λ per time step required to avoid convergence plotted against the constraint angle θ. The identical solid curve in both plots represents results for the $(1, \lambda)$-σSA-ES. The dashed lines in the right hand graph represent results for the $(1, \lambda)$-ES with cumulative step size adaptation and several values of the cumulation parameter. The vertical line in the right hand graph marks the constraint angle below which the $(1 + 1)$-ES converges to a non-stationary point.

convergence for any value of θ. However, the number of offspring per time step that is needed becomes very large as constraint angles become increasingly acute. (The plots suggest that the value of λ required is inversely proportional to θ.) In comparison, the $(1, \lambda)$-ES with cumulative step size adaptation manages to avoid convergence using significantly smaller values of λ.

5 Discussion and Future Work

To conclude, we have analysed the behaviour of the $(1, \lambda)$-σSA-ES for a linear problem with a single linear constraint of general orientation. A simple stationarity requirement has been used to approximate the average distance of the strategy from the constraint plane if the step size is fixed. The balance condition proposed by Lunacek and Whitley [11] has then been used to establish whether the adaptive strategy will converge to a non-stationary point or diverge. It has been found that divergence, which is the desirable behaviour, for increasingly acute constraint angles requires increasingly larger numbers of offspring generated per time step. Compared to the $(1, \lambda)$-ES with cumulative step size adaptation, for a given value of the constraint angle, the number of offspring required by the $(1, \lambda)$-σSA-ES is much higher.

There are multiple opportunities for further improving the understanding of the interaction of step size adaptation mechanisms and constraint handling techniques using the approach pursued here. Obvious extensions include considering the log-normal and two-point rules in place of the deterministic two-point rule, but differences are likely to be quantitative rather than qualitative. Regarding the $(\mu/\mu, \lambda)$-σSA-ES, it may be expected that the bias toward larger step sizes that results from the arithmetic averaging of mutation strengths has a

beneficial impact on the performance of the strategy, but the magnitude of that effect remains to be seen. Further future work includes the consideration of other constraint handling approaches, such as the simple repair mechanism previously considered for the $(1, \lambda)$-ES with cumulative step size adaptation [1], and of further constrained test problems.

Acknowledgements. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Arnold, D.V.: Analysis of a repair mechanism for the $(1, \lambda)$-ES applied to a simple constrained problem. In: Genetic and Evolutionary Computation Conference — GECCO 2011, pp. 853–860. ACM Press (2011)
[2] Arnold, D.V.: On the behaviour of the $(1, \lambda)$-ES for a simple constrained problem. In: Foundations of Genetic Algorithms 11, pp. 15–24. ACM Press (2011)
[3] Arnold, D.V., Brauer, D.: On the Behaviour of the (1+1)-ES for a Simple Constrained Problem. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN X. LNCS, vol. 5199, pp. 1–10. Springer, Heidelberg (2008)
[4] Arnold, D.V., MacLeod, A.: Step length adaptation on ridge functions. Evolutionary Computation 16(2), 151–184 (2008)
[5] Bäck, T., Fogel, D.B., Michalewicz, Z.: Handbook of Evolutionary Computation. Oxford University Press (1997)
[6] Beyer, H.-G.: Ein Evolutionsverfahren zur mathematischen Modellierung stationärer Zustände in dynamischen Systemen. PhD thesis, Hochschule für Architektur und Bauwesen, Weimar (1989)
[7] Beyer, H.-G.: Toward a theory of evolution strategies: Self-adaptation. Evolutionary Computation 3(3), 311–347 (1996)
[8] Beyer, H.-G.: The Theory of Evolution Strategies. Springer (2001)
[9] Beyer, H.-G., Schwefel, H.-P.: Evolution strategies — A comprehensive introduction. Natural Computing 1(1), 3–52 (2002)
[10] Hansen, N.: An analysis of mutative σ-self-adaptation on linear fitness functions. Evolutionary Computation 14(3), 255–275 (2006)
[11] Lunacek, M., Whitley, L.D.: Searching for Balance: Understanding Self-adaptation on Ridge Functions. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN IX. LNCS, vol. 4193, pp. 82–91. Springer, Heidelberg (2006)
[12] Meyer-Nieberg, S., Beyer, H.-G.: Mutative Self-adaptation on the Sharp and Parabolic Ridge. In: Stephens, C.R., Toussaint, M., Whitley, L.D., Stadler, P.F. (eds.) FOGA 2007. LNCS, vol. 4436, pp. 70–96. Springer, Heidelberg (2007)
[13] Meyer-Nieberg, S., Beyer, H.-G.: Self-adaptation in Evolutionary Algorithms. In: Lobo, F.G., Lima, C.F., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms. SCI, vol. 54, pp. 47–74. Springer, Heidelberg (2007)
[14] Rechenberg, I.: Evolutionsstrategie — Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Friedrich Frommann Verlag (1973)
[15] Rechenberg, I.: Evolutionsstrategie 94. Friedrich Frommann Verlag (1994)
[16] Schwefel, H.-P.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Birkhäuser Verlag (1977)

An Empirical Evaluation of O(1) Steepest Descent for NK-Landscapes

Darrell Whitley, Wenxiang Chen, and Adele Howe

Colorado State University, Fort Collins, CO, USA

Abstract. New methods make it possible to do approximate steepest descent in O(1) time per move for k-bounded pseudo-Boolean functions using stochastic local search. It is also possible to use the average fitness over the Hamming distance 2 neighborhood as a surrogate fitness function and still retain the $O(1)$ time per move. These are average complexity results. In light of these new results, we examine three factors that can influence both the computational cost and the effectiveness of stochastic local search: 1) Fitness function: $f(x)$ or a surrogate; 2) Local optimum escape method: hard random or soft restarts; 3) Descent strategy: next or steepest. We empirically assess these factors in a study of local search for solving NK-landscape problems.

Keywords: Stochastic Local Search, NK-Landscapes, Surrogate Fitness.

1 Introduction

The objective function for a number of combinatorial optimization problems, including MAX-kSAT [10] and NK-Landscapes [5], can be expressed as k-bounded pseudo-Boolean functions [7]. New results show that a form of approximate steepest descent can be implemented that requires on average $O(1)$ per move for k-bounded pseudo-Boolean functions [9].

Let X represent the set of candidate solutions, where each solution is a binary string of length N. Let $z \in X$ be the current solution. Let the function $N(z)$ generate the neighbors of solution z under the Hamming distance 1 "bit-flip" neighborhood. Thus, $x \in N(z)$ denotes that $x \in X$ is a neighbor of z. Typically, steepest descent Stochastic Local Search (SLS) requires $O(N)$ time. Using Walsh analysis it is possible to achieve an $O(1)$ average time complexity for each approximate steepest descent move [9].

The Walsh analysis makes it possible to compute the average fitness of the neighborhood reachable via each of the potential next moves. Viewing the search space as a tree rooted at the current solution z where x is a child of z such that $x \in N(z)$, it is also possible to compute neighborhood average of solutions reachable in two moves from vertex z via vertex x (i.e., these are the grandchildren of z that are also children of x).

$$Avg(N(x)) = 1/N \sum_{\mathbf{i}=1}^{N} f(y_i) \quad \text{where } y_i \in N(x) \text{ and } x \in N(z) \tag{1}$$

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 92–101, 2012.

Whitley and Chen [9] prove that approximate steepest descent using $Avg(N(x))$ as a surrogate function can execute in $O(1)$ time.

In this paper we explore several decisions related to designing an effective local search algorithm based on using $Avg(N(x))$. First, we explore whether there is any advantage to using $Avg(N(x))$ as a surrogate fitness function over using $f(x)$. We hypothesize that more plateaus are found in NKq-landscapes, which should favor the use of the $Avg(N(x))$ fitness function [1,8].

Second, a fundamental decision is what to do when a local optimum is encountered. One simple answer is to use a random restart. However, the proof of $O(1)$ complexity suggests that there is a significant advantage to using soft-restarts: executing a small number of random moves to escape a local optimum, where each move can be executed in $O(1)$ time. The use of soft restarts transforms the search into an Iterated Local Search algorithm [4]. Since a hard random restart requires a complete $O(N)$ reinitialization of the search, there is an advantage to using random walks to escape local optima.

Finally, our new theoretical results now make it possible to perform both *steepest* descent or *next* descent move selection in $O(1)$ time. Given that there is now no difference in cost, is there now an advantage to using *steepest* descent in place of *next* descent?

Given the constant time $O(1)$ move selection, we empirically evaluate the impact of critical design decisions on performance for k-bounded pseudo-Boolean NK-landscape and NKq-landscape problems. We control for runtime and assess solution quality.

2 The Theoretical $O(1)$ Result

This section briefly summarizes the theoretical results of Whitley and Chen [9]. Any discrete function $f : \{0,1\}^N \implies \mathbb{R}$ can be represented in the Walsh basis:

$$f(x) = \sum_{i=0}^{2^n-1} w_i \psi_i(x)$$

where w_i is a real-valued constant called a Walsh coefficient and $\psi_i(x) = -1^{i^T x}$ is a Walsh function that generates a sign. Alternatively, $\psi_i(x) = -1^{bitcount(i \wedge x)}$ where $bitcount(i \wedge x)$ counts how many 1-bits are in the string produced by the logical operation $i \wedge x$. The *order* of the i^{th} Walsh function is $bitcount(i)$.

Normally generating the Walsh coefficients requires that the entire search space be enumerated. However, Rana et al. [6] show for the MAX-kSAT problem that if a function is composed of subfunctions, each of which is a function over k Boolean variables, then the order of nonzero Walsh coefficients of the corresponding subfunction f_j is also bounded by 2^k. This result holds for all k-bounded pseudo-Boolean functions, including NK-Landscapes [2,3,7].

We will use a vector denoted by w' to store the Walsh coefficients, which will include the sign relative to solution x such that : $w'_i(x) = w_i \psi_i(x)$.

We assume the Walsh coefficients and their signs have been computed for some initial solution x. We will use b to index all of the Walsh coefficients in vector $w'(x)$. Let p be a string with exactly one bit set to 1. Let $p \subset b$ denote that bit p has value 1 in string b (i.e., $p \wedge b = p$). We can then compute the sum of all of the components in w' that are affected when local search flips bit p.

$$S_p(x) = \sum_{\forall b,\ p \subset b} w'_b(x)$$

Let $y_p \in N(x)$ be the neighbor of string x generated by flipping bit p. Then $f(y_p) = f(x) - 2(S_p(x))$ for all $y_p \in N(x)$. Assuming $y_i \in N(x)$ and $y_j \in N(x)$, then $f(x)$ can be viewed as a constant in the local neighborhood and

$$S_i(x) > S_j(x) \iff f(y_i) < f(y_j)$$

and thus selecting the maximal $S_i(x)$ yields the minimal neighbor of $f(x)$ [9].

If bit p is flipped, we must update element S_q by flipping the sign of the Walsh coefficients that are jointly indexed by p.

$$S_q(y_p) = S_q(x) - 2 \sum_{\forall b,(p \wedge q) \subset b} w'_b(x)$$

$$\forall b, \quad \text{if } (p \subset b) \quad \text{then } w'_b(y_p) = -w'_b(x) \quad \text{else } w'_b(y_p) = w'_b(x)$$

The vector S needs to be initialized after the first local search move. After that, only select elements of the vector S must be updated after a bit flip.

Whitley and Chen show that on average the expected number of elements in the vector S that must be updated is $O(1)$. The proof assumes those variables that appear more than $\mathbf{T}$ times across all subfunctions become Tabu after 1 flip, where $\mathbf{T}$ is a constant. All other variables can be flipped. A variable that is Tabu remains Tabu for N bit flips. The exact number of the updates is $k(k-1)+1 = O(1)$ when $\mathbf{T}$ is equal to the expected number of times a variable would appear in a randomly generated subfunction of an NK-landscape. However, empirical data suggest that no Tabu mechanism is necessary: during local search the waiting time between bit flips for variables that appear more than $\mathbf{T}$ times across all subfunctions is $\geq N$.

For constant time steepest descent, it also must be true that there cannot be too many unexploited improving moves. For example, assume that search starts from an extremum, and *every move is an improving move.* To do true steepest descent, we must use a priority queue in the form of a heap to select the best improving move, which results in $O(lg\ N)$ time to select the best improving move. However, in practice, there are typically few improving moves. We can implement approximate steepest descent as follows: assume that a threshold M is selected. We will store the location of improving moves in a buffer B. Let $|B|$ denote the number of elements stored in B. If $|B| \leq M$, then we will scan B and select the steepest descent improving move. If $|B| > M$, then we sample M moves from B (a form of "tournament selection") and select the best improving move from sample M. This yields an approximation to the steepest descent

improving move. In practice, we implemented true steepest descent because the number of improving moves is typically small [9].

$Avg(N(x))$ (see equation 1) can also be computed in $O(1)$ time on average. Let $\varphi'_{p,j}$ sum all Walsh coefficients of order j that reference bit p.

$$\varphi'_{p,j}(x) = \sum_{\forall b, bitcount(b)=j, p \subset b} w'_b(x)$$

We can now define the update for the vector S as follows, for $y_p \in N(x)$:

$$S_i(x) = \sum_{j=1}^{k} \varphi'_{i,j}(x) \quad \text{and} \quad S_i(y_p) = S_i(x) - 2 \sum_{\forall b, (p \wedge i) \subset b} w'_b(x) \tag{2}$$

We next define a new vector Z that computes a parallel result

$$Z_i(x) = \sum_{j=1}^{k} j\varphi'_{i,j}(x) \quad \text{and} \quad Z_i(y_p) = Z_i(x) - 2 \sum_{\forall b, (p \wedge i) \subset b} bitcount(b)(w'_b(x)) \tag{3}$$

where $bitcount(b)$ is a lookup table that stores the order of Walsh coefficient w'_b.

Whitley and Chen [9] prove that for any k-bounded pseudo-Boolean function, when flipping bit p and moving from solution z to solutions $x \in N(z)$:

$$Avg(N(x)) = Avg(N(z)) - 2S_p(z) + \frac{4}{N} Z_p(z) \tag{4}$$

Note that the vector S and Z are updated using exactly the same Walsh coefficients. Thus the cost of updating vector Z is the same as the cost of updating the vector S and the same $O(1)$ average time complexity holds for computing the approximate steepest descent move for $Avg(N(x))$.

3 Implementation Details

Algorithm 1, which we refer to as Walsh-LS, outlines the inputs which define 1) the fitness function to use (eval), 2) the descent method to use (descMeth), and 3) the escape scheme to use when a local optimum is reached (escape). Algorithm 2 implements the Update of the S, Z and w vectors.

DESCENT decides on the bit to be flipped. Improving moves are stored in the buffer. For the current experiments we implemented true steepest descent in which the index of the best improving bit is returned as bestI, with ties being broken randomly. For *next descent*, the first bit in buffer is returned.

ESCAPE METHOD is triggered when the algorithm reaches a local optimum. If escape is *random walk* then 10 bits are randomly flipped. Each bit flip requires an update to the S vector; the cost on average is still $O(1)$. If escape is *random restart* then the S vector must be reinitialized at $O(N)$ cost.

Algorithm 1. Sol ← WALSH-LS(eval, descMeth, escape)

```
1  bsfSol ← curSol ← INIT() ;              // current and best-so-far solutions
2  s, z, buffer ← WALSHANALYSIS(f, curSol) ;
3  while Termination criterion not met do
4      improve, bestI ← DESCENT(buffer, descMeth) ;
5      if improve == True then
6          w, s, z, buffer ← UPDATE(w, s, z, buffer, bestI, eval);
7          curSol ← FLIP(curSol, bestI) ;      // flips the bestI^th bit of curSol
8      else                                    // local optimum: perturbs current solution
9          bsfSol ← SELECT(curSol, bsfSol) ; // select sol with better fitness
10         curSol ← ESCAPE METHOD(curSol, escape) ;
11         for i in DifferentBit(bsfSol, curSol) do // for each different bit
12             w, s, z, buffer ← UPDATE(w, s, z, buffer, i, eval);
13 bsfSol ← SELECT(curSol, bsfSol) ;
14 return bsfSol
```

3.1 Experiment Design

Our experiments explore how the use of the surrogate fitness function interacts with the Descent and Escape methods. Based on our understanding of the surrogate fitness function, we posit two hypotheses.

1. The low complexity and the lookahead nature of the surrogate will support superior performance on problems where plateaus frequently appear in the

Algorithm 2. w, s, z, buffer ← UPDATE(w, s, z, buffer, p, eval)

```
1  s[p] ← -s[p];
2  for each q interacts with p do // update s vector
3      for each w[i] touching both p and q do
4          s[q] ← s[q] - 2 * w[i];
5  if eval is f(x) then
6      for each s[i] touching p do // update buffer
7          if s[i] is an improving move then BUFFER ← APPEND(buffer, i);
8  else                                                  // eval is Avg(N(f))
9      z[p] ← -z[p];
10     for each q interacts with p do // update z vector
11         for each w[i] touching both p and q do
12             z[q] ← z[q] - 2 * ORDER(w[i]) * w[i];
13     for each z[i] touching p do // update buffer
14         if z[i] is an improving move then BUFFER ← APPEND(buffer, i);
15 For each w[i] touching p do w[i] ← -w[i]     \\ update Walsh coefficients
```

fitness landscape, as is the case for NKq-landscapes. Because of the low complexity of evaluating the possible moves, the effect may be independent of descent method, but because different descents may lead to different portions of the search space, it would not be surprising to find an interaction effect.
2. Performing an $O(1)$ cost random walk upon reaching a local optimum should provide an advantage over an $O(N)$ cost hard random restart. This effect should become more pronounced as the problem size increases.

We chose NKq-landscapes with q=2 to increase the occurrence of plateaus. We limited the values of K to 2, 4 and 8 and N to 50, 100, 200 and 500 for two reasons. First, using the fast Walsh update is only advantageous when $N \gg 2^k$. Second, for $K > 8$ the random nature of the subfunctions used in NK-landscapes makes such problems become increasingly random and disordered as K increases. We randomly generated 24 problems, one for each combination of problem type, K and N value. Each combination of factors was run for 30 trials on each problem. The computational platform was Fedora 16 using Xeon Processor E5450 at 3.00GHz. All algorithms are implemented in Python 2.7.2.

To control for the amount of computation done in each configuration, we counted the number of times that UPDATE is executed and terminated a trial when the number is $100 \times N$. Normally the number of fitness evaluations would be counted. However, Walsh-LS requires only partial fitness evaluation (line 6 and line 12 in Algorithm 1) and we count only the updates that need to be recomputed. This gives a clear advantage to random walk over random restart as an escape mechanism. A single random restart uses N updates because $O(N)$ of the elements stored in vector S must be updated. A random walk of length 10 does 10 updates to the S vector.

3.2 Solution Quality

Without a baseline, it can be difficult to know whether differences in performance are meaningful. Observed quality may be near optimal (producing a floor effect) or far away, suggesting considerable room for improvement. In addition, observed differences may be small or mitigated by variance in results across trials. To establish a "best" solution, we run SLS with steepest descent and random walk for both fitness functions 50 times longer than in the normal experiments and harvest the overall $Best$ solution. Then we normalize solution quality to the value $(\frac{f(x)-Best}{Best})$ for each problem instance.

Figure 1 shows the normalized quality of solutions. Some results are within a fraction of the baseline solution while other are 6 times worse. The variance in solution quality was approximately 10 times greater on NKq-landscapes compared to NK-landscapes. For each combination of K and NK/NKq instances, we ran two-way ANOVAs with problem size N and algorithm as the independent variables and solution quality as the dependent. All main effects and interaction effects were significant at $p < .00001$ level, which indicates distinctions between algorithms' performance that varies with problem size.

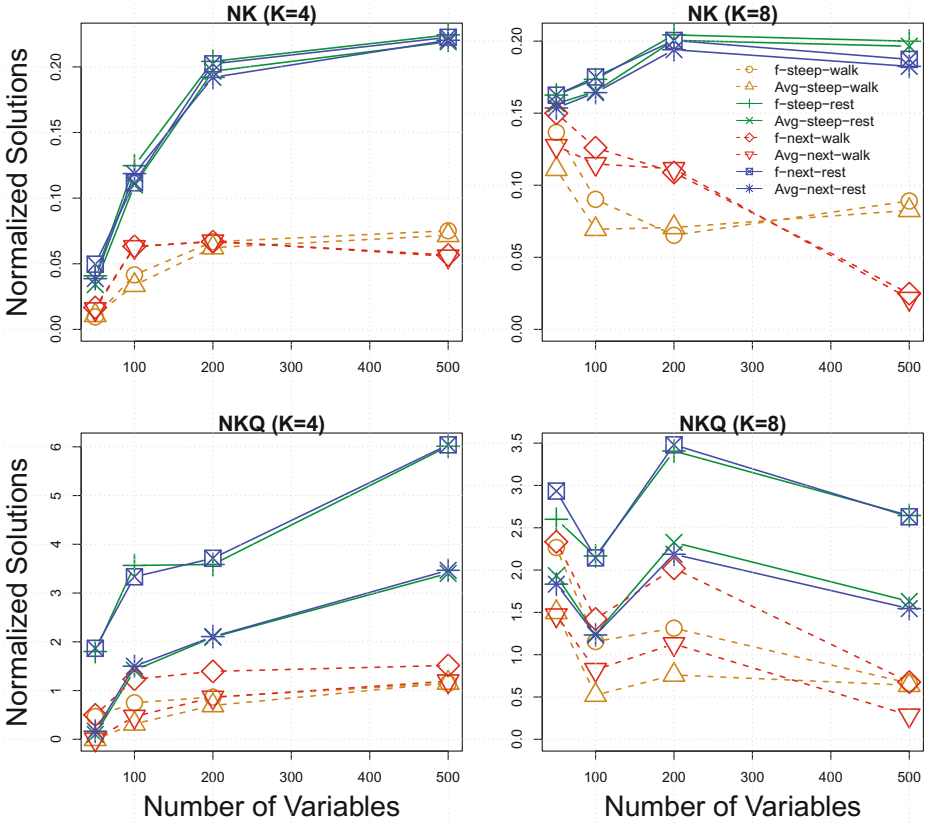


Fig. 1. Normalized Solutions, averaged across trials, found by Walsh-LS across factors. Upper graphs show NK-landscape problems; lower show NKq-landscape problems.

In Figure 1, dashed lines are for configurations that employed a random walk; solid lines employed hard random restarts. As expected, hard random restarts produced poorer results compared to the soft-restarts using a random walk to escape local minima, and the effect appears independent of other design decisions. Table 1 shows that Walsh-LS with random walk visits more local optima and usually visits more *distinct* local optima than Local Search with random restart. Even short random walks are not simply returning back where they started. The ratio of distinct local optima to total local optima suggests that a walk length of 20 might be better than a walk length of 10.

Tables 2 and 3 show means and standard deviations of fitness evaluations. We ran Wilcoxon rank-sum tests for two values of N and two values of K using only random walk for escape and comparing f to Avg in each case. For NK landscapes, statistical tests indicate that in all cases but one the differences are not significant (at the p=0.05 level) when using $Avg(N(x))$ vs. $f(x)$. But for $N = 100$ and K=8 we find p=0.007 for the steepest descent case.

Table 1. The number of local optima visited by Walsh-LS with steepest descent for problems with $N = 100$, $K = 2$ and $q = 2$. The "Total" rows indicate the number of local optima encountered; "Distinct" rows show the number of those that are unique. We used a Walk Length of 10 in all experiments, but present various Walk Lengths here to show the impact on the number of optima visited.

Problem	Eval	# of Locals	Walsh-LS random restarts	Walk Length 10	20	30	40	50
NK-Landscape	$f(x)$	Total	326	619	460	387	344	322
		Distinct	326	219	422	385	343	322
	$Avg(N(x))$	Total	324	622	465	383	345	322
		Distinct	324	234	431	382	345	322
NKq-Landscape	$f(x)$	Total	431	697	553	484	431	427
		Distinct	431	659	552	484	431	427
	$Avg(N(x))$	Total	325	671	516	420	363	324
		Distinct	324	347	481	414	356	323

Table 2. Means and standard deviations of fitness evaluations for *NK* problems, organized by configurations of fitness function and descent method (using only random walk for escape), best values in bold.

Descent	Eval	K=4 N=100	K=4 N=500	K=8 N=100	K=8 N=500
steepest	f	.231 ± .003	.223 ± .004	.231 ± .006	.238 ± .005
	Avg	**.229** ± .004	**.222** ± .004	**.226** ± .006	**.236** ± .005
next	f	.236 ± .004	.219 ± .003	.238 ± .006	.224 ± .005
	Avg	.236 ± .003	.219 ± .003	**.236** ± .006	**.223** ± .003

Table 3. Means and standard deviations of fitness evaluations for *NKq* problems, organized by configurations of fitness function and descent method (using only random walk for escape), best values in bold.

Descent	Eval	K=4 N=100	K=4 N=500	K=8 N=100	K=8 N=500
steepest	f	.035 ± .007	.039 ± .006	.065 ± .010	.060 ± .005
	Avg	**.026** ± .005	.039 ± .005	**.046** ± .006	**.059** ± .006
next	f	.045 ± .008	.045 ± .004	.073 ± .010	.060 ± .005
	Avg	**.029** ± .004	**.039** ± .003	**.055** ± .009	**.046** ± .004

For NKq problems, the advantage of utilizing $Avg(N(X))$ is clearer. The results using $Avg(N(x))$ are better than $f(x)$ in most cases. In all but two cases, $p < 0.0001$; for $N = 500$ when steepest descent is used the p values are $p = 0.9$ for $K = 4$ and $p = 0.5$ for $K = 8$.

3.3 Runtime Results

The number of "updates" (and thus the number of "moves") was used to control termination. A random restart is implemented as a series of "moves" from the current solution to the new randomly generated solution, which has $O(N)$

complexity. Thus the total execution time used by hard random restarts and the random walk soft restarts is basically the same. There was also virtually no difference in the steepest descent and the next descent runtimes; next descent was faster, but only by a small insignificant amount.

In our current implementation, computing $Avg(N(x))$ requires the use of both the S and Z vectors; Thus the cost of computing $Avg(N(x))$ was approximately twice the cost of computing $f(x)$ for each move.

However, to be more efficient note that

$$\begin{aligned} Avg(N(x_p)) &= Avg(N(z)) - 2S_p(z) + \frac{4}{N}Z_p(z) && \text{by Eqn 4} \\ &= Avg(N(z)) - 2(\sum_{j=1}^{k}\varphi'_{z,j}(x)) + \frac{4}{N}\sum_{j=1}^{k} j\varphi'_{z,j}(x) && \text{by Eqn 2,3} \\ &= Avg(N(z)) + \sum_{j=1}^{k}\frac{(4j-2N)}{N}\varphi'_{z,j}(x) \end{aligned}$$

Construct a new vector $w_i^*(x) = \frac{(4j-2N)}{N}w'(x) = \frac{(4j-2N)}{N}w(x)\psi_i(x)$.

Let $Z_p^*(z) = \frac{4}{N}Z_p(z) - 2S_p(z)$ which yields: $Avg(N(x)) = Avg(N(z)) + Z_p^*(z)$

Using the update rules for vectors S and Z when bit p is flipped:

$$\begin{aligned} Z_i^*(y_p) &= \frac{4}{N}Z_i(y_p) - 2S_i(y_p) \\ &= \frac{4}{N}[Z_i(x) - 2\sum_{\forall b,(p\wedge i)\subset b} j * w'_b(x)] - 2[S_i(x) - 2\sum_{\forall b,(p\wedge i)\subset b} w'_b(x)] \\ &= Z_i^*(x) - 2\sum_{\forall b,(p\wedge i)\subset b}\frac{4j-2N}{N}w'_b(x) \\ &= Z_i^*(x) - 2\sum_{\forall b,(p\wedge i)\subset b} w_b^*(x) \end{aligned}$$

Thus, $Avg(N(x))$ can be computed in precisely the same number of updates needed to compute $f(x)$ and $Z^*(x)$ can directly be used as a proxy for $Avg(N(x))$. Furthermore $f(x)$ can be efficiently computed on demand given $Avg(N(x))$ [7].

4 Conclusions

In light of new methods that allow steepest and next descent to be implemented in $O(1)$ time, we evaluated the impact of the fitness function, the descent method and the method used to escape local optima on NK and NKq landscapes. Not surprisingly, random walks perform much better than random restarts; using random walks in place of random restarts transforms the algorithm into an Iterated Local Search algorithm [4].

Somewhat more surprising, we find little or no difference between steepest and next descent methods. This could be due to the fact that NK-landscapes have little inherent structure; the results might differ in other domains.

Finally, for NKq landscapes, using $Avg(N(x))$ as the evaluation function instead of $f(x)$ generally improved performance. This appears to be because $Avg(N(x))$ results in fewer plateaus and fewer local optima compared to $f(x)$.

Acknowledgments. This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-11-1-0088. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

1. Geard, N.: A comparison of neutral landscapes: NK, NKp and NKq. In: IEEE Congress on Evolutionary Computation (CEC 2002), pp. 205–210 (2002)
2. Heckendorn, R., Whitley, D.: A Walsh analysis of NK-landscapes. In: Proceedings of the Seventh International Conference on Genetic Algorithms (1997)
3. Heckendorn, R.B.: Embedded landscapes. Evolutionary Computation 10(4), 345–369 (2002)
4. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2004)
5. Kauffman, S., Weinberger, E.: The NK Model of Rugged Fitness Landscapes and its Application to Maturation of the Immune Response. Journal of Theoretical Biology 141(2), 211–245 (1989)
6. Rana, S.B., Heckendorn, R.B., Whitley, L.D.: A tractable Walsh analysis of SAT and its implications for genetic algorithms. In: Mostow, J., Rich, C. (eds.) AAAI/IAAI, pp. 392–397. AAAI Press / The MIT Press (1998)
7. Sutton, A.M., Whitley, L.D., Howe, A.E.: Computing the moments of k-bounded pseudo-Boolean functions over Hamming spheres of arbitrary radius in polynomial time. Theoretical Computer Science 425, 58–74 (2012)
8. Verel, S., Ochoa, G., Tomassini, M.: Local optima networks of NK landscapes with neutrality. IEEE Transactions on Evolutionary Computation 14(6), 783–797 (2010)
9. Whitley, D., Chen, W.: Constant Time Steepest Ascent Local Search with Statistical Lookahead for NK-Landscapes. In: GECCO 2012: Proceedings of the Annual Conference on Genetic and Evolutionary Computation Conference (2012)
10. Zhang, W.: Configuration landscape analysis and backbone guided local search: part i: Satisfiability and maximum satisfiability. Artificial Intelligence 158, 1–26 (2004)

Experimental Supplements to the Computational Complexity Analysis of Genetic Programming for Problems Modelling Isolated Program Semantics

Tommaso Urli[1], Markus Wagner[2], and Frank Neumann[2]

[1] DIEGM, Università degli Studi di Udine, 33100 Udine, Italy
[2] School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia

Abstract. In this paper, we carry out experimental investigations that complement recent theoretical investigations on the runtime of simple genetic programming algorithms [3, 7]. Crucial measures in these theoretical analyses are the maximum tree size that is attained during the run of the algorithms as well as the population size when dealing with multi-objective models. We study those measures in detail by experimental investigations and analyze the runtime of the different algorithms in an experimental way.

Keywords: genetic programming, problem complexity, multiple-objective optimization, experimental evaluation.

1 Introduction

In the last decade, Genetic Programming algorithms have found various applications [8] in a number of domains, however their behaviour is hard to understand in a rigorous manner. Recently, the first computational complexity results have been presented for simple genetic programming algorithms [3, 7]. The algorithms that have been considered are a stochastic hill-climber called (1+1)-GP and a population-based multi-objective programming algorithm called SMO-GP that takes into account the given problem F and the complexity C of a solution. These algorithms have been analyzed on problems with isolated program semantics taken from [5] which can be seen as the analogue of linear pseudo-Boolean functions [2] known from the computational complexity analysis of evolutionary algorithms working with fixed length binary representations.

The theoretical results provided in [3, 7] bring up several questions that remain unanswered in these papers. In particular, for different combinations of algorithms and problems no (or no exact) runtime bounds are given. In our paper, we explore the different open cases and questions in an experimental way. Similar to [1, 6], this should guide further rigorous analyses by exploring the important measures within a computational complexity analysis of the algorithms and give experimental estimates on the actual runtime of the algorithms on the different problems. Our experimental investigations, will concentrate on

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 102–112, 2012.

important measures such as the maximum tree size during the run of the single-objective algorithms analyzed in [3] and the maximum population size of the multi-objective algorithm analyzed in [7]. It can be observed from the analyses carried out in these two papers, that both measures have a different implication on the runtime of the analyzed genetic programming algorithms. Other experimental results indicate that both measures do not grow large during the run of the algorithms which would imply a fast optimization process. Furthermore, our experimental results on the actual runtime of (1+1)-GP and SMO-GP indicate an efficient optimization process.

The paper is structured as follows. In Section 2, we introduce the problems and algorithms and summarize the computational complexity results for them. (1+1)-GP is experimentally investigated in Section 3 and the behavior of SMO-GP is examined in Section 4. We finish with some concluding remarks.

2 Preliminaries

In our experimental investigations, we will treat the algorithms and problems analyzed in [3, 7]. We consider tree-based genetic programming, where a possible solution to a given problem is given by a syntax tree. The inner nodes of such a tree are labelled by symbols from a function set F, and the leaves of the tree are labelled by terminals from a set T. The problems that we examine are Weighted ORDER (WORDER) and Weighted MAJORITY (WMAJORITY). For all, the only function is the binary join operation (denoted by J), and the terminal set is a set of $2n$ variables, where $\overline{x}_i$ represents the complement of x_i. Thus, $F := \{J\}$ and $T := \{x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_n, \overline{x}_n\}$. With each variable x_i, we associate a weight $w_i \in \mathbb{R}$, $1 \leq i \leq n$. Thus, the variables can differ in their contribution to the fitness of a tree. Without loss of generality, we assume that $w_1 \geq w_2 \geq \ldots \geq w_n \geq 0$.

For a given syntax tree X, its computed value S is obtained by parsing the syntax tree in-order according to the problem semantics. For WORDER, x_i is contained in S iff it is present in the tree and there is no $\overline{x}_i$ that is visited in the in-order parse before x_i. For WMAJORITY, x_i is in S iff x_i occurs in the tree at least once, and at least as often as its complement $\overline{x}_i$ (see Algorithms 1 and 2). The weight w_i of a variable x_i contributes to the fitness iff x_i is positive and contained in set S. We get the problems ORDER and MAJORITY as special cases where $w_i = 1$, $1 \leq i \leq n$, holds.

Algorithm 1. WORDER(X)

input: a syntax tree X
init : an empty leaf list l, an empty statement list S
1 Parse X in-order and insert each leaf the rear of l as it is visited;
2 Generate S by parsing l front to rear and adding a leaf to S only if its complement is not yet in S;
3 WORDER $(X) = \sum_{x_i \in S} w_i$;

Algorithm 2. WMAJORITY(X)

input: a syntax tree X
init : an empty leaf list l, an empty statement list S
1 Parse X in-order and insert each leaf the the rear of l as is is visited;
2 For $1 \leq i \leq n$: if count$(x_i \in l) \geq$ count$(\overline{x}_i \in l)$ and count$(x_i \in l) \geq 1$, then add x_i to S;
3 WMAJORITY $(X) = \sum_{x_i \in S} w_i$;

As GP mechanisms, we investigate the single-objective (1+1)-GP and the Simple Multi-Objective Genetic Programming (SMO-GP) algorithm. For the (1+1)-GP, we consider the problem of computing a solution X which maximizes a given function $F(X)$. In the case of the parsimony approach, we additionally take into account the complexity $C(X)$ of a solution (measured as the total number of nodes in the tree). Here, we optimize the multi-criteria fitness function MO-F$(X) = (F(X), C(X))$ with respect to the lexicographic order, that is, MO-F$(X) \geq$ MO-F(Y) holds iff $F(X) > F(Y) \vee (F(X) = F(Y) \wedge C(X) \leq C(Y)$.

For SMO-GP, we will treat the two objective $F(X)$ and $C(X)$ as equally important and use standard notations from the field of multi-objective optimization. A solution Y *weakly dominates* a solution X (denoted by $Y \succeq X$) iff $(F(Y) \geq F(X) \wedge C(Y) \leq C(X))$. A solution Y *dominates* a solution X (denoted by $Y \succ X$) iff $((Y \succeq X) \wedge (F(Y) > F(X) \vee C(Y) < C(X))$. A *Pareto optimal solution* is a solution that is not dominated by any other solution in the search space. All Pareto optimal solutions together form the Pareto optimal set, and the set of corresponding objective vectors forms the Pareto front. The classical goal in multi-objective optimization is to compute for each objective vector of the Pareto front a Pareto optimal solution. SMO-GP starts with a single solution and keeps at any time during the optimization a set of non-dominated solutions among the set of all solutions seen so far.

Note that this trade-off between solution complexity and solution quality has successfully applied in industry tools such as Datamodeller [4].

(1+1)-GP and SMO-GP only use the mutation operator HVL-Prime to generate offspring. HVL-Prime allows for the production of trees of varying complexity, and is based on the operations *substitution*, *deletion*, and *insertion*. For an application of HVL-Prime, a parameter k has to be chosen. k determines the number of operations that HVL-Prime performs: (1) in the *single-operation* case $k = 1$ holds, (2) in the *multi-operation* case $k = 1 + \text{Pois}(1)$ holds, where Pois(1) denotes the Poisson distribution with parameter 1. We refer the reader to [3, 7] for a detailed description on HVL-Prime. Depending on the number of operations used in the mutation operator, we get the algorithms (1+1)-GP-single and SMO-GP-single and their corresponding multi-mutation variants (1+1)-GP-multi and SMO-GP-multi.

The complete algorithms are outlined in Algorithms 3 and 4.

Algorithm 3. (1+1)-GP

```
1 Choose an initial solution X;
2 repeat
3     Set Y := X;
4     Apply mutation to Y;
5     If selection favors Y over X then
      set X := Y;
```

Algorithm 4. SMO-GP

```
1 Choose an initial solution X;
2 Set P := {X};
3 repeat
4     Randomly choose X ∈ P;
5     Set Y := X;
6     Apply mutation to Y;
7     If {Z ∈ P|Z ≻ Y} = ∅ set
      P := (P \ {Z ∈ P|Z ⪰ Y}) ∪ {Y};
```

2.1 Theoretical Results

The computational complexity analysis of genetic programming analyzes the expected number of fitness evaluations until an algorithm has produced an optimal solution for the first time. This is called the *expected optimization time*. In the case of multi-objective optimization the number of fitness evaluations until the whole Pareto front has been computed is analyzed and referred to as the expected optimization time. The bounds from [3, 7] are listed in Table 1. As it can be seen, all results take into account tree sizes of some kind: either the maximum tree size T_{max} during the search plays a role in the bound, or the size of the initial tree T_{init} does. It is also unknown how tight the given bounds are. The maximum tree size for (1+1)-GP and the population size for SMO-GP play a relevant role in the theoretical analysis and will be further investigated in the rest of the paper. Lastly, note that the upper bounds marked with $\star$ hold only if the algorithm has been initialized in the particular, i.e. non-redundant, way described in [7].

Table 1. Computational complexity results from [3, 7]

F(X)	(1+1)-GP, F(X) [3]		(1+1)-GP, MO-F(X) [7]		SMO-GP, MO-F(X) [7]	
	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)
ORD	$O(nT_{max})$	$O(nT_{max})$	$O(T_{init} + n \log n)$	?	$O(nT_{init} + n^2 \log n)$	
WORD	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?
MAJ	$O(n^2 T_{max} \log n)$	?	$O(T_{init} + n \log n)$		$O(nT_{init} + n^2 \log n)$	
WMAJ	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?

2.2 Experimental Setup

In the remainder of this paper, we will empirically confirm and verify the theoretical results from [3, 7]. We consider (1+1)-GP and SMO-GP, each in their single and multi-operation variants, and investigate problems of sizes n =20, 40, 60, ..., 200 (although, for space reasons, the results in tables are shown only for $n = 100$). For the initialization, we consider the schemes init_0 (empty tree) and init_{2n} (in which a $2n$ leaves tree is generated by applying $2n$ *insertion* mutations at random positions). In total, our experiments span twelve problems: WORDER and WMAJORITY in their F(X) and MO-F(X) variants. The weight settings are set as follows: (1) $w_i = 1$, $1 \leq i \leq n$, for ORDER and MAJORITY, (2) $w_i \in [0, 1]$ chosen uniformly at random, $1 \leq i \leq n$, for WORDER-RAN and WMAJORITY-RAN, and (3) $w_i = 2^{n-i}$, $1 \leq i \leq n$, for WORDER-BIN and WMAJORITY-BIN.

The following experiments were performed on AMD Opteron 250 CPUs (2.4GHz), on Debian GNU/Linux 5.0.8, with Java SE RE 1.6 and were given a maximum runtime of 3 hours and a budget of 10^9 evaluations. Furthermore, each experiment has been repeated 400 times, which results in a standard error of the mean (the standard deviation of the sampling distribution) of $1/\sqrt{400} = 5\%$.

3 (1+1)-GP

3.1 Tree size

The theoretical bounds for (1+1)-GP on ORDER and MAJORITY presented in [3] depend on the maximum tree size that is encountered during the run of the algorithms. We investigate the maximum tree size experimentally in order to see whether bloat occurs when applying the algorithms. For (1+1)-GP-single using the parsimony approach, i. e. using the function MO-F(X), the difference between the solution value S and the number of leaves not preceded by their complements can not increase during the run of the algorithm [7].

First, we investigate the tree sizes typically observed during the optimization for the different (1+1)-GP algorithms. Table 2 reports results for $n = 100$, but similar results hold for the other input sizes. The maximum tree size observed during the run of (1+1)-GP on MO-F(X) when using single-operation and empty initialization is $2n-1$, which is the minimum possible size of an optimal solution. This was expected, since the algorithm can only increase the tree by a single leaf in every accepting step. These values increase by about 10-20% in the case of init_0, when multiple HVL-Prime applications are allowed per mutation step. When the acceptance criteria is weakened by switching to the F(X) variant (i.e. the current tree can be replaced by larger ones of identical fitness), then the tree sizes are about 2.5 times larger in the single-operation case, and about 3 times larger in the multi-operation case.

Similarly, when running (1+1)-GP on MO-F(X), if the population is initialized with trees of $2n$ leaves, the largest trees encountered are of size $2 \cdot (2n) - 1$, i.e. the tree size of the initial solution, in the single-operation case, and are just minimally larger (about 1%) in the multi-operation case.

Table 2. Maximum tree sizes encountered until the individual X_{max} with maximum fitness is found. Shown are median m and median interquartile ranges iqr. Here $k = 1$ and $k = 1 + Pois(1)$ refer respectively to the single and multi-operation variants.

k	F(X)	n	(1+1)-GP, F(X)				(1+1)-GP, MO-F(X)			
			init_0		init_{2n}		init_0		init_{2n}	
			m	iqr	m	iqr	m	iqr	m	iqr
k=1	ORDER	100	519	94.5	593	100	199	0	399	2
	WORDER-RAN	100	513	85	594	90	199	0	399	0.5
	WORDER-BIN	100	513	94	591	88.5	199	0	399	0
	MAJORITY	100	507	78.5	563	72	199	0	399	0
	WMAJORITY-RAN	100	499	76.5	567	74.5	199	0	399	0
	WMAJORITY-BIN	100	499	74.5	567	75	199	0	399	0
k=1+Pois(1)	ORDER	100	670	138	742	143	223	12	399	6
	WORDER-RAN	100	667	136.5	713	131	229	12	399	6
	WORDER-BIN	100	665	150.5	735	132.5	231	12	399	4
	MAJORITY	100	624	96	668	102	239	14	401	8
	WMAJORITY-RAN	100	617	104	678	116.5	241	16	401	8
	WMAJORITY-BIN	100	635	114.5	671	116.5	243	14	401	8

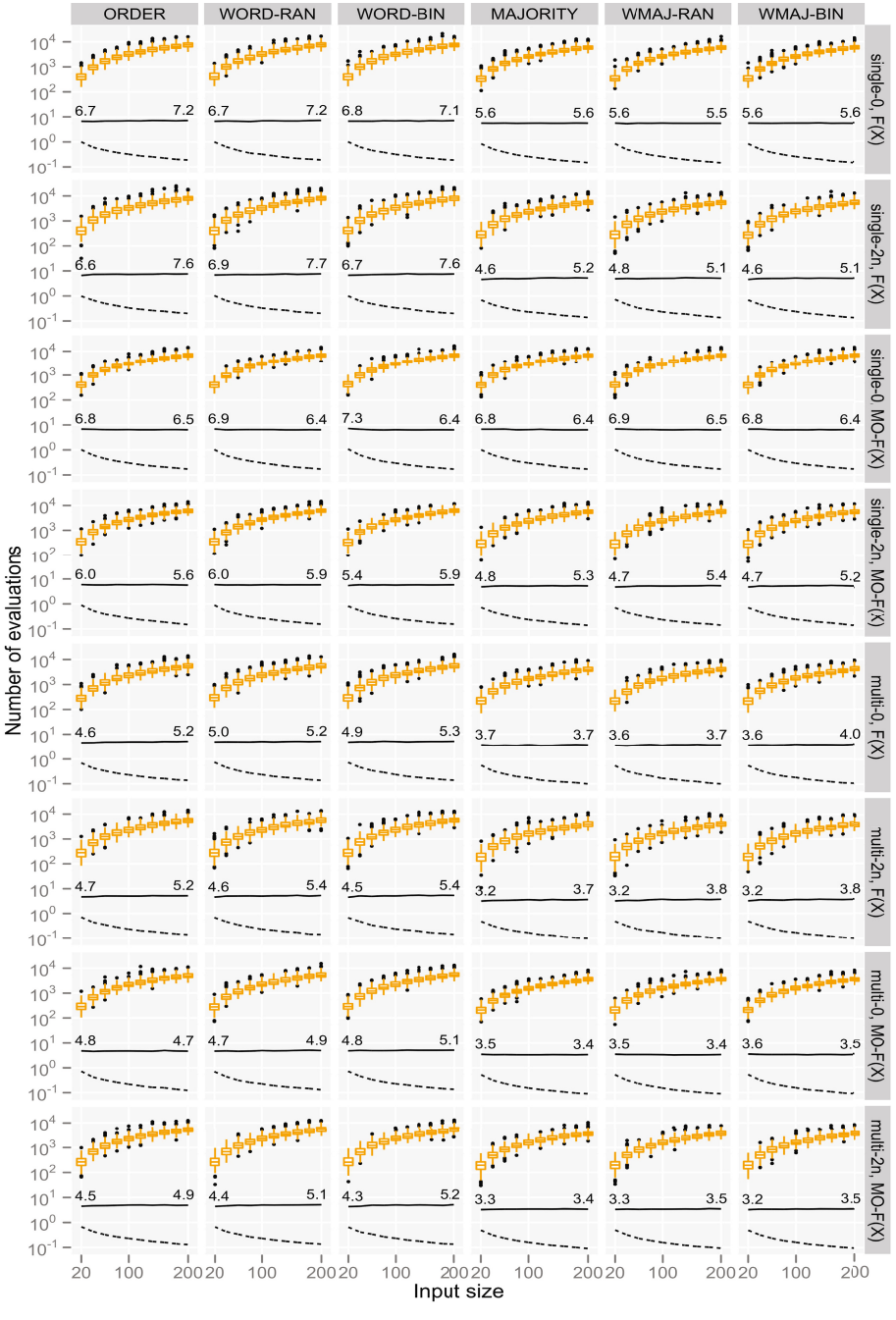


Fig. 1. Number of evaluations required by (1+1)-GP until the individual X_{max} with maximum fitness is found, shown as box plots. The solid line is the median of the number of evaluations divided by $n \log n$, the dashed line is the same median divided by n^2.

For the non-parsimony variants, however, the largest trees are about 50% larger when solving ORDER, and almost 100% when solving MAJORITY.

3.2 Runtime

Figure 1 shows the distributions of the required evaluations for the (1+1)-GP variants as box plots. The line plots represent the medians divided by different polynomials and suggest the asymptotic behavior of the algorithms: the solid line is the median number of evaluations needed to produce the individual with the optimal fitness value divided by $n \log n$, and the dashed line is the same number, but divided by n^2.

For all combinations of algorithms and problems these plots indicate an expected optimization time of $O(n \log n)$, as the solid lines closely resemble constant functions (see the y-values for $n = 20$ and $n = 200$), and the y-values of the dashed lines are decreasing with increasing values of n. The constant factor obtained by dividing the median number of evaluations by $n \log n$ is overall higher in the single-operation variants of the algorithm, suggesting that applying multi mutations can help getting earlier to the optimal solution.

One important observation is that the algorithms' asymptotic behavior appears to be same, when initialized with the empty tree, and with trees with $2n$ leaves. For the setups where a theoretical bound in Table 1 is missing, the experimental results give a strong indication about the expected optimization time being $O(n \log n)$.

4 SMO-GP

4.1 Tree Size and Population Size

Table 3 shows the maximum tree sizes and maximum population sizes that were observed up to the following two events. Firstly, until the individual X_{max} with maximum fitness is found, and secondly, until the population represents the entire true Pareto front P_{Pareto}.

It can be seen that tree and population sizes observed by SMO-GP-single are independent of the initialization. In all cases, no trees with more than the size of the Pareto optimal solution X with $F(X) = n$ (size $2n - 1$) (when using init_0) and the initial tree size $2 \cdot (2n) - 1$ (when using init_{2n}) ever belong to the population. In the multi-operation cases, the maximum population sizes are rarely higher, and the same holds for the maximum tree sizes.

4.2 Runtime

Just as in the previous section, we show now the distributions of the required evaluations as box plots in Figure 2. As before, yellow box plots represent the number of evaluations to get to X_{max}, while red box plots represent now the number of evaluations to get to P_{Pareto}. In this plot, the lines are the medians

Table 3. Maximum tree sizes and maximum population sizes encountered for SMO-GP on the MO-F(X) problem variants: (1) until the individual X_{max} with maximum fitness is found, (2) until the population represents the entire true Pareto front P_{Pareto}. Shown are median m and interquartile ranges iqr. init_0 denotes the initialization with the empty tree, and init_{2n} the one with randomly constructed trees with $2n$ leaf nodes.

		F(X)	n	maximum tree size				max. population size			
				to X_{max}		to P_{Pareto}		to X_{max}		to P_{Pareto}	
				m	iqr	m	iqr	m	iqr	m	iqr
SMO-GP, with k=1	init_0	ORDER	100	199	0	199	0	101	0	101	0
		WORDER-RAN	100	199	0	199	0	101	0	101	0
		WORDER-BIN	100	199	0	199	0	101	0	101	0
		MAJORITY	100	199	0	199	0	101	0	101	0
		WMAJORITY-RAN	100	199	0	199	0	101	0	101	0
		WMAJORITY-BIN	100	199	0	199	0	101	0	101	0
	init_{2n}	ORDER	100	399	0	399	0	101	0	101	0
		WORDER-RAN	100	399	0	399	0	101	0	101	0
		WORDER-BIN	100	399	0	399	0	101	0	101	0
		MAJORITY	100	399	0	399	0	101	0	101	0
		WMAJORITY-RAN	100	399	0	399	0	101	0	101	0
		WMAJORITY-BIN	100	399	0	399	0	101	0	101	0
SMO-GP, with k=1+Pois(1)	init_0	ORDER	100	207	6.5	207	6.5	101	0	101	0
		WORDER-RAN	100	211	8	211	8	102	2	102	1
		WORDER-BIN	100	211	6	211	6	102	1	102	2
		MAJORITY	100	215	10.5	215	10.5	101	0	101	0
		WMAJORITY-RAN	100	223	12	223	12	103	2	103	1
		WMAJORITY-BIN	100	219	10	219	10	102	2	103	2
	init_{2n}	ORDER	100	399	4	399	4	101	0	101	0
		WORDER-RAN	100	399	4	399	4	102	2	102	1
		WORDER-BIN	100	399	4	399	4	102	1	102	2
		MAJORITY	100	399	4	399	4	101	0	101	0
		WMAJORITY-RAN	100	400	6	400	6	103	2	104	2
		WMAJORITY-BIN	100	401	6	401	6	102	2	103	1

divided by different polynomials and suggest the asymptotic behavior of the algorithms: the solid line is the median number of evaluations needed to get to the Pareto front divided by $n^2 \log n$, and the dashed line is the same number, but divided by n^3. For all combinations of algorithms and the problems, these plots indicate an expected optimization time of $O(n^2 \log n)$ for ORDER and MAJORITY, as the solid lines closely resemble constant functions, and the y-values of the dashed lines are decreasing with increasing values of n. For the weighted variants, however, the solid lines appear to be slowly rising, indicating a runtime in $\Omega(n^2 \log n) \cap O(n^3)$, although the runtime is extremely close to $O(n^2 \log n)$.

Furthermore, it can be observed that there is a significant time difference, for SMO-GP-multi, between finding the individual with the optimal fitness value and finding the entire Pareto front. For SMO-GP-single, this time difference is negligible, which is the reason why the corresponding orange box plots are scarcely identifiable behind the red ones.

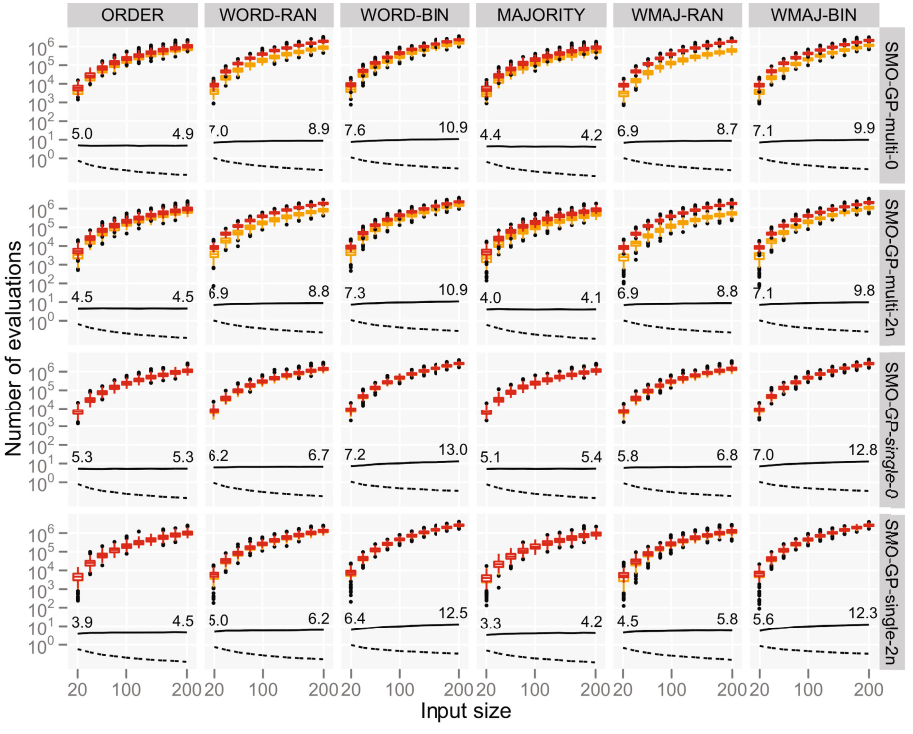


Fig. 2. Shown as box plots is the number of evaluations required: (1) until the individual X_{max} with maximum fitness is found (orange), (2) until the population represents the entire true Pareto front P_{Pareto} (red). The solid line is the median of the latter number of evaluations divided by $n^2 \log n$, the dashed line is it divided by n^3.

5 Conclusions

In this paper, we carried out experimental investigations to complement recent theoretical results on the runtime of two genetic programming algorithms [3, 7]. Crucial measures in these theoretical analyses are the maximum tree size that is attained during the run of the algorithms, as well as the population size when dealing with multi-objective models. Furthermore, virtually no theoretical results for the multi-operation variants are known to date. It is also unknown how tight the given bounds are. The analysis of our empirical investigations allowed us to fill in the gaps in the theory with conjectures about the expected optimization time (see Tables 4 and 5) of these algorithms.

Our experimental evaluation shows that the expected optimization time of (1+1)-GP F(X) is very close to $O(n \log n)$. Our results, however, are based on an initial tree size, i.e. T_{init}, which is always linear in n, and thus the T_{init} term suggested by theoretical results is always dominated by the $O(n \log n)$ term. Nevertheless, it is easy to show that by using arbirarily large initial tree sizes it is possible to obtain expected optimization times in which the T_{init} term is

Table 4. Summary of our conjectures (†) and the existing upper bounds from Table 1

F(X)	(1+1)-GP, F(X)		(1+1)-GP, MO-F(X)	
	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)
ORDER	$O(nT_{max})$ [3] $O(T_{init} + n \log n)$ †	$O(nT_{max})$ [3] $O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †
WORDER	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †
MAJORITY	$O(n^2 T_{max} \log n)$ [3] $O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †
WMAJORITY	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †

Table 5. Summary of our conjectures (†) and the existing upper bounds from Table 1

F(X)	SMO-GP, MO-F(X)	
	k=1	k=1+Pois(1)
ORDER	$O(nT_{init} + n^2 \log n)$[7]	$O(nT_{init} + n^2 \log n)$[7]
WORDER	$O(n^3)\star$ [7] $O(nT_{init} + n^2 \log n)$ †	$O(nT_{init} + n^2 \log n)$ †
MAJORITY	$O(nT_{init} + n^2 \log n)$[7]	$O(nT_{init} + n^2 \log n)$[7]
WMAJORITY	$O(n^3)\star$ [7] $O(nT_{init} + n^2 \log n)$ †	$O(nT_{init} + n^2 \log n)$ †

relevant. For this reason we conjecture an expected optimization time of $O(T_{init} + n \log n)$. Following the same reasoning for SMO-GP, we conjecture a runtime of $O\left(nT_{init} + n^2 \log n\right)$ by noting that the observed runtimes are very close to $O(n^2 \log n)$ and that the algorithm has to evolve a population of size $O(n)$.

As a further development for this line of research, it would be interesting to prove these conjectured bounds theoretically and to show how they are related to maximum population size reached during an optimization run.

References

[1] Briest, P., Brockhoff, D., Degener, B., Englert, M., Gunia, C., Heering, O., Jansen, T., Leifhelm, M., Plociennik, K., Röglin, H., Schweer, A., Sudholt, D., Tannenbaum, S., Wegener, I.: Experimental Supplements to the Theoretical Analysis of EAs on Problems from Combinatorial Optimization. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN VIII. LNCS, vol. 3242, pp. 21–30. Springer, Heidelberg (2004)

[2] Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science 276, 51–81 (2002)

[3] Durrett, G., Neumann, F., O'Reilly, U.-M.: Computational complexity analysis of simple genetic programing on two problems modeling isolated program semantics. In: FOGA, pp. 69–80. ACM (2011)

[4] Evolved Analytics LLC. DataModeler 8.0. Evolved Analytics LLC (2010)
[5] Goldberg, D.E., O'Reilly, U.-M.: Where Does the Good Stuff Go, and Why? How Contextual Semantics Influences Program Structure in Simple Genetic Programming. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 16–36. Springer, Heidelberg (1998)
[6] Lässig, J., Sudholt, D.: Experimental Supplements to the Theoretical Analysis of Migration in the Island Model. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 224–233. Springer, Heidelberg (2010)
[7] Neumann, F.: Computational complexity analysis of multi-objective genetic programming. In: GECCO. ACM (to be published, 2012); arxiv.org: CoRR abs/1203.4881
[8] Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programming. lulu.com (2008)

ACO Beats EA on a Dynamic Pseudo-Boolean Function

Timo Kötzing[1] and Hendrik Molter[2]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
[2] Saarland University, Saarbrücken, Germany

Abstract. In this paper, we contribute to the understanding of the behavior of bio-inspired algorithms when tracking the optimum of a dynamically changing fitness function over time. In particular, we are interested in the difference between a simple evolutionary algorithm (EA) and a simple ant colony optimization (ACO) system on deterministically changing fitness functions, which we call *dynamic fitness patterns*. Of course, the algorithms have no prior knowledge about the patterns.

We construct a bit string optimization problem where we can show that the ACO system is able to follow the optimum while the EA gets lost.

1 Introduction

Bio-inspired algorithms are an important class of randomized search heuristics, valued for their easy applicability also in challenging environments. In particular, environments with uncertainty are settings difficult for problem specific algorithms, while bio-inspired algorithms can more easily deal with the uncertainty.

Jin and Branke [9] discuss different sources of uncertainty and surveys the literature evolutionary algorithms in uncertain environments. Two important sources are a *noisy* fitness function and a *dynamic* fitness function. In the case of a noisy fitness function, the fitness of a search point follows a random distribution, which is the same distribution at each evaluation of the fitness. In the case of a dynamic fitness function, the fitness of a search point at any given time is a deterministic value, but this value changes over time.

Bio-inspired algorithms seem to be very robust against noise and dynamic changes which makes them very popular for these two optimization problem classes. Two prominent types of bio-inspired algorithm are *Evolutionary Algorithms* (EA) and *Ant Colony Optimization* (ACO) systems.

In this paper we are concerned with *dynamic* fitness functions; the goal for an algorithm will be to track the optimum of a dynamically changing fitness function over time. We consider the setting of *pseudo-Boolean fitness functions*, where the search space is given as all bit strings of a fixed length n and the fitness of any search point is a real value.

Some results about *evolutionary* algorithms tracking optima in pseudo-Boolean optimization are given in [1,5,6,15,8]. Some of these papers analyze

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 113–122, 2012.

settings where the optimum changes *randomly*; we are more interested in how bio-inspired algorithms can *track* an optimum that changes *deterministically* (but, of course, the algorithm has no prior knowledge of what changes will happen). We believe that an analysis in such settings on certain dynamic patterns gives a better view on the strengths of bio-inspired algorithms than its performance on an optimum which moves away in a random direction – those changes are hard to control, due to their inherent randomness and the many choices for the optimum to evade in our search space.

In this paper, we try to gain new insights in the differences between the behavior of EA and ACO on these optimization problems; more specifically, we analyze the *(1+1)-EA* and a version of the *Max-Min-ant system* (*MMAS*, introduced in [16]) given in Section 2. We compare the performance of these algorithms on a dynamic optimization problem where we can show that *MMAS* is able to follow the optimum while the *(1+1)-EA* gets lost (see Section 3.1).

The theoretical analysis of these algorithms is very challenging; therefore we use simple dynamic optimization problems, derived from the *OneMax* problem; this approach was also taken in some of the previous literature for the *(1+1)-EA*. There are many analyses of *MMAS* on *static* variants of OneMax (see, for example, [13,7,4,11,12,10,17]), but so far there is no theoretical work on the performance of ACO algorithms on dynamic problems.

We introduce the notion of a *dynamic fitness pattern*. Instead of considering random movements of the optimum, and analyzing an algorithms performance at tracking this random behavior, we are interested in how the *(1+1)-EA* and *MMAS* behave in settings with more structure.

We construct a "maze", in which the *(1+1)-EA* cannot track the optimum and will get lost with high probability. On the other hand, *MMAS can* track the optimum, also with high probability. Key to the success of *MMAS* is the use of intermediate pheromone values, which serve as a medium-term memory: pheromones build up according to solutions that have been good *recently*, extracting *trends* in the fitness pattern. The *(1+1)-EA* misses these trends and, instead, oscillates between different good solutions.

We use *drift analysis* as our key tool to analyze the behavior of the algorithms; in particular, we use many drift theorems from the literature, including a drift theorem with tail bounds from [2], which allows us to derive high probability results.

In general, high probability results are sparse in the analysis of bio-inspired algorithms; a notable recent exception is [17], where many tail bounds are given.

We proceed as follows. In Section 2 we introduce all necessary mathematical definitions. In Section 3 we introduce and analyze a maze where *MMAS* manages to follow the optimum and *(1+1)-EA* loses track of it. We conclude in Section 4.

2 Mathematical Preliminaries

In this section we introduce all formal definitions that we will use in the analysis.

Our general problem setting is *pseudo-Boolean function* optimization. The solution space is the set of all bit strings of length n (for fixed n). We let $h(x, y)$

denote the *Hamming distance* between two bit strings x and y (the number of bits where x and y disagree).

One of the simplest functions for static optimization problems is the OneMax function. This function has proven to be very useful for theoretical analysis of evolutionary algorithms. For any bit string x of length n we let

$$\text{OneMax}(x) = n - h(x, 1^n).$$

The OneMax function is maximized by the all-ones bit string and, importantly, the fitness of a bit string x is better the closer x is to the all-ones string, i.e., the more 1s are in x.

Next we formally introduce the *(1+1)-EA* and *MMAS*.

The *(1+1)-EA* is depicted in Algorithm 1. It keeps a current-best solution x and, in each iteration, mutates it by flipping each bit independently with some mutation probability. This mutation probability is typically $1/n$, where n is the number of bits in the bit string.

If the mutant has a better current fitness than the current fitness of x (x is reevaluated), then x is replaced with its mutant, otherwise x is kept and the mutant discarded.

Algorithm 1. (1+1)-EA

```
1 initialize x;
2 repeat
3 |   x' ← mutate(x);
4 |   if fitness(x') ≥ fitness(x) then
5 |   |   x ← x';
6 until forever;
```

The second algorithm we analyze is an *Ant Colony Optimization* (ACO) system. Here ants drop pheromones on the bits and these serve as a probability distribution to construct new solutions. More specifically, we look at a Max-Min-ant system (*MMAS*, [16]). *MMAS* uses maximum and minimum values for the pheromone values (typically $1 - 1/n$ as maximal and $1/n$ as minimal pheromone values).

MMAS also maintains a current-best solution, initialized to a random bit string x. The pheromone value τ_i, for each bit i, is initialized to $1/2$.

In each iteration, we construct a new solution x' by drawing a new bit string where each bit x'_i is 1 with probability τ_i. Then we compare the fitness of this string to the reevaluated fitness of our current-best solution and replace the current-best solution if the new solution is better. Then we update the pheromone values using the current-best solution.

Note that, as long as no better solution is constructed, the pheromones are updated with always the same current-best solution over and over again until the pheromones hit the respective limits.

Algorithm 2. MMAS

```
initialize x, τ;
repeat
    x' ← constructSolution(τ);
    if fitness(x') ≥ fitness(x) then
        x ← x';
    τ ← update(τ, x);
until forever;
```

The update step works as follows. For each bit x_i, we update the respective pheromone value τ_i to a value τ_i' as follows.

$$\tau_i' = \begin{cases} \min\left\{(1-\rho)\tau_i + \rho, 1 - \frac{1}{n}\right\}, & \text{if } x_i = 1; \\ \max\left\{(1-\rho)\tau_i, \frac{1}{n}\right\}, & \text{if } x_i = 0. \end{cases}$$

When a pheromone value hits a threshold, we call it *saturated*.

3 The Maze

In this section we consider bit strings of any given length n; the mutation probability of the *(1+1)-EA* is set to $1/n$, and the pheromone bounds of *MMAS* are $1/n$ and $1 - 1/n$, respectively.

3.1 Definition of the Maze

We give the following definition of a dynamic fitness pattern where *MMAS* can track the optimum while *(1+1)-EA* gets lost. The idea behind the pattern is, that we start with OneMax, but then let each bit oscillate one after another with a 001-oscillation and then set it to zero. We set the fitness of the optimal string to $n+2$, the fitness of the string, where only the oscillating bit is flipped, to $n+1$, and for the rest we use OneMax. Intuitively, when in an oscillation phase there are strictly more 0s than 1s, *MMAS* will converge (in pheromone) to 0, while the *(1+1)-EA* will oscillate with its current best solution between the different optima. Note that any oscillation pattern that contains more 1s than 0 will not be trackable by *MMAS* (and neither by the *(1+1)-EA*).

We will see in Section 3.2 that *MMAS* is able to track down all the zeros, because the pheromone values of the oscillating bit drop down to the lower border; however, as we will see in Section 3.3, the *(1+1)-EA* loses the zero with probability at least $1/4$ each time we move the oscillating bit one step. Then *(1+1)-EA* falls back to OneMax and is not able to find the optimum again, since it the fitness function leads to the all-ones string.

Formally, we define the dynamic fitness pattern as follows. Let $\mathrm{opt}_{001}(t)$ denote a function that equals 1 at all times t which have a remainder of 2 when divided by 3, and 0 otherwise.

Definition 1 (Maze). Let $\diamond$ denote bit string concatenation, $k > 0$ and let

$$f(i,t) = 0^i \diamond \mathrm{opt}_{001}(t) \diamond 1^{n-i-1},$$

$$f'(i,t) = 0^i \diamond (1 - \mathrm{opt}_{001}(t)) \diamond 1^{n-i-1}$$

define two bit strings of length n, where i determines the position of the oscillating bit and t the time point of the oscillation. These two bit strings will be the optimal and second best solution. Let $t_0 = kn^3 \log(n)$. For any bit string of length n and any t, we define

$$\mathrm{Maze}_k(x,t) = \begin{cases} n+2, & \text{if } x = f(\lfloor \frac{t-t_0}{t_0} \rfloor, t) \\ & \quad \text{and } t > t_0; \\ n+1, & \text{if } x = f'(\lfloor \frac{t-t_0}{t_0} \rfloor, t) \\ & \quad \text{and } t > t_0; \\ \mathrm{OneMax}(x), & \text{otherwise.} \end{cases}$$

We will choose a suitable constant k later. Note that we start the oscillation of the first bit after an initial phase of t_0 iterations, after which each oscillation takes t_0 iterations. The initial phase ensures that both *MMAS* and the *(1+1)-EA* will have found the all-ones bit string as their current-best solution before the oscillations start; the oscillation is, as we will see, long enough for *MMAS* to track the optimum.

3.2 MMAS on Maze

In this section we will show that *MMAS* can track the optimum of the maze with high probability. We start by showing that, during the oscillation phase of a bit between two 1s and 0, *MMAS* will drift towards the 1-bit

Lemma 2. Suppose $\rho \in \Theta(1/n)$. For all $c > 0$, during a single bit 110-oscillation, where in every iteration we construct a new solution with some probability p converging to $1/e$ and update pheromones with the current-best solution otherwise, the pheromone value is saturated as $1 - 1/n$ in $\mathcal{O}(n^3 \log(n))$ iterations with probability $1 - \mathcal{O}(n^{-c})$.

Proof. First we calculate the expected values for the pheromone value and current-best solution after one oscillation. We use the potential function $g(\tau, x) = \tau + q\rho x$, where q is a constant. We set this constant to $q = \frac{7}{2}$.

We begin with the $x = 1$ case, where we get

$$E[\tau'] = \tau\rho(p - 3 \pm \mathcal{O}(\rho)) + \rho(3 - p) + \tau \pm o(\rho),$$

$$E[x'] = \tau(p \pm \mathcal{O}(\rho)) + 1 - p \pm \mathcal{O}(\rho).$$

For the potential difference, we get

$$E[g(\tau', x')] - g(\tau, x) = \tau\rho(p - 3 + qp \pm \mathcal{O}(\rho)) + \rho(3 - p + qp) \pm o(\rho).$$

For $q = \frac{7}{2}$ and p close enough to $1/e$, this yields a drift in $\Omega((1-\tau)\rho)$.

For the $x = 0$ case, we get

$$E[\tau'] = \tau^3\rho(-p^3 \pm \mathcal{O}(\rho)) + \tau^2\rho(p^2 \pm \mathcal{O}(\rho)) + \tau\rho(5p - 2p^2 - 3 \pm \mathcal{O}(\rho)) + \tau,$$

$$E[x'] = \tau^3(-p^3 \pm \mathcal{O}(\rho)) + \tau^2(p^2 - p^3 \pm \mathcal{O}(\rho)) + \tau(2p - 2p^2 \pm \mathcal{O}(\rho)).$$

For the potential difference, we get

$$E[g(\tau', x')] - g(\tau, x) = \Omega(\tau\rho(5p + q2p - 2p^2 - q2p^2 - 3)).$$

For $q = \frac{7}{2}$ and p close enough to $1/e$, this yields a drift in $\Omega(\tau\rho)$.

Thus we have an overall drift of $\Omega(\min\{\tau\rho, (1-\tau)\rho\})$. In particular, we have a uniform additive drift of $\Omega(\rho/n)$. Using the drift theorem with tail bounds from [2, Theorem 1], we obtain the result. Note that this wastes a lot, as the drift theorem with tail bounds is intended for use with multiplicative drift, while we use it on additive drift; also, the statement of the theorem requires a (1+1)-EA, but the proof, published in [3, Theorem 5], shows that this requirement is unnecessary. □

Note that, for the maze, will use Lemma 2 symmetrically for drifting *down* with the pheromone. We will choose k large enough so that during the oscillation of a bit, the pheromone value of the oscillating bit will at least once be $1/n$. Additionally, we have to make sure that the pheromone value stays low.

Lemma 3. Suppose $\rho \in \Theta(1/n)$. Let $k \geq 1$ and let i be the position of the oscillating bit of Maze_k. If the pheromone value τ_i is saturated at the lower threshold, for all $c > 0$, it will stay below $(\log(n))^2/n$ for at least $kn^3 \log(n)$ iterations with probability $1 - \mathcal{O}(n^{-c})$.

Proof. This can be seen with a simple rescaling of the search space and an application of the drift theorem concerned with negative drift from Oliveto and Witt [14].

Note that the largest possible jump away from the lower threshold is 3ρ, since in every update the pheromone value is changes by at most ρ and we update 3 times per oscillation. After rescaling of the search space by $1/\rho$, we see that probability to make jumps larger than 3 is 0, so that the theorem from [14] easily applies. □

Another important ingredient for analyzing *MMAS* on the maze, we have to show that *MMAS* does not get lost when the oscillation switches from bit i to bit $i+1$. This can only happen, if in the last iteration of oscillating i, the ith bit of the current-best solution is 1. This happens with probability at most $1/n$, so we expect it to happen once in the run of the maze.

We show that, in this case, *MMAS* is able to regain track of the optimum again with high probability. W.l.o.g. we show this behavior for the case that all pheromones are saturated at the upper border, i.e. at $1 - 1/n$.

Lemma 4. Suppose $\rho \in \Theta(1/n)$ and the fitness function is $f(x) = n - \text{OneMax}(x)$ (this implies that the all-zeros string is the optimum of f). Furthermore, assume the current-best solution is $1^{n-1}0$ and the first $(n-1)$ pheromones are saturated at $1-1/n$, while the last pheromone value is at $1-O((\log(n))^2/n)$. Then we have, for all $c > 0$, *MMAS* samples 1^n within $\mathcal{O}(\log(n))$ iterations with probability $1 - \mathcal{O}(n^{-c})$.

Proof. Let $c > 0$, and let $t = 3c\log(n)$. We show that *MMAS* samples 1^n within t iterations with probability $1 - \mathcal{O}(n^{-c})$. Within these t iterations, *MMAS* will change more and more bits in its current-best solution to 0. For this to happen, *MMAS* needs to, in some iteration, sample one of the bits with pheromone $1 - 1/n$ as 0. For all $i < t$, let X_i be the number of bits that *MMAS* samples as 0 in iteration i, and that where not sampled as 0 in an earlier iteration. Clearly, for all $i < t$, $E(X_i) \leq 1$. Let $X = \sum_{i=1}^{t} X_i$ be the total number of different bits sampled as 0 within the first t iterations.

Using a Chernoff bound we see that

$$P(X > 3t) \leq n^{-c}.$$

Let $k = 3t = 9c\log(n)$. The following probabilities are conditional on $X \leq 3t = k$.

For the case that we lower the pheromone value of a bit, we lower it by at most ρ. Hence, within t iterations, at most k new bits have a pheromone lower than $1 - 1/n$, by at most $t\rho = o(1)$; the last bit, after t iterations, has pheromone $1 - O((\log(n))^2/n + t\rho)$. Thus, those $k + 1$ bits have a pheromone of $1 - \text{polylog}(n)/n$.[1] All other bits still have maximal pheromone. Hence, the probability to sample 1^n in *any particular* of the t iteration is at least

$$(1 - 1/n)^{n-k-1} \cdot (1 - \text{polylog}(n)/n)^{k+1}.$$

Using Bernoulli's inequality, we can lower bound this probability with

$$\frac{1}{e} \cdot (1 - (2k)\text{polylog}(n)/n) = \frac{1}{e} \cdot (1 - \text{polylog}(n)/n).$$

Now we bound the probability that we do *not* sample 1^n in *any* of the t iterations from above with

$$(1 + \frac{1}{e}(-1 + \text{polylog}(n)/n))^t.$$

We can upper bound this probability by

$$\exp\left(\frac{t}{e}(-1 + \text{polylog}(n)/n)\right) \leq \mathcal{O}(n^{-c}).$$

We now have two chances for *MMAS* to fail to sample 1^n in the first t iterations: either by lowering pheromone on more than k bits, or by failing to sample 1^n

[1] With polylog(n) we denote the set of all functions bounded above by $c\log(n)^d$, for some $c, d > 0$.

conditional on not having decreased pheromones on more than k bits. Both failure probabilities are below n^{-c}; thus, *MMAS* samples 1^n in t iterations with probability $1 - \mathcal{O}(n^{-c})$. □

The following theorem is taken from [17, Corollary 5] and gives high probability bounds for *MMAS* optimizing OneMax.

Theorem 5 ([17]). For all $c > 0$, *MMAS* optimizes OneMax in time $\mathcal{O}(n \log(n)/\rho)$ with probability $1 - \mathcal{O}(n^{-c})$.

We are now ready to give the central theorem of this paper.

Theorem 6. Suppose $\rho \in \Theta(1/n)$. For all $c > 0$ there is a k such that, for n large enough, *MMAS* can follow the optimum[2] of the dynamic fitness pattern Maze_k with probability $1 - \mathcal{O}(n^{-c})$.

Proof. Let $c > 0$. Let k' be the largest implicit constant of the runtime bounds of Lemma 2 and 4, as well as Theorem 5, for obtaining a failure probability of $\mathcal{O}(n^{-c-1})$. Let $k = 3k'$.

Since we run OneMax for the first $kn^3 \log(n)$ iterations, we know by Theorem 5 that *MMAS* finds the optimum and all pheromone values are saturated before the oscillations starts with probability $1 - \mathcal{O}(n^{-c-1})$. During the oscillation of any bit, by (the symmetric version of) Lemma 2, we lower the pheromone of the oscillating bit to $1/n$ with probability $1-\mathcal{O}(n^{-c-1})$ after one third the oscillation of the bit.

Using Lemma 3, the pheromone of an oscillating bit during its oscillation will never be further away than $\log(n)^2/n$ with probability $1-\mathcal{O}(n^{-c-1})$. Thus, when the next bit starts oscillating, by Lemma 4, *MMAS* needs at most one third the oscillation to sample again a solution that has fitness $n+1$ or $n+2$ with probability $1 - \mathcal{O}(n^{-c-1})$. *MMAS* can now recover all pheromones to the proper borders within another third of the oscillation and after this, the process repeats.

MMAS loses the optimum or does not saturate the pheromone value of the oscillating bit by the end of the oscillation with probability n^{-c-1}.

By induction and the union bound for the failure probabilities, *MMAS* follows the optimum of Maze_k with probability $1 - \mathcal{O}(n^{-c})$. □

3.3 (1+1)-EA on Maze

Now we take a look at the *(1+1)-EA*. Because of space limitations, we omit the proofs of this section.

Consider the oscillation of bit i; we will show that the expected value of bit i of the current-best solution of the *(1+1)-EA* is at least $\frac{1}{4}$.

[2] The algorithm is said to "follow the optimum" if, at the end of all phases, the distance of the current-best solution to the current optimum is constant.

Lemma 7. Let $k \geq 1$; consider the *(1+1)-EA* optimizing Maze_k during the oscillation of bit i, and suppose the current-best solution of the *(1+1)-EA* has a 0 on all positions before and at the oscillating bit, and a 1 on all others. For all t, let X^t be the random variable denoting the value of bit i of the current-best solution t iterations later. Then, for all $t \geq n \log(n)$, $E[X^t] \geq 1/4$.

Now we show that *(1+1)-EA* loses track of the optimum with high probability.

Theorem 8. For all $c > 0$ and $k \geq 1$, the *(1+1)-EA* loses track of the optimum of Maze_k with probability $1 - \mathcal{O}(n^{-c})$.

4 Conclusions and Future Work

We have given an example of a dynamic fitness pattern where *MMAS* is able to track the optimum with high probability, while the *(1+1)-EA* loses the optimum with high probability. This shows that there are instances where *MMAS* performs strictly better than the *(1+1)-EA*.

The intuition behind this difference is as follows. Consider a singe bit position, where the optimizing algorithm is supposed to find out whether a 0 or a 1 is better at that position. Suppose none of the other positions influence the fitness of the bit string. For an oscillating optimum at that bit position, the *(1+1)-EA*, in each iteration, will give a definite statement about what bit it thinks to be better (whatever bit the best-so-far bit string has at that position), while the *MMAS* uses intermediate pheromone values as a medium-term memory. This enables *MMAS* to "average" over the phases of oscillation, smoothing out the dynamic fitnesses of the search points, while the *(1+1)-EA* reacts strongly, too strongly, in each iteration.

As one line of future work it would be interesting to see how evolutionary algorithms with a large population size handle similar dynamic problems.

More importantly, though, future research goals should include finding results for more general classes of dynamic fitness patterns. In particular, dynamic combinatorial optimization might offer interesting settings for the analysis of the performance of bio-inspired algorithms.

References

1. Chen, T., Chen, Y., Tang, K., Chen, G., Yao, X.: The impact of mutation rate on the computation time of evolutionary dynamic optimization (2011), http://arxiv.org/abs/1106.0566
2. Doerr, B., Goldberg, L.A.: Drift Analysis with Tail Bounds. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part I. LNCS, vol. 6238, pp. 174–183. Springer, Heidelberg (2010)
3. Doerr, B., Goldberg, L.A.: Adaptive drift analysis. CoRR, abs/1108.0295 (2011)
4. Doerr, B., Neumann, F., Sudholt, D., Witt, C.: On the runtime analysis of the 1-ANT ACO algorithm. In: Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 33–40. ACM (2007)

5. Droste, S.: Analysis of the (1+1) EA for a dynamically changing OneMax-variant. In: IEEE Congress on Evolutionary Computation (CEC 2002), pp. 55–60. IEEE Press (2002)
6. Droste, S.: Analysis of the (1+1) EA for a Dynamically Bitwise Changing OneMax. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 909–921. Springer, Heidelberg (2003)
7. Gutjahr, W.J., Sebastiani, G.: Runtime analysis of ant colony optimization with best-so-far reinforcement. Methodology and Computing in Applied Probability 10, 409–433 (2008)
8. Jansen, T., Schellbach, U.: Theoretical analysis of a mutation-based evolutionary algorithm for a tracking problem in the lattice. In: Genetic and Evolutionary Computation Conference (GECCO 2005), pp. 841–848 (2005)
9. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments—a survey. IEEE Transactions on Evolutionary Computation 9, 303–317 (2005)
10. Kötzing, T., Neumann, F., Sudholt, D., Wagner, M.: Simple max-min ant systems and the optimization of linear pseudo-boolean functions. In: Foundations of Genetic Algorithms (FOGA 2011), pp. 209–218 (2011)
11. Neumann, F., Sudholt, D., Witt, C.: Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. Swarm Intelligence 3, 35–68 (2009)
12. Neumann, F., Sudholt, D., Witt, C.: A few ants are enough: ACO with iteration-best update. In: Genetic and Evolutionary Computation Conference (GECCO 2010), pp. 63–70. ACM (2010)
13. Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. Algorithmica 54, 243–255 (2009)
14. Oliveto, P.S., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. Algorithmica 59, 369–386 (2011)
15. Rohlfshagen, P., Lehre, P.K., Yao, X.: Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In: Genetic and Evolutionary Computation Conference (GECCO 2009), pp. 1713–1720 (2009)
16. Stützle, T., Hoos, H.H.: MAX-MIN ant system. Journal of Future Generations Computer Systems 16, 889–914 (2000)
17. Zhou, D., Luo, D., Lu, R., Han, Z.: The use of tail inequalities on the probable computational time of randomized search heuristics. Theoretical Computer Science (to appear, 2012)

Runtime Analysis of Simple Interactive Evolutionary Biobjective Optimization Algorithms

Dimo Brockhoff[1], Manuel López-Ibáñez[2], Boris Naujoks[3], and Günter Rudolph[4]

[1] INRIA Lille - Nord Europe, DOLPHIN Team, 59650 Villeneuve d'Ascq, France
dimo.brockhoff@inria.fr
[2] IRIDIA, Université Libre de Bruxelles (ULB), Av. F. Roosevelt 50,
CP 194/6, 1050 Brussels, Belgium
manuel.lopez-ibanez@ulb.ac.be
[3] Institute for Informatics, Cologne University of Applied Sciences, Steinmüllerallee 1,
D-51643 Gummersbach, Germany
boris.naujoks@fh-koeln.de
[4] Fakultät für Informatik, Technische Universität Dortmund, 44221 Dortmund, Germany
Guenter.Rudolph@tu-dortmund.de

Abstract. Development and deployment of interactive evolutionary multiobjective optimization algorithms (EMOAs) have recently gained broad interest. In this study, first steps towards a theory of interactive EMOAs are made by deriving bounds on the expected number of function evaluations and queries to a decision maker. We analyze randomized local search and the (1+1)-EA on the biobjective problems LOTZ and COCZ under the scenario that the decision maker interacts with these algorithms by providing a subjective preference whenever solutions are incomparable. It is assumed that this decision is based on the decision maker's internal utility function. We show that the performance of the interactive EMOAs may dramatically worsen if the utility function is non-linear instead of linear.

1 Introduction

Interactive algorithms for multi-criteria decision making are typically based on real-world case studies and designed to work well in practice. However, to the best of our knowledge, approaches combining interactive decision making with EMOAs have not been analyzed mathematically in terms of their runtimes and the expected number of questions asked (queries) to the decision maker (DM). For example, it is sometimes claimed that only a small number of queries are performed in practice [5, p. 135], but we are not aware of any rigorous analysis of how many queries are actually necessary to obtain the most preferred solution or an approximation thereof. Specifically for interactive EMOAs, which are *randomized* search heuristics where no guarantee can be given on when good search points are found, a theoretical understanding of their *expected* optimization time will be extremely helpful when comparing approaches and predicting their performance on future unknown problems.

This paper provides a first attempt to theoretically analyze the runtime and the number of queries to the DM of interactive EMOAs. To this end, we propose two simple interactive randomized search heuristics and analyze them on the two standard binary

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 123–132, 2012.

test problems Leading Ones Trailing Zeros (LOTZ, [4]) and Counting Ones Counting Zeros (COCZ, [3]). The algorithms are interactive versions of the well-known randomized local search (RLS) and the (1+1)-EA and neither do not know the DM's preferences nor do they assume anything about the DM. Whenever the Pareto-dominance relation does not indicate a search direction between the current search point and the mutated offspring because the two solutions are incomparable, the algorithms query the DM about her opinion and accept the solution that was favored by the DM. For several *simulated* DMs, where a specific underlying utility function of the DM is assumed, tight bounds on the expected runtimes and the expected number of queries to the DM for the two interactive algorithms can be proven—using established proof techniques that have been developed to bound the expected runtime of other randomized search heuristics.

Our proofs should be seen as a starting point for analyzing interactive approaches. More involved algorithms and more complicated models of the DM will have to be analyzed in the future. It is also a first step towards understanding the consequences of adding increasingly more complicated utility functions and how algorithms should be designed to deal with them. One major conclusion from our analyses is that the performance of simple interactive EMOAs dramatically changes when the DM is acting according to linear or non-linear utility functions.

Section 2 provides a mathematical prelude and some basic assumptions before the two proposed interactive algorithms are presented in Sec. 3. The biobjective test problems are described in Sec. 4. The runtime analysis starts in Sec. 5 for the LOTZ problem, which is followed in Sec. 6 by the runtime analysis of the iRLS on COCZ. Section 7 summarizes our findings and provides an outlook to future research.

2 Mathematical Prelude and Basic Assumptions

Let $\mathbb{B}^n = \{0,1\}^n$ be the finite discrete search space of binary strings of length $n \in \mathbb{N}$. We consider the simultaneous maximization of two objective functions $f\colon \mathbb{B}^n \to \mathbb{N}_0^2$ with $f(x) = (f_1(x), f_2(x))$ and define the weak dominance relation $\succeq$ on the search space $\mathbb{B}^n$ via $x \succeq y$ iff $f_1(x) \geq f_1(y)$ and $f_2(x) \geq f_2(y)$ for $x, y \in \mathbb{B}^n$ and the dominance relation $\succ$ via $x \succ y$ iff $x \succeq y$ and $f(x) \neq f(y)$. The decision maker is interpreted as a black box or oracle that is queried to decide which of two given *objective vectors* ("$f(x)$ or $f(y)$?") is better.[1]

In its simplest form, we assume that the DM has an underlying *value* or *utility function* $u(x)$ [2, p. 68 & 80f] according to which she is making her decision

$$\mathrm{DM}(f(x), f(y)) = f(x) \cdot \mathbb{I}_{\{u(f(x))>u(f(y))\}} + f(y) \cdot \mathbb{I}_{\{u(f(x))\leq u(f(y))\}}$$

where $\mathbb{I}_A$ is the indicator function, giving 1 iff its argument A is true, and 0 otherwise.

The utility function u is thereby a function that maps an objective vector $f(x)$ to a real value, and an objective vector $f(x)$ is said to be preferred by the DM over a

[1] Limiting the queries to objective vectors simplifies the proofs presented in this paper. In principle, the DM can also be queried about her preference among solutions x and y directly, but the theoretical results will be different: it is expected that the number of queries to the DM will increase since, for discrete problems, the total number of solution pairs is typically larger than the number of objective vector pairs.

Algorithm 1. (1+1)-iEA and iRLS

```
1: init: choose x^0 uniformly at random; t = 0
2: repeat
3:    y ← mutate(x^t) { iRLS: 1 bit flip; (1+1)-iEA: independent bit flip }
4:    if f(y) ⪰ f(x^t) then
5:       x^{t+1} ← y
6:    else if f(x^t) ≻ f(y) then
7:       x^{t+1} ← x^t
8:    else
9:       u* ← DM(f(x^t), f(y)) {only asks the DM once about x^t and y}
10:      if u* = f(x^t) then
11:         x^{t+1} ← x^t
12:      else
13:         x^{t+1} ← y
14:   t ← t + 1
15: until DM terminates
```

vector $f(y)$ iff $u(f(x)) > u(f(y))$. In case of a *weighted sum*, the utility function is $u(f(x)) = w_1 \cdot f_1(x) + (1 - w_1) \cdot f_2(x)$ and for a *weighted Chebyshev utility function*, u is defined as $u(f(x)) = \max_{i \in \{1,2\}} \{w_i \cdot |z_i^* - f_i(x)|\}$ for every $x \in \mathbb{B}^n$, with $w = (w_1, 1-w_1) \in \mathbb{R}^2$ being the corresponding weight vector and $z^* = (z_1^*, z_2^*) \in \mathbb{R}^2$ a pre-defined utopian vector with $z_i^* \geq \max_{x \in \mathbb{B}^n} f_i(x)$ for $i \in \{1, 2\}$.

Both the weighted sum and the weighted Chebyshev utility function ensure that an objective vector $f(x)$ has a higher utility than another vector $f(y)$ if x is dominating y. Utility functions with this property are termed Pareto compliant.

3 The Interactive (1+1)-EA and (1+1)-RLS Algorithms

The simplest interactive evolutionary algorithm imaginable is based on the (1+1)-EA [1] and it is shown in Algorithm 1. After drawing a parent uniformly at random from $\mathbb{B}^n$, the algorithm iterates the following steps until the DM decides to terminate it. First, an offspring is generated by a mutation of the parent. If only a single bit chosen uniformly at random is flipped then we shall call the algorithm iRLS, whereas it is termed (1+1)-iEA if each bit is flipped independently with probability $1/n$. If the offspring dominates or equals the parent, then it is accepted and serves as parent of the next iteration. If the offspring is dominated by the parent, then it is rejected and the parent passes to the next iteration. If offspring and parent are incomparable then the DM decides which of them is accepted. We assume that the DM can be modeled by a scalar utility function so that the individual with larger utility is accepted. This utility function is assumed to be Pareto compliant such that the algorithms decide according to the DM's preferences in case of dominated or dominating offspring.

We further assume that the algorithm stores all objective vector pairs presented to the DM so far, such that the DM is never asked to rank the same pair of objective vectors more than once in order to keep the queries to the DM as low as possible.

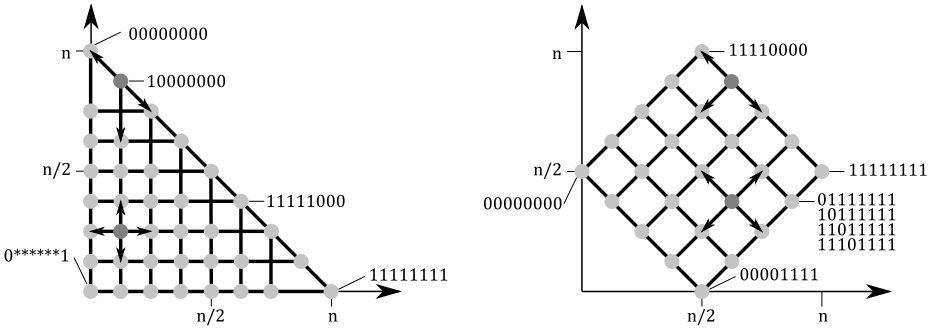


Fig. 1. Illustration of the objective space of the bi-objective LOTZ (left) and COCZ (right) problems for bitstrings of length $n = 8$. The neighborhood between objective vectors in terms of the 1-bit mutation of the iRLS is indicated with black lines and the neighbors of two example solutions for each problem are shown with arrows.

4 The LOTZ and COCZ Problems

In the following section, we will present the analyses of the runtime behavior of the (1+1)-iEA and iRLS on two simple well-known test problems:

Definition 1. *The bi-objective maximization problem with objective function* $f(x) = (LO(x), TZ(x))$ *where*

$$LO(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j \qquad \text{and} \qquad TZ(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} (1 - x_j)$$

for $x \in \mathbb{B}^n$ *is termed the* Leading Ones Trailing Zeros (LOTZ) *problem.*

Definition 2. *The bi-objective maximization problem with objective function* $f(x) = (CO(x), CZ(x))$ *where*

$$CO(x) = \sum_{i=1}^{n} x_i \qquad \text{and} \qquad CZ(x) = \sum_{i=1}^{\lfloor n/2 \rfloor} x_i + \sum_{i=\lfloor n/2 \rfloor + 1}^{n} (1 - x_i)$$

for $x \in \mathbb{B}^n$ *is termed the* Counting Ones Counting Zeros (COCZ) *problem.*

Both problems have been the basis of the first runtime analyses of simple MOEAs such as the SEMO and global SEMO algorithms. These problems have a linear Pareto front with $n + 1$ (LOTZ) and $\lceil n/2 \rceil + 1$ (COCZ) different Pareto-optimal objective vectors. The total number of different objective vectors is $\Theta(n^2)$ in both cases. Figure 1 illustrates their objective space and the neighborhood of objective vectors with respect to 1-bit mutations.

5 Runtime Analysis of iRLS and (1+1)-iEA on LOTZ

Let us now investigate the runtime and the number of DM queries for the iRLS and the (1+1)-iEA until the most preferred solution of the LOTZ problem is found.

Theorem 1. *When optimizing the LOTZ function, the iRLS needs in expectation $\Theta(n^2)$ function evaluations to find the solution that is most preferred by the DM if the utility function of the DM is either a weighted sum or the weighted Chebyshev. To this end, an expected number of $O(n)$ queries to the DM are performed.*

Proof. With the same arguments as for the SEMO algorithm [3], the iRLS will start the optimization with high probability in the lower left corner of the objective space and needs $\Theta(n)$ improvements to reach the Pareto front. Before reaching the Pareto front for the first time, any solution produced by a 1-bit flip either dominates or is dominated by the current solution, and, hence, the DM is never asked. An improvement towards the Pareto front is possible by flipping the first 0-bit or the last 1-bit in the current solution, which happens with a probability of $2/n$ in each iteration and corresponds to a waiting time of $O(n)$. Hence, the Pareto front is reached in $O(n^2)$ iterations. Since all bits are chosen uniformly at random in the beginning and no incentive is given to bias this uniform probability during the search, also the lower bound of $\Omega(n^2)$ steps to reach the Pareto front holds because in expectation there is less than one "free-rider" bit per improvement, cf. the argumentation in [1].

Once the Pareto front is reached for the first time, all new solutions are either dominated by the current one, and, hence, discarded immediately, or incomparable. In the latter case, the DM is asked which solution is preferred. If the DM's underlying utility function is a weighted sum, then either all Pareto-optimal solutions are equally preferred (for equal weights) or one of the extremes (1^n or 0^n) is the most preferred, due to the fact that all Pareto-optimal objective vectors of LOTZ lie on a line. In the case of equal weights, the expected runtime is, therefore, $\Theta(1)$. Otherwise, the time to reach the most preferred extreme depends on its distance from the first Pareto-optimal solution found and the number of iterations to move from one to the other. The first Pareto-optimal solution found has, with high probability, a number of leading ones (or trailing zeros) within $[1/4n, 3/4n]$ due to Chernoff bounds and the fact that the expected number of leading ones in the first Pareto-optimal point found is $n/2$ for symmetry reasons. Hence, the algorithm needs $\Theta(n)$ successful steps to move to the most preferred extreme. Moreover, the probability of each successful step is $1/n$ and its waiting time $\Theta(n)$. Hence, the most preferred extreme is reached in $\Theta(n^2)$ iterations.

Finally, the expected number of queries to the DM is $\Theta(n)$ in the case of unequal weights (and zero in the case of equal weights), because there are only $\Theta(n)$ nondominated objective vectors in the Pareto front and from each Pareto-optimal search point, at most two others can be generated by 1-bit mutations.

If the underlying utility function is a weighted Chebyshev, any Pareto-optimal solution can be the most preferred one, and, hence, depending on the location of this solution, the number of iterations to reach it (resp. the number of DM queries) may be closer to $\Theta(n^2)$ (resp. $\Theta(n)$) or closer to $\Theta(1)$. In any case, the above upper bounds on the runtime ($O(n^2)$) and number of DM queries ($O(n)$) hold. □

Overall, we proved the expected runtime of the iRLS to be $\Theta(n^2)$: $\Theta(n^2)$ to reach the Pareto front and an additional $O(n^2)$ to find the most preferred Pareto-optimal point. If a weighted sum is assumed as the DM's utility function, the expected number of queries to the DM is either $\Theta(1)$, if all Pareto-optimal points are equally preferred, or $\Theta(n)$, otherwise. If the utility function of the DM turns out to be a weighted Chebyshev function, the expected number of DM queries depends on the weights of the Chebyshev function and can be only bounded by $O(n)$ from above and $\Omega(1)$ from below as both cases are possible (if one of the extremes or the point on the diagonal is preferred).

It turns out that the above proven bounds on the runtime and the number of DM queries do not hold for arbitrary utility functions:

Observation 1. *Already for quadratic utility functions which are Pareto compliant, it can happen that the expected runtime of the iRLS is not finite anymore.*

Proof. Assuming that the quadratic utility function $u(f(x)) = -(n - f_1(x) + 1) \cdot (n - f_2(x) + 2)$ has to be maximized, the most preferred solution is the rightmost Pareto-optimal point, i.e., the objective vector with largest f_1-value. When comparing two incomparable solutions to the left of the objective space's diagonal, the objective vector with smaller f_1- and larger f_2-value will be preferred by the DM. Hence, iRLS will converge towards the leftmost Pareto-optimal point, from which the most preferred search point cannot be reached anymore because an n-bit flip would be necessary to jump there. As the probability of reaching this left extreme of the Pareto-optimal front is non-zero, the expected runtime is not finite anymore. □

With a similar argumentation, it can be shown that the (1+1)-iEA can have an exponential runtime if the above quadratic utility function has to be maximized. Hence, let us consider the (1+1)-iEA with its independent bit flip mutation only for the case of a weighted sum utility function. It turns out that, in this case, the expected runtime is the same as for the iRLS.

Theorem 2. *When optimizing the LOTZ function and if the utility function of the DM is a weighted sum, the (1+1)-iEA needs $\Theta(n^2)$ function evaluations, in expectation, to find the most preferred solution.*

Proof. For a weighted sum as utility function, we assume w.l.o.g. $w_1 \geq 0.5 \geq 1 - w_1$ and consider the following drift function $g(x)$ when the current search point x of the (1+1)-iEA is mapped to an objective vector (i, j):

$$g(x) = n - \lfloor w_1 \cdot i + (1 - w_1) \cdot j \rfloor \ .$$

The intuition behind $g(x)$ is to consider the maximum number of consecutive f_1-improvements needed in the future course of the algorithm to reach the Pareto front, i.e., the maximum distance to the Pareto front in f_1 direction over all points that have a weighted sum utility which is not smaller than the one for the current search point. As $w_1 \geq 0.5$, it is clear that this largest distance is upper bounded by the distance in f_1-direction between the search point with objective vector $(\lfloor w_1 \cdot i + (1 - w_1) \cdot j \rfloor, 0)$ and the Pareto front, which is exactly $g(x)$.

Now, let us consider the course of $g(x)$. We have $g(x) \leq n$ and $g(x)$ never increases due to the selection of the (1+1)-iEA. Furthermore, if $g(x) \leq 1$, then the current search point is Pareto-optimal and only Pareto-optimal points will be accepted until the most preferred solution is found. We now divide the analysis into two phases: the first phase ends when the first Pareto-optimal point is found, while the second phase starts with the first Pareto-optimal point found and ends when the most preferred solution is found. The length of the first phase can be bounded from above by the time until $g(x)$ becomes smaller than 1 under the assumption that x stays non-Pareto-optimal as long as $g(x) > 1$.In this case, the probability to increase f_1 by 1 while f_2 stays constant by mutation is at least $1/n \cdot (1-1/n)^{n-1} \geq 1/en$ with the Euler constant $e \approx 2.71$ and therefore, in expectation, at most en steps of the (1+1)-iEA are necessary for this event, by which $g(x)$ decreases by $w_1 \geq 0.5$. Hence, in expectation, $2en$ of those steps are sufficient to decrease $g(x)$ by 1 and overall at most $2en^2$ many steps are needed in expectation to reach the Pareto front. By Chernoff bounds, the probability that in $4en^2$ steps, the Pareto front is not reached is exponentially small and the runtime bound for the first phase is proven to be $O(n^2)$.

Considering phase two, we distinguish two further cases. Either, the new solution is also Pareto-optimal or we are back in the scenario of phase one, i.e., $g(x) > 1$ and the new solution is non-Pareto-optimal. In the latter case, $g(x)$ is further decreased but we do not spend additional time because we accounted for it already in phase one. In the first case, at most $n-1$ improvements in the first objective function value are necessary to reach the most preferred point where such an improvement happens with a specific 1-bit flip (i.e. again with a probability of at least $1/en$). The expected number of steps needed for at most $n-1$ of those improvements is then smaller than en^2 and, by Chernoff bounds, the probability to need more than $2en^2$ steps is exponentially small and the runtime for the second phase is also $O(n^2)$. □

In the above case of the (1+1)-iEA optimizing the LOTZ problem, the number of DM queries is trivially upper-bounded by $O(n^2)$ since the number of possible objective vector pairs is bounded by $O(n^2)$. However, one can show a stronger result for which we only sketch the proof here due to space limitations.

Theorem 3. *When optimizing the LOTZ function and if the utility function of the DM is a weighted sum, the (1+1)-iEA queries the DM in expectation $O(n)$ times until the most preferred solution is found.*

Sketch of Proof: From the proof of Theorem 2, we know already that the algorithm typically needs $O(n)$ improvements to reach the most preferred solution for which we need to wait $O(n)$ function evaluations each. If we know the probabilities $p_{i,j}$ to reach an incomparable search point from the current objective vector (i,j) with $1 \leq i,j \leq n$ and $i+j \leq n$ and we can upper bound them by a constant p, we can upper bound the expected number of incomparable solutions produced from (i,j) within a phase of cn steps ($c \in \mathbb{N}$ a constant) in which an improvement is likely by the expectation of the binomially distribution with parameters p and cn:

$$\sum_{k=1}^{cn} k \cdot \binom{cn}{k} (p_{i,j})^k (1-p_{i,j})^{cn-k} \leq \sum_{k=1}^{cn} k \cdot \binom{cn}{k} p^k (1-p)^{cn-k} = cnp \ . \quad (1)$$

Note that the Eq. 1 does not take into account the fact that for each objective vector pair the DM is only asked once such that the expected number of incomparable solutions in cn steps of the (1+1)-iEA is actually smaller. When looking at the exact probabilities[2] $p_{i,j}$, it turns out that they can be easily bounded from above by $4/n$ for large enough n—independent of i and j such that Eq. 1 becomes $4c$. That means that in each phase of length cn with n large enough, only a constant number of incomparable solution is generated in expectation, which results, with high probability, in $O(n)$ incomparable solution pairs for which the DM is queried until linearly many improvements have been achieved. □

6 Runtime Analysis of the iRLS on COCZ and Linear Functions

In addition to the LOTZ problem, we now analyze the iRLS on the COCZ problem and point out how the result is related to the optimization of linear functions if the weighted sum is assumed as underlying utility function of the DM.

Theorem 4. *When optimizing the COCZ function, the iRLS needs, in expectation, $\Theta(n \log n)$ function evaluations and $O(n)$ queries to the DM to find the solution that is most preferred by the DM if the utility function of the DM is a weighted sum.*

Proof. Similar to the proof of SEMO's runtime [3], we partition the search space into sets F_i ($0 \leq i \leq \lfloor n/2 \rfloor$) such that all solutions with a number of i 1-bits in the first half of their bitstrings are in set F_i. The most important observation for proving the above theorem is that the 1-bit mutation of the iRLS allows only two scenarios. Either (i) the mutation happens in the first half of the bitstring; in this case, both objectives are perfectly correlated such that the mutated offspring y is dominating the previous solution x_t or it is dominated by it; in any case, the DM is not asked in this situation and the current search point will never fall back to a set F_i with smaller index. Or (ii) the mutation takes place in the second half of the bitstring; then, the mutated offspring y is incomparable to x_t due to the fact that both objectives are anti-correlated; both solutions belong to the same set F_i and the DM is asked to compare them.

With these observations, it is easy to prove upper and lower bounds on the running time of the iRLS on COCZ. With probability $\frac{\lfloor n/2 \rfloor - i}{n}$, the iRLS leaves the set F_i (case (i)), namely if one of the $\lfloor n/2 \rfloor - i$ zeros in the first half of the bitstring of x_t is flipped. This results in a runtime until the first Pareto-optimal point is found, of $\Theta(n \log n)$, see the argumentation for the (1+1)EA on the ONEMAX function, e.g., in [6].

With a similar argumentation, the most preferred Pareto-optimal solution is found after (a possibly additional) $O(n \log n)$ steps as also in the second half of the bitstring, the iRLS has to perform the optimization of ONEMAX (or ZEROMAX, depending on the weight w_1). Overall, the iRLS needs an expected number of $\Theta(n \log n)$ function evaluations until the most preferred search point is found.

[2] For each non-Pareto-optimal solution with objective vector (i, j), for example, one can upper bound the probabilities to reach every incomparable non-Pareto-optimal objective vector $(i + \Delta i, j - \Delta j)$ with $1 \leq \Delta j \leq j$ and $1 \leq \Delta i \leq n - i - j - 2 - \Delta j$ by $\frac{1}{n^2}\left(1 - \frac{1}{n}\right)^{i+j+\Delta i-\Delta j}$ and sum those probabilities up for all possible values of $(\Delta i, \Delta j)$.

Regarding the number of DM queries, we argue that the algorithm performs two independent movements: (i) towards the front, where it is solely the number of ones in the first half of the bitstring that determines how far the current search point is away from the front and for which the DM is never queried, and (ii) the movement towards the extremes of the (local) Pareto front(s) F_i, where it is only important how many ones are present in the second half of the bitstring. Let us denote the objective vector of the current search point by a tuple (i, j) where i indicates the number of 1-bits in the first half and j the number of 1 bits in the second half of the bitstring of the current search point. Assuming without loss of generality, that the weight w_1 of the DM's weighted sum utility function is larger than 0.5, the rightmost Pareto-optimal point and thus the all-one-string is the most preferred solution. All accepted objective vectors, and therefore all visited (i, j) positions will lie on a line connecting the objective vector of the initial search point and this most preferred point whereas the current search point will never decrease in its i and j coordinates due to the 1-bit mutation and the acceptance step of the iRLS. Then, there are only $O(n)$ different objective vectors on this line. On the contrary, there are in expectation also $\Omega(n)$ many objective vectors on that line because, with high probability, the initial search point has about half its bits set to 0 and the other half set to 1 and thus starts with a j-value of about $n/4$. With the additional argument that for each of the $\Theta(n)$ objective vectors accepted throughout the search, maximally two questions can be asked to the DM, the overall amount of expected DM queries is proven to be $O(n)$. □

As we have seen, the iRLS asks the DM in expectation $O(n)$ times on the COCZ problem. This upper bound is due to the fact that the algorithm keeps track about which objective vector pairs have been presented to the DM in order to not ask her twice. If this property of the algorithm is relaxed towards an approach without memory, the number of DM calls increases to $\Theta(n \log n)$ in expectation.

As both objective functions of the COCZ problem are linear, this result can also be obtained from a more general analysis which even holds for the (1+1)-iEA.

Observation 2. *If both objective functions are linear functions and the underlying utility function of the DM is the weighted sum, the overall fitness function of the DM is also linear; in case, we ask the DM all the time, this will be like having to solve a linear function. It is well known that randomized local search and the (1+1)-EA, to which the iRLS and (1+1)-iEA reduce if the DM is asked at every iteration have an expected runtime on linear functions of* $\Theta(n \log n)$ *[7].*

7 Summary and Outlook

The theoretical analysis of interactive multiobjective evolutionary algorithms is a necessary step towards better understanding interactive approaches in order to be able to recommend certain algorithms over others in practical optimization. In this study, we have provided the first of such analysis. The algorithms iRLS and (1+1)-iEA have been proposed which are simple variants of the well-known algorithms RLS and (1+1)-EA for single-objective optimization which ask the decision maker whenever the mutation step produces an incomparable search point. Rigorous analyses of their expected optimization time and the expected number of DM calls until the most preferred solution is

found have been performed on the two well-known bi-objective binary problems LOTZ and COCZ. It turns out that the expected runtime and the number of DM calls highly depend on the assumed model of the DM, i.e., her underlying utility function. The LOTZ problem is one example where the change from a linear to a quadratic preference model changes the runtime of the iRLS from polynomial to infinite.

Though performed on basic test functions and simple algorithms which do not assume anything about the DM's preferences, our analyses open up a new research direction of analyzing more involved interactive optimization algorithms on more realistic multi-objective optimization problems. The proof techniques used in this study are standard and highly related to the proof techniques previously used to analyze population-based evolutionary multiobjective optimization algorithms. Hence, we expect that interactive approaches can be also analyzed on more complicated problems and with more complicated models of the decision maker. Furthermore, we hope that this study will initiate a discussion about the consequences of assuming increasingly more realistic utility functions and on how algorithms should be designed to deal with those.

Acknowledgments. The authors would like to thank the Research Center Schloss Dagstuhl[3] for hosting the "Learning in Multiobjective Optimization" seminar (id 12041) from which this work originates. We would also like to thank Anne Auger for many valuable discussions about the proofs and the anonymous reviewers for spotting some wrong arguments in the earlier manuscript.

References

1. Droste, S., Jansen, T., Wegener, I.: On the Analysis of the (1+1) Evolutionary Algorithm. Theoretical Computer Science 276, 51–81 (2002)
2. Keeney, R.L., Raiffa, H.: Decision with Multiple Objectives: Preferences and Value Tradeoffs. Wiley, New York (1976)
3. Laumanns, M., Thiele, L., Zitzler, E.: Running Time Analysis of Multiobjective Evolutionary Algorithms on Pseudo-Boolean Functions. IEEE Transactions on Evolutionary Computation 8(2), 170–182 (2004)
4. Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., Deb, K.: Running Time Analysis of Multiobjective Evolutionary Algorithms on a Simple Discrete Optimization Problem. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN VII. LNCS, vol. 2439, pp. 44–53. Springer, Heidelberg (2002)
5. Miettinen, K.: Nonlinear Multiobjective Optimization. Kluwer, Boston (1999)
6. Oliveto, P.S., Yao, X.: Runtime Analysis of Evolutionary Algorithms for Discrete Optimization. In: Auger, A., Doerr, B. (eds.) Theory of Randomized Search Heuristics: Foundations and Recent Developments, pp. 21–52. World Scientific Publishing (2011)
7. Witt, C.: Optimizing Linear Functions with Randomized Search Heuristics - The Robustness of Mutation. In: Dürr, C., Wilke, T. (eds.) Symposium on Theoretical Aspects of Computer Science (STACS 2012). Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, pp. 420–431. Schloss Dagstuhl - Leibniz-Center for Informatics (2012)

[3] http://www.dagstuhl.de/en

Parsimony Pressure versus Multi-objective Optimization for Variable Length Representations

Markus Wagner and Frank Neumann

School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia

Abstract. We contribute to the theoretical understanding of variable length evolutionary algorithms. Such algorithms are very flexible but can encounter the bloat problem which means solutions grow during the optimization run without providing additional benefit. We explore two common mechanisms for dealing with this problem from a theoretical point of view and point out the differences of a parsimony and a multi-objective approach in a rigorous way. As an example to point out the differences, we consider different measures of sortedness for the classical sorting problem which has already been studied in the computational complexity analysis of evolutionary algorithms with fixed length representations.

1 Introduction

Evolutionary algorithms that work with a variable length representation often encounter the bloat problem which means that individuals grow without providing additional benefit to the quality of the solutions. Even worse such a growth of the individuals can block the optimization process such that problems that are relatively easy to optimize can not be handled by variable length evolutionary algorithms. Due to this problem, different methods have been introduced to deal with the bloat problem. Our goal is to study the behavior of variable length evolutionary algorithms from a mathematical perspective. We will examine algorithms for distinguished classes of problems and point out the impact of different approaches for dealing with the bloat problem in a rigorous way.

The most prominent example of a variable length evolutionary algorithm is genetic programming [7] which often evolves tree structures for a given problem. Recently, the first computational complexity results on these type of algorithm have been obtained. They follow the line of research that has successfully followed for evolutionary algorithms with fixed length representation (see the books [1, 11] for an overview). Variable length representations increase the search space significantly and in the light of genetic programming it seems to be wishful to better understand the behavior of algorithms using such representations from a theoretical point of view.

The computational complexity analysis of variable length evolutionary algorithms has started just recently. For example, Cathabard et al. [2] investigated

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 133–142, 2012.

non-uniform mutation rates for problems with unknown solution lengths. They used a simple evolutionary algorithm to find a bitstring with an unknown number of leading ones, and although the bitstring had some predetermined maximum length, only an unknown number of initial bits was used by the fitness function. Durrett et al. [3] investigated worst-case and average-case runtimes of a simple tree-based genetic programming algorithm. The tackled problems were separable, with independent and additive fitness structures. Kötzing et al. [6] analysed simple GP algorithms for the MAX problem.

One prominent way of dealing with the bloat problem is the parsimony approach. In the case, that two solutions have equal quality the solution of lower complexity is preferred. Another way of coping with the bloat problem is to use a multi-objective approach which uses in each iteration of a variable length evolutionary algorithm a population which represents the different trade-offs according to the original goal function and the complexity of a solution. The solutions that represent the trade-offs are called Pareto optimal. Note that the parsimony approach is a scalarization approach as it uses these Pareto optimality and a lexicographic ordering. It is known that each global solution is also Pareto optimal, but not all Pareto optimal solutions can necessarily be found through scalarizations (e.g., see [13]). Both approaches of coping with the bloat problem have recently been examined for the problems ORDER and MAJORITY in the context of genetic programming [9, 14].

We further explore the use of parsimony pressure and multi-objective models. In [9] it is shown that both approaches help for ORDER and MAJORITY, but the differences between these two approaches are not examined. In this paper, we point out that switching from the parsimony approach to the multi-objective one can significantly reduce the runtime. In particular, we show that the parsimony approach can have local optima which lead to an infinite runtime whereas the multi-objective approach is able to compute the optimal solution within a polynomial number of steps.

We show these results for a classical problem from the computational complexity analysis of evolutionary algorithms with fixed-length representations, namely the sorting problem (sorting). Scharnow, Tinnefeld, and Wegener [12] considered sorting as an optimization problem and investigated different fitness functions measuring the sortedness of a permutation of elements. Different fitness functions lead to problems of different difficulties. Our goal is to explore how variable-length evolutionary algorithms behave on these problems. We take it as a prominent example to discuss the differences between a parsimony approach and a multi-objective one. In particular, we show that the parsimony approach can end up for a lot of the different sortedness measures in local optima when using a variable length representation whereas the multi-objective approach allows to compute the whole Pareto front in expected polynomial time.

Our paper is organized as follows. In Section 2, we introduce the two models and the different measures of sortedness. We examine the parsimony approach in Section 3 and show that it leads to local optima in the search space. In Section 4, we show that the multi-objective approach is able to compute the whole

Mutate Y by applying k operations. For each operation, randomly choose to either substitute, insert, or delete.

- If substitute, replace a randomly chosen element of Y with a new element $u \in E$ selected uniformly at random.
- If insert, choose an element v in Y uniformly at random and select $u \in E$ uniformly at random. Randomly decide whether u is inserted before or after v in Y.
- If delete, randomly choose an element v of Y and delete it.

Fig. 1. Mutation operator

Pareto front of the underlying optimization problem in expected polynomial time. Finally, we finish with some conclusions.

2 Preliminaries

Our goal is to study the difference between a parsimony and a multi-objective approach for variable length evolutionary algorithms. Solutions contain (possibly multiple) elements from a set E of elements. We will consider mutation-based algorithms which produce new solutions by applying the mutation operator outlined in Figure 1. The mutation operator is parametrized by a parameter k which determines the number of operations applied to the individual Y. For single operations $k = 1$ holds. In the case of multiple operations, k is chosen according to $1 + Pois(1)$ where $Pois(1)$ denotes the Poisson distribution with expectation 1.

We will consider a given problem F and the complexity of a solution C measured by the number of elements in the solution. C should be minimized and we assume that F should be maximized. The notions can be easily adjusted to the minimization of a problem F, which will later on be considered.

In the parsimony approach, we optimize the multi-criteria fitness function MO-F$(X) = (F(X), C(X))$ with respect to the lexicographic order, that is, MO-F$(X) \geq$ MO-F(Y) holds iff

$$F(X) \geq F(Y) \vee (F(X) = F(Y) \wedge C(X) \leq C(Y)) \,. \tag{1}$$

In the multi-objective case, we treat the two criteria F and C as equally important and consider the classical Pareto dominance relations:

1. A solution X *weakly dominates* a solution Y (denoted by $X \succeq Y$) iff $(F(X) \geq F(Y) \wedge C(X) \leq C(Y))$.
2. A solution X *dominates* a solution Y (denoted by $X \succ Y$) iff $((X \succeq Y) \wedge (F(X) > F(Y) \vee C(X) < C(Y))$.
3. Two solution X and Y are called *incomparable* iff neither $X \succeq Y$ nor $Y \succeq X$ holds.

A *Pareto optimal solution* is a solution that is not dominated by any other solution in the search space. All Pareto optimal solutions together form the

Pareto optimal set, and the set of corresponding objective vectors forms the Pareto front. The classical goal in multi-objective optimization is to compute for each objective vector of the Pareto front a Pareto optimal solution. Alternatively, if the Pareto front is too large, the goal then is to find a representative subset of the front, where the definition of 'representative' depends on the choice of the conductor.

2.1 Sortedness Measures

We will analyze our algorithms on different measures of sortedness for the classical sorting problem. It can be stated as follows. Given a totally ordered set $E = \{1, \ldots, n\}$ of n elements, the task is to find a permutation π_{opt} of the elements of E such that

$$\pi_{opt}(1) < \pi_{opt}(2) < \ldots < \pi_{opt}(n)$$

holds, where $<$ is the order on E. Without loss of generality, we assume $\pi_{opt} = id$, i. e. $\pi_{opt}(i) = i$ for all i, throughout this paper.

The set of all permutations π forms a search space that has already been investigated in [12] for the analysis of permutation-based evolutionary algorithms. The authors of that paper investigate sorting as an optimization problem whose goal is to maximize the sortedness of a given permutation. We will consider the following fitness functions measuring the sortedness of a given permutation introduced in [12]:

- HAM(π), measuring the number of elements at correct position, which is the number of indices i such that $\pi(i) = i$,
- RUN(π), measuring the number of maximally sorted blocks, which is the number of indices i such that $\pi(i + 1) < \pi(i)$ plus one,
- EXC(π), measuring the minimal number of pairwise exchanges in π, in order to sort the sequence.

Note that $EXC(\pi)$ can be computed in linear time, based on the cycle structure of permutations. If the sequence is sorted, it has n cycles. Otherwise, it is always possible to increase the number of cycles by exchanging an element that is not sitting at its correct position with the element that is currently sitting there. For any given permutation π consisting of $n - k$ cycles, $EXC(\pi) = k$.

We do not consider the functions INV (pairs in order) and LAS (longest ascending sequence) given in [12] as they are easy to be optimized for all the algorithms that we consider.

We will investigate the different measures for variable-length evolutionary algorithms. Consequently, we might have to deal with incomplete permutations as not all elements have to be contained in a given individual. Most measures can also be used for incomplete permutation, but we have to make sure that complete permutations always obtain a better fitness than incomplete ones. Furthermore, the sortedness measure should guide the algorithm from incomplete permutations to complete ones.

Algorithm 1. Derivation of $F(X)$

1 Generate π by parsing X front to rear and adding an element to π only if it is not yet in π;
2 Return $F(\pi)$;

Algorithm 2. (1+1) GP-single for maximization

1 Choose an initial solution X;
2 **repeat**
3 Set $Y := X$;
4 Apply the mutation operator (given in Figure 1) with $k = 1$ to Y;
5 **if** $f(Y) \geq f(X)$ **then** set $X := Y$;

We will use the sortedness measures as above and use the following special fitness assignments that enforce the previously stated properties.

- $\mathrm{RUN}(\pi) = n + 1$ if $|\pi| = 0$, otherwise $\mathrm{RUN}(\pi) = b + m$ is the sum of the number of maximally sorted blocks b, and the number of elements missing $m = n - |\pi|$,
- If $|\pi| \leq n$ then $\mathrm{EXC}(\pi) = e + m + 1$, otherwise $\mathrm{EXC}(\pi) = e$, where e is the number of necessary exchanges e, and $m = n - |\pi|$ the number of elements missing.

Note that e can be computed for incomplete permutations as well, as only the order $<$ on E has to be respected. This means that, the permutations $\pi_1 = (1, 4)$ and $\pi_2 = (1, 2, 3, 4)$ require no changes, but $\mathrm{EXC}(\pi_1) \neq \mathrm{EXC}(\pi_2)$, as the number of missing elements differs.

For example, for a tree X with $\pi = (2, 3, 4, 5, 1, 6)$ and $n = 7$, the sortedness results are $\mathrm{HAM}(X) = 1$, $\mathrm{RUN}(X) = 2 + 1 = 3$, and $\mathrm{EXC}(X) = 4 + 1 + 1 = 6$.

We now define our multi-objective variants of sorting. When adding the complexity of a data-structure as the second measure, we get the problems MO-HAM, MO-RUN, and MO-EXC, respectively. Given a variable length solution X and a problem F, we will refer by $F(X) = F(\pi)$ to its fitness. Here π is obtained from X (see Algorithm 1) by parsing X and adding an element x to π if it is not yet contained in it.

3 Local Optima and the Parsimony Approach

In this section, we consider simple variable length evolutionary algorithms using the parsimony approach. To stress the use of variable-length representations and to make the connection to recent investigations on the computational complexity of genetic programming [3, 9], we will view our algorithms as simple genetic programming algorithms.

The single-objective variant called (1+1) GP-single starts with an initial solution X, and produces in each iteration one single offspring Y by applying the

mutation operator given in Figure 1 with $k = 1$. This means that it is a stochastic hillclimber which explores its local neighborhood. In the case of maximization, Y replaces X if $f(Y) \geq f(X)$ holds. Minimization problems are tackled in the analogous way.

In the following, we show that the parsimony approach leads to local optima for various types of sortedness measure.

Let $I_1 = (n, 2, 3, \ldots, n-3, n-2, n-1, 1)$ be the initial solution. We point out that this is a local optimum for (1+1) GP-single on MO-EXC leading to an infinite optimization time.

Theorem 1. *Let I_1 be an initial solution. Then the optimization time of (1+1) GP-single on MO-EXC is infinite.*

Proof. The individual I_1 has an EXC-value of 1 and a length of n. In order to reduce the fitness down to 0, it would be necessary to move the n from the head of the permutation to its end.

For this to happen, deletions and substitutions cannot be considered, as they would produce incomplete permutations, and incomplete permutations have EXC-values of at least 2.

Similarly, this situation cannot be solved using a single insertion: it is not possible to introduce n at its correct position within the permutation, as the existing n is preventing the new one from becoming expressed.

Therefore, it it not possible to improve the number of elements sitting at their correct (relative) position via a single mutation. Thus, (1+1) GP-single takes infinitely long, when initialized with I_1. □

We continue by investigating the sortedness measure RUN. Without loss of generality, let n be even and $I_2 = \left(\frac{n}{2}+1, \frac{n}{2}+2, \ldots, n-1, n, 1, 2, \ldots, \frac{n}{2}-1, \frac{n}{2}\right)$ be an initial solution. The following theorem shows that I_2 is a local optimum for (1+1) GP-single on MO-RUN.

Theorem 2. *Let I_2 be the initial solution. Then the optimization time of (1+1) GP-single on MO-RUN is infinite.*

Proof. The individual I_2 has a RUN-value of 2, which cannot be improved via a single insertion: $n/2$ elements have to change their positions in the inorder parsed list that is used for the computation of the RUN-value. Furthermore, a single deletion or substitution results in a worse sortedness value as one element is then missing (as defined in Section 2.1). Therefore, the runtime of the single-operation case of (1+1) GP is infinite, when initialised with this particular individual. □

Finally, we consider the sortedness measure *HAM* and investigate the initial solution $I_3 = (1, n-2, 3, 4, 5, \ldots, n-3, 2, n-1, n)$. We show that this is a local optimum for MO-HAM.

Theorem 3. *Let I_3 be the initial solution. Then the optimization time of (1+1) GP-single on MO-HAM is infinite.*

Algorithm 3. SMO-GP

```
1 Choose an initial solution X;
2 Set P := {X};
3 repeat
4     Choose X ∈ P uniformly at random;
5     Set Y := X;
6     Apply mutation to Y;
7     if {Z ∈ P | Z ⪰ Y} = ∅ then set P := (P \ {Z ∈ P | Z ≻ Y}) ∪ {Y};
```

Proof. The individual I_3 has the elements 2 and $n-2$ at incorrect positions, resulting in a HAM-value of $n-2$. It is not possible to maintain the HAM-value (or improve it) via deletions, as they decrease the number of elements at correct positions. Substitutions can also not maintain the HAM-value. A substitution of the $n-2$ by a 2 would result in the elements to the right to shift away from their correct position as in both cases the element at the third position would no longer get expressed. This leaves only the option of using insertions, in order to generate an individual that is accepted. The element $n-2$ cannot be introduced successfully at its correct position as its current occurrence is blocking a later expression. If the 2 is introduced at its correct position, then the resulting permutation is $(1, 2, n-2, 3, 4, 5, \ldots, n-3, n-1, n)$ as the second 2 is no longer gets expressed, and the corresponding HAM-value for this permutation is 4.

Thus, the runtime of the single-operation case of (1+1) GP is infinite, when initialised with this particular individual. □

4 Multi-objective Approach

We consider the Simple Evolutionary Multi-Objective Genetic Programming (SMO-GP) algorithm introduced in [9] and motivated by the SEMO algorithm for fixed length representations of Laumanns et al. [8]. Variants of SEMO have been frequently used in the runtime analysis of evolutionary multi-objective optimization for fixed length representations [4, 5, 10, 11]. SMO-GP (see Algorithm 3) is a population-based approach that starts with a single solution and keeps in each iteration a set of non-dominated solutions obtained during the optimization run. In each iteration, it picks one solution uniformly at random and produces one offspring Y by mutation. Y is introduced into the population iff it is not weakly dominated by any other solution in P. If Y is added to the population all individuals that are dominated by Y are discarded.

SMO-GP-single uses the mutation operator given in Figure 1 with k=1. We also consider SMO-GP-multi which differs from SMO-GP-single by choosing k according to $1 + Pois(1)$.

In this section, we analyze the performance of the SMO-GP variants on each one of the fitness functions introduced in Section 2.1. In particular, we analyze the expected number of iterations to compute the optimal solution. We call this the *expected optimization time* of the algorithms.

The following lemma bounds the expected time until the empty solution has been included into the population, when considering an arbitrary optimization problem:

Lemma 1 (Neumann [9]). *Let I_{init} be the size of the initial solution and k be the number of different fitness values of a problem F. Then the expected time until the population of SMO-GP-single and SMO-GP-multi applied to MO-F contains the empty solution is $O(kI_{init})$.*

Theorem 4. *The expected optimization time of SMO-GP-single and SMO-GP-multi is $O(nI_{init} + n^3 \log n)$ on MO-EXC and MO-RUN and $O(nI_{init} + n^4)$ on MO-HAM.*

Note that for all three problems, only solutions of complexity i, $0 \leq i \leq n$ can be Pareto optimal. For RUN, these are solutions X with $C(X) = i$ and $\text{RUN}(X) = n + 1 - i$, $0 \leq i \leq n$.

Proof. In the following, we will first prove the theorem for MO-RUN. The proofs for MO-EXC and MO-HAM follow the same structure.

RUN has $n+1$ different fitness values. Using Lemma 1, the empty solution is produced after an expected number of $O(kI_{init})$ steps.

In the following steps, we will bound the time needed to discover the whole Pareto front, once the empty solution is introduced into the population. Let us assume that the population contains all Pareto optimal solutions with complexities j, $0 \leq j \leq i$. Then, a population which includes all Pareto optimal solutions with complexities j, $0 \leq j \leq i+1$, can be achieved by producing a solution Y that is Pareto optimal and that has complexity $i+1$. Y can be obtained from a Pareto optimal solution X with $C(X) = i$ by inserting any of the $n - i$ missing elements into the correct position. This operation produces from a solution of complexity i a solution of complexity $i + 1$.

Based on this idea we can bound the expected optimization time once we can bound the probability for such steps to happen. Choosing X for mutation has probability at least $1/(n + 1)$ as the population size is upper bound by $n + 1$. Next, the mutation step carrying out just one operation happens with at least $1/e$, and the inserting operation of the mutation operator is chosen with probability $1/3$. As $n - i$ out of the n elements are missing, any of those can be inserted. However, the correct position for such a randomly chosen element has to be chosen, in order to produce the Pareto optimal solution of complexity $i+1$. This probability is at least $1/2 \cdot 1/n$, as the number of leaf nodes is bound by n, and the probability to insert as the correct child of the newly introduced inner node is at least $1/2$. Thus, the total probability of such a generation is

$$\frac{1}{n+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{n-i}{n}.$$

Now, we use the method of fitness-based partitions [15] according to the $n + 1$ different fitness values of i. Thus, as there are only n Pareto-optimal improvements possible once the empty solution is introduced into the population, the expected time until all Pareto optimal solutions have been generated is:

$$\sum_{i=0}^{n} \left(\frac{1}{n+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{n-i}{n} \right)^{-1} = 6en^2(n+1) \cdot \sum_{i=0}^{n} \frac{1}{n-i} = O(n^3 \log n).$$

Taking into account the expected time to produce the empty solution, the expected time until the whole Pareto front of MO-RUN has been computed is $O\left(nI_{init} + n^3 \log n\right)$.

The proof for MO-EXC follows the same structure. First, note that if the ordering within the permutation requires an exchange, then this individual is dominated by individuals of same complexity that require fewer exchanges. Just as with MO-EXC, let us assume that the population contains all Pareto optimal solutions with complexities j, $0 \leq j \leq i$. Then, a population which includes all Pareto optimal solutions with complexities j, $0 \leq j \leq i+1$, can be achieved by inserting any of the $n-i$ missing elements into the correct position of the Pareto optimal individual X with $C(X) = i$. The probability for such a step to happen is at least $\frac{1}{n+2} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{n-i}{n}$. Now, as $n+2$ different EXC-values are possible, and by summing up the waiting times as done for MO-RUN, the expected optimization time is $O(nI_{init} + n^3 \log n)$.

Similarly, we can prove an upper bound for MO-HAM. First, note that each Pareto optimal solution with HAM-value i represents a perfectly sorted permutation of the i elements $1, \ldots, i$. Just as above, let us assume that the population contains all Pareto optimal solutions with complexities j, $0 \leq j \leq i$. Then, a population which includes all Pareto optimal solutions with complexities j, $0 \leq j \leq i+1$, can be achieved by inserting the element $i+1$ into its correct position (i. e., as the rightmost leaf) in the Pareto optimal individual X with $\mathrm{HAM}(X) = C(X) = i$. The probability for such a step to happen is at least $\frac{1}{n+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{1}{n} = \Omega\left(\frac{1}{n^3}\right)$ and the corresponding waiting time is $O(n^3)$. There are $n+1$ different HAM-values. This implies that the expected optimization time is $O(nI_{init} + n^4)$. □

5 Conclusions

Variable length representations are frequently used in evolutionary algorithms. The most prominent example using such a representation is genetic programming. With this paper, we have contributed to the theoretical understanding when using such a representation. We discussed two methods for dealing with bloat which frequently occurs when using such a representation. To point out the differences between these two approaches, we examined different measures of sortedness that have been analyzed for evolutionary algorithms with fixed length representations. Our analysis for the parsimony approach shows that variable length representations might have difficulties when dealing with simple measures of sortedness due to the presence of local optima. Our runtime analysis for simple multi-objective algorithms shows that they compute the whole Pareto front for the examined sortedness measures in expected polynomial time.

References

[1] Auger, A., Doerr, B. (eds.): Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific (2011)
[2] Cathabard, S., Lehre, P.K., Yao, X.: Non-uniform mutation rates for problems with unknown solution lengths. In: FOGA, pp. 173–180. ACM, New York (2011)
[3] Durrett, G., Neumann, F., O'Reilly, U.-M.: Computational complexity analysis of simple genetic programing on two problems modeling isolated program semantics. In: FOGA, pp. 69–80. ACM (2011)
[4] Friedrich, T., He, J., Hebbinghaus, N., Neumann, F., Witt, C.: Approximating covering problems by randomized search heuristics using multiobjective models. Evolutionary Computation 18(4), 617–633 (2010)
[5] Giel, O., Lehre, P.K.: On the effect of populations in evolutionary multiobjective optimisation. Evolutionary Computation 18(3), 335–356 (2010)
[6] Kötzing, T., Sutton, A., Neumann, F., O'Reilly, U.-M.: The Max problem revisited: the importance of mutation in genetic programming. In: GECCO (to appear, 2012)
[7] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
[8] Laumanns, M., Thiele, L., Zitzler, E.: Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. IEEE Trans. Evolutionary Computation 8(2), 170–182 (2004)
[9] Neumann, F.: Computational complexity analysis of multi-objective genetic programming. In: GECCO (to appear, 2012), http://arxiv.org/abs/1203.4881
[10] Neumann, F., Wegener, I.: Minimum spanning trees made easier via multi-objective optimization. In: GECCO, pp. 763–770. ACM Press (2005)
[11] Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity. Springer (2010)
[12] Scharnow, J., Tinnefeld, K., Wegener, I.: The analysis of evolutionary algorithms on sorting and shortest paths problems. Journal of Mathematical Modelling and Algorithms 3, 349–366 (2004)
[13] Shukla, P.K., Deb, K.: On finding multiple Pareto-optimal solutions using classical and evolutionary generating methods. European Journal of Operational Research 181(3), 1630–1652 (2007)
[14] Urli, T., Wagner, M., Neumann, F.: Experimental Supplements to the Computational Complexity Analysis of Genetic Programming for Problems Modelling Isolated Program Semantics. In: Coello Coello, C.A., et al. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 102–112. Springer, Heidelberg (2012)
[15] Wegener, I.: Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In: Evolutionary Optimization. International Series in Operations Research and Management Science, vol. 48, pp. 349–369. Springer, US (2003)

An Evolutionary and Graph-Based Method for Image Segmentation

Alessia Amelio[1,2] and Clara Pizzuti[1]

[1] Institute for High Performance Computing and Networking,
National Research Council of Italy, CNR-ICAR,
Via P. Bucci 41C, 87036 Rende (CS), Italy
pizzuti@icar.cnr.it
[2] DEIS, Università della Calabria
Via P. Bucci 41C, 87036 Rende (CS), Italy
aamelio@deis.unical.it

Abstract. A graph-based approach for image segmentation that employs genetic algorithms is proposed. An image is modeled as a weighted undirected graph, where nodes correspond to pixels, and edges connect similar pixels. A fitness function, that extends the normalized cut criterion, is employed, and a new concept of nearest neighbor, that takes into account not only the spatial location of a pixel, but also the affinity with the other pixels contained in the neighborhood, is defined. Because of the locus-based representation of individuals, the method is able to partition images without the need to set the number of segments beforehand. As experimental results show, our approach is able to segment images in a number of regions that well adhere to the human visual perception.

1 Introduction

Image segmentation is an important problem in pattern recognition that aims at partitioning an image into uniform regions [4]. More formally, the problem can be stated as follows: let R be an image constituted by a set of pixels. Segmenting the image R consists in subdividing R into a finite number of non-overlapping and connected regions $R_1 ... R_s$ such that

$$R = \bigcup_{i=1}^{S} R_i, R_i \cap R_j = \emptyset, i \neq j$$

A homogeneity measure must be defined over pixels that takes into account characteristics such as intensity, color, or texture. Pixels belonging to the same region are similar on the base of the homogeneity measure adopted, while adjacent regions are significantly dissimilar with respect to the same features.

1.1 Related work

The image segmentation problem has been intensively investigated with the use of several computational techniques, and many different methods have been proposed. A broad classification divides the existing methods in two main categories [20]: boundary

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 143–152, 2012.

detection-based approaches and region clustering-based approaches. The former approaches search for closed boundary contours by detecting pixels that sensibly change in intensity. Boundaries of objects are obtained by linking such pixels in contours. The main limitation of these approaches is that a threshold value must be set in order to produce a continuous contour [16,8]. Region cluster-based methods group similar closed pixels into clusters. Many of these approaches use Fuzzy C-means [2] or the K-means method, such as [1,13]. A drawback of these methods is that the number of clusters must be predetermined, which implies that a user should know in advance the region number of the image to segment. In order to overcome these limitations, methods based on representing an image as a graph have been introduced. One of the earliest graph-based methods dates back over 40 years and it is based on the minimum spanning tree (MST) of a graph [21]. Zahn's method gives a weight to edges on the base of differences between pixel intensities, and breaks large edges by fixing a threshold. Improvements on the policy of edge breaking were proposed by Urquhart in [18]. Wu and Leahy [19] presented a method based on the minimization of the concept of cut, which is the weight of edges connecting two regions. To avoid unnatural cuts of small groups of isolated nodes, Shi and Malik [17] introduced a new measure of dissimilarity between two groups named *normalized cut*. More recently, Felzenszwalb and Huttenlocher [5] defined a measure of evidence of the boundary between two regions by considering both the differences of intensity across the boundary and among neighboring pixels within a region.

1.2 Evolutionary-Based Related Work

In the last years much effort has been done in the definition of effective evolutionary-based approaches for solving complex problems of computer vision. In particular, evolutionary techniques have been successfully applied to the image segmentation problem, that is of prior importance for facing more complex higher level problems such as *Object Recognition*. A survey on the application of genetic algorithms for image enhancement and segmentation can be found in [15]. Many of the approaches use a representation of the image based either on the cluster centers or on the label of the cluster a pixel is assigned to. A color image segmentation algorithm based on evolutionary approach has been proposed by Halder and Pathak in [7]. Each individual is a sequence of cluster centers and the cost function is the inverse of the sum of Euclidean distances of each point from their respective cluster centers. In order to determine the most appropriate number k of clusters, the algorithm is repeated for values of k equals to 2 until k_{max}. The choice of the best k is done by computing a cluster validity index based on inter and intra distances between clusters, while the value of k_{max} must be given as input to the algorithm. Jiao in [9] proposed an evolutionary image texture classification algorithm where the individuals are the cluster representatives, and the total distance between the pixels and the corresponding centroids is optimized. The distance for a couple of pixels is the value of the shortest path between them in the undirected weighted graph representing the image. In the same paper the author defines a Memetic Image Segmentation approach, where a genetic algorithm is applied on a set of regions previously extracted from a watershed segmentation in order to refine or merge the partitions into clusters. In this case each gene of a chromosome is the cluster label of the corresponding pixel.

The association of the regions with the clusters is evolved by optimizing the total distance between the pixels and the corresponding centroids. In the former approach the number of clusters must be fixed a priori, while in the latter an approximate initial number is obtained by using the watershed segmentation, and then a final local search procedure merges regions to obtain the optimal number of clusters. Lai and Chang [10] proposed a hierarchical structure of the chromosome, composed by control genes, representing a partitioning in regions, and parametric genes, containing the representative gray levels of each region. The goal is to optimize a fitness function that is the sum of the distances between the gray level of each pixel and the representative gray level of its region. The number of control genes, as stated by the authors, is a $soft$ estimate of the upper bound of the number of regions. Merzougui et al. [12] proposed an evolutionary based image segmentation technique where the individuals are the components of the cluster centers and the fitness is the mean distance between the pixels and the centroids. In order to determine the optimal number of clusters, a criterion based on separability and compactness of the clusters is first applied. Di Gesú and Bosco [6] introduced an image segmentation algorithm where each chromosome represents the position and the region label where the pixel is located. The fitness function is defined on the similarity value and the spatial proximity between a pixel (chromosome) and the mean gray value of its corresponding region.

Because of the representation adopted, one of the main problems of the described approaches is the determination of the number of regions. Though different criteria are used to fix this number beforehand, the genetic algorithm cannot change this number while executing. The method we propose in the following dynamically computes the number of regions that optimizes the fitness function.

1.3 Contributions

In this paper we present a new graph-based algorithm, named *GeNCut* (*Genetic NCut*), to solve the image segmentation problem by using an evolutionary approach. In particular, we represent an image as a weighted undirected graph, then a genetic algorithm optimizing a fitness function is executed in order to find an optimal partitioning of the graph and, consequently, a good segmentation of the image. The fitness function is an extension of the normalized cut concept of Shi and Malik [17] that allows for a simultaneous k-way partitioning of the image without the need of fixing the number k of divisions beforehand, which is typical of many image segmentation approaches. In fact, because of the locus-based representation of individuals adopted, k is automatically determined by the optimal value of the objective function. Experiments on images of different difficulty show that *GeNCut* outperforms the method of Shi and Malik by partitioning natural and human scenes in meaningful objects.

The paper is organized as follow. In the next section the problem of image segmentation is defined together with its formalization as a graph partitioning problem and a description of the homogeneity measure adopted. Section 3 introduces the concept of normalized cut and the fitness function used by *GeNCut*. Section 4 explains the genetic representation and operators employed. Section 5 presents the experimental results. Finally, section 6 summarizes the approach presented and outlines future work.

2 Problem definition

An image R can be represented as a weighted undirected graph $G = (V, E, w)$, where V is the set of the nodes, E is the set of edges in the graph, and $w : E \rightarrow \mathcal{R}$ is a function that assigns a value to graph edges. Each node corresponds to a pixel in the image, and a graph edge (i, j) connects two pixels i and j, provided that these two pixels satisfy some property suitably defined that takes into account both pixel characteristics and spatial distance. The weight $w(i, j)$ associated with a graph edge (i, j) represents the likelihood that pixels i and j belong to the same image region and provides a similarity value between i and j. The higher the value of $w(i, j)$, the more likely the two pixels are members of the same region. Let W be the adjacency weight matrix of the graph G. Thus W_{ij} contains the weight $w(i, j)$ if the nodes i and j are connected, zero otherwise. Depending on the method adopted to compute the weights, any two pixels may or may not be connected. In our approach we employed the *Intervening Contour* method described in [11,3]. In this framework, given a generic pixel, the magnitude of the orientation energy at that pixel is considered. If the maximum image edge magnitude along a straight line connecting the two pixels i and j in the image plan is large, then a deep change and, consequently, an intervening contour is present, indicating that the two pixels don't belong to the same segment. Hence, the weight $w(i, j)$ between these pixels will be low. On the other hand, if the image edge magnitude is sufficiently weak, this usually happens in a region that is flat in brightness, the affinity between the two pixels will be very high. More formally, the weight $w(i, j)$ between the pixels i and j is computed as:

$$w(i,j) = \begin{cases} e^{-max_{x \in line(i,j)} ||Edge(x)||^2 / 2a^2} & \text{if } ||X(i) - X(j)||_2 < r, i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

where $a = (max_{y \in I} ||Edge(y)||) \times \sigma$, *Edge(x)* is the image edge strength at position *x*, I is the image plan, *line(i,j)* is a straight line between i and j, *X(i)* is the spatial location of the pixel i, *r* is a distance threshold and σ is the image edge variance. In order to compute the weight between the pixels i and j, image edges across various scales are considered.

3 Objective Function

In the last few years many different criteria have been defined to partition a graph representing an image into non-overlapping connected components. Shi and Malik [17] introduced the dissimilarity measure *normalized cut* to divide a graph into two subgraphs, that revealed successful for image segmentation. The concept of normalized cut is an extension of the notion of *cut* proposed by Wu and Leahy [19] that avoids the bias for partitioning in small sets of nodes. Given a partition of a graph G in two disjoint sets of nodes A and B, the cut between A and B is defined as

$$cut(A, B) = \sum_{i \in A, j \in B} w(i, j)$$

In [17] the authors pointed out that the cut value diminishes when small sets of isolated nodes are generated. Thus a disassociation measure, that takes into account the total edge weight connecting two partitions, has been introduced. Let

$$assoc(A, V) = \sum_{i \in A, t \in V} w(i, t)$$

be the total connection from nodes in A to all the nodes in V, then the normalized cut is defined as

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

Shi and Malik formalize the problem of minimizing the normalized cut as a generalized eigenvalue problem and compute an optimal partition by using the eigenvector with the second smallest eigenvalue. Two extensions of the approach to k-way partitioning are also proposed. The former recursively partitions the groups obtained in the previous step by checking the values of the eigenvectors, the latter exploits the top n eigenvectors and the clustering algorithm K-means. A main limitation of this method is that the number k of desired partitions must be fixed beforehand.

We now introduce an extension of the concept of normalized cut that is used as criterion to partition a graph in a generic number k of regions. Note that, the value of k in our approach must not be fixed in advance, but it is determined by the optimal value of the objective function. Let $G = (V, E, w)$ be the graph representing an image, W its adjacency matrix, and $P = \{S_1, \ldots, S_k\}$ a partition of G in k clusters.

For a generic cluster $S \in P$, let

$$c_s = \sum_{i \in S, j \notin S} W_{ij} \qquad m_s = \sum_{i \in S, j \in S} W_{ij} \qquad m = \sum_{i \in V, j \in V} W_{ij}$$

be respectively the sum of weights of edges on the boundary of S, the sum of weights of edges inside S, and the total graph weight sum. The *weighted normalized cut WNCut* measures for each cluster $S \in P$ the fraction of total edge weight connections to all the nodes in the graph

$$WNCut = \sum_{s=1}^{k} \frac{c_s}{m_s + c_s} + \frac{c_s}{(m - m_s) + c_s}$$

Note that c_s corresponds to $cut(A, B)$ where $B = V - A$. Because of the affinity measure w defined in the previous section, and the relationship between cut and $assoc$ formalized in [17], more uniform regions can be obtained with low cut values between the subgraphs representing the regions and the rest of the graph. This implies that low values of $WNcut$ should be preferred.

4 Genetic Representation and Operators

The genetic algorithm uses the locus-based adjacency representation proposed in [14]. In this graph-based representation an individual of the population consists of N genes

$g_1, \ldots, g_N$ and each gene can assume allele values j in the range $\{1, \ldots, N\}$. Genes and alleles represent nodes of the graph $G = (V, E, w)$ modelling an image, and a value j assigned to the ith gene is interpreted as a link between the pixels i and j. This means that in the clustering solution found i and j will belong to the same region.

The initialization process assigns to each node i one of its neighbors j. This guarantees a division of the graph in connected groups of nodes. The kind of crossover operator adopted is uniform crossover. Given two parents, a random binary vector is created. Uniform crossover then selects the genes where the vector is a 0 from the first parent, and the genes where the vector is a 1 from the second parent, and combines the genes to form the child. The mutation operator, analogously to the initialization process, randomly assigns to each node i one of its neighbors.

The genetic operators need to determine the neighbors of each node. In our approach we introduced the concept of neighbors of a node by taking into account not only the spatial closeness, but also the pixel affinity. More in details, given a generic node i in the graph, let $w^h_{max} = \{w^1, \ldots, w^h \mid w^1 \geq, \ldots, \geq w^h\}$ be the first h highest weights of row i in the weight adjacency matrix W.

The h nearest neighbors of i, denoted as nn^h_i, are then defined as $nn^h_i = \{j \mid w(i,j) \in w^h_{max}\}$.

nn^h_i is thus the set of those pixels that are no more than r pixels apart from i, and that have maximum similarity with i. It is worth to note that, even if h is fixed to 1, the number of nearest neighbors of i could be sufficiently large if many of its spatial neighbors have the same maximum weight w^h_{max}. This definition of nearest neighbors guaranties to choose the most similar neighbors during the initialization process, and to bias the effects of the mutation operator towards the most similar neighbors, thus it contributes to improve the results of the method.

5 Experimental Results

In this section we present the results of *GeNCut* on five images with details of increasing complexity, and compare the performances of our algorithm in partitioning natural and human scenes in meaningful objects with the segmentations obtained by the algorithm of Shi and Malik [17] (in the following referred as $NCut$) on the same images. The *GeNCut* algorithm has been written in MATLAB 7.14 R2012a, using the Genetic Algorithms and Direct Search Toolbox 2. In order to set parameter values, a trial and error procedure has been employed and then the parameter values giving good results for the benchmark images have been selected. Thus we set crossover rate to 0.9, mutation rate to 0.2, elite reproduction 10% of the population size, roulette selection function. The population size was 100, the number of generations 50. The value h of nearest neighbors to consider has been fixed to either 1 or 2. As already pointed out, this does not mean that the number of neighbors is 1 or 2, but that the first (and second) most similar neighbors are taken into account for the initialization and mutation operators. The fitness function, however, is computed on the overall weight matrix. For all the data sets, the statistical significance of the results produced by *GeNCut* has been checked by performing a t-test at the 5% significance level. The p-values returned are very small, thus the significance level is very high since the probability that

a segmentation computed by *GeNCut* could be obtained by chance is very low. The version of the $Ncut$ software that we used is written in MATLAB and it is available at http://www.cis.upenn.edu/ jshi/software/. The weight matrix of each image is the same for both methods, and, as already described in section 2, it is based on the Intervening Contour framework by fixing $r = 10$, number of scales 3, number of orientations 4 and $\sigma = 0.1$. Since $NCut$ needs the number k of clusters, we executed the algorithm by using two different inputs. The first sets the number k of segments to the same number of clusters found by *GeNCut*, the second one is a higher value. In the following, for each image, we compare the segmentation results of *GeNCut* and $NCut$ by depicting the contours of the regions obtained by the two approaches. For a more clear visualization, we show two images. The first image reports the boundary lines of the segmentation obtained on the original color image, the second one delineates the same contours without the image background.

Fig. 1 shows the execution of *GeNCut* and $NCut$ on an image of a melanoma. In particular, *Fig.* 1(a) is the original image, *Fig.* 1(b) and *Fig.* 1(c) display the segmentation obtained by *GeNCut* with and without the background image resp., while *Fig.* 1(d) and *Fig.* 1(e) are the results of $NCut$ when the number of segments is fixed to two, and *Fig.* 6(a) when this number is 5. *Fig.* 1(b)(c) show that *GeNCut* is able to find the right partitioning and correctly discriminates between the melanoma and the skin, although we don't set a priori the number of segments. *Fig.* 1(d)(e) and *Fig.* 6(a) point out that if $NCut$ receives the true number of segments, it is able to find the correct partitioning, otherwise an over-segmentation is obtained. In general, however, given an input image, it is hard to know a priori the true number of partitions.

The next experiment represents a more complex scenario, due to the presence of irregular shapes (clouds) around a spherical object (moon) (*Fig.* 2(a)). *Fig.* 2(b)(c) illustrate

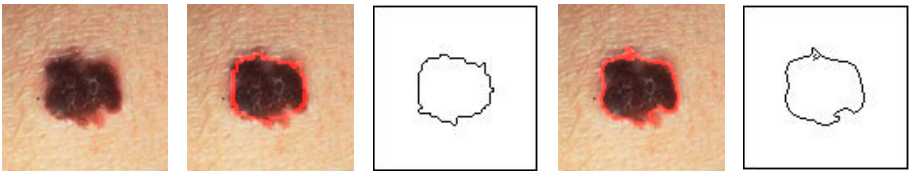


Fig. 1. (a) The original image representing a melanoma, (b) the segmentation result on the original image of *GeNCut* with h=2, (d) $NCut$ with $k = 2$ and (c-e) the corresponding contours

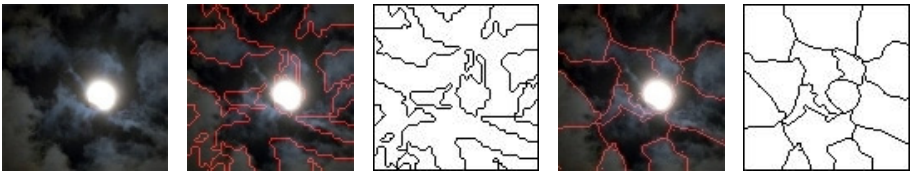


Fig. 2. (a) The original moon image, (b) the segmentation results on the original image using *GeNCut* with h=1, (d) $NCut$ with $k = 16$ and (c-e) the corresponding contours

the results obtained by using the *GeNCut* approach, while *Fig.* 2(d)(e) and *Fig.* 6(b) the $NCut$ method when the number of segments is set to 16 and 22, respectively. Although the halo makes difficult to segment the moon, by using our algorithm we are able to perform a segmentation that is more flexible in capturing the real shape of the clouds. $NCut$, instead, realizes a more flat partitioning with an equal number of segments that is not able to distinguish and capture some inner parts of the original image. *Fig.* 3(a) and *Fig.* 4(a) show two different kinds of landscapes: a natural picture and a snatch from an X-SAR image of the Vesuvius volcano (Italy), acquired by the Spaceborne Imaging Radar-C/X-Band Synthetic Aperture Radar (SIR- C/X-SAR) aboard the Space Shuttle Endeavour in 1994. For both the images, our algorithm is able to discover the meaningful objects, *Fig.* 3(b)(c) and *Fig.* 4(b)(c), respectively, while a poor segmentation of the major components like in *Fig.* 3(d)(e) and *Fig.* 6(c) is obtained, despite the setting of the same number of segments, naturally extracted from our technique. The satellite image in *Fig.* 4(a) is a scene where it is quite difficult to differentiate the meaningful objects due to the details of the terrain. However, as it can be observed in *Fig.* 4(b)(c), *GeNCut* is able to separate the volcano area from the landscape and to distinguish the building barely visible at the bottom right corner and the area of the deep sea. All these significative features are not visible in the segmentation results of the $NCut$ approach, *Fig.* 4(d)(e), even if we increase the number of partitions, *Fig.* 6(d). Finally, we used *GeNCut* to segment a human face image (*Fig.* 5(a)). In this case the two approaches are comparable. Although more details are discovered from $NCut$ in correspondence of the eyes, it over-segments the face (*Fig.* 5(d)(e) and *Fig.* 6(e)). On the other hand, *GeNCut* obtains a uniform and natural segmentation of the face (*Fig.* 5(b)(c)) that is able to capture also the shape of the nose, although it appears linked to the eyes, probably due to the similar gray intensities along the contours of the nose and the contours of the eyes.

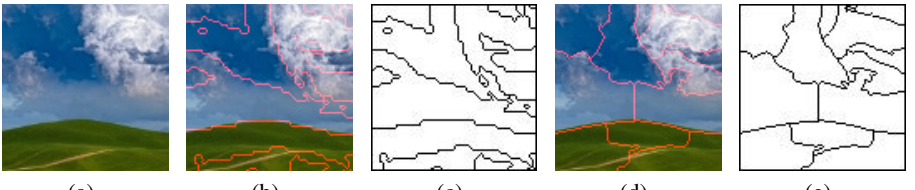


Fig. 3. (a) The original image, (b) the segmentation results on the original image using *GeNCut* with h=1, (d) $NCut$ with $k = 12$ and (c-e) the corresponding contours

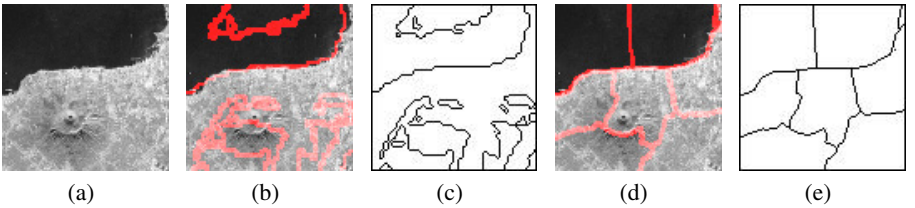
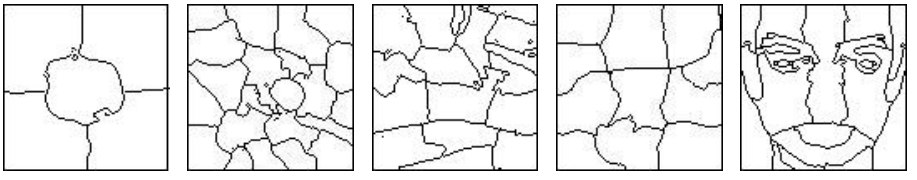


Fig. 4. (a) An X-SAR image of the Vesuvius volcano, (b) the segmentation results on the original image using *GeNCut* with h=1, (d) $NCut$ with $k = 8$, and (c-e) the corresponding contours

(a) (b) (c) (d) (e)

Fig. 5. (a) The original face image, (b) the segmentation results on the original image using *GeNCut* with h=2, (d) $NCut$ with $k = 12$ and (c-e) the corresponding contours



(a) (b) (c) (d) (e)

Fig. 6. The segmentation results representing the contours using $NCut$ with (a) $k = 5$, (b) $k = 22$, (c) $k = 20$, (d) $k = 12$, (e) $k = 20$

6 Conclusions

The paper presented a graph-based approach to image segmentation that employs genetic algorithms. A fitness function, that extends the normalized cut criterion introduced in [17], is proposed, and a new concept of nearest neighbor, that takes into account not only the spatial location of a pixel, but also the affinity with the other pixels contained in the neighborhood, is defined. The locus-based representation of individuals, together with the fitness function adopted, revealed particularly apt to deal with images modeled as graphs. In fact, as experimental results showed, our approach is able to segment images in a number of regions that well adhere to the human visual perception. Future work will extend the segmentation process to partition more complex texture images including medical X-ray and ultrasound images, by combining the contour information considered in our approach with textural features of the regions.

Acknowledgements. This work has been partially supported by the project *MERIT : MEdical Research in Italy*, funded by MIUR.

References

1. Chen, C.W., Luo, J., Parker, K.J.: Image segmentation via adaptive k-means clustering and knowledge-based morphological operations with biomedical applications. IEEE Transactions on Image Processing 7(12), 1673–1683 (1998)
2. Chen, S., Zhang, K.: Robust image segmentation using fcm with spatial constrains based on a new kernel-induced distance measure. IEEE Transactions on Systems Man and Cybernetics B 34, 1907–1916 (2004)

3. Cour, T., Bénézit, F., Shi, J.: Spectral segmentation with multiscale graph decomposition. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005)), pp. 1124–1131 (2005)
4. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. John Wiley & Sons, New York (1973)
5. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. International Journal of Computer Vision 59(2), 167–181 (2004)
6. Di Gesú, V., Lo Bosco, G.: Image Segmentation Based on Genetic Algorithms Combination. In: Roli, F., Vitulano, S. (eds.) ICIAP 2005. LNCS, vol. 3617, pp. 352–359. Springer, Heidelberg (2005)
7. Halder, A., Pathak, N.: An evolutionary dynamic clustering based colour image segmentation. International Journal of Image Processing 4, 549–556 (2011)
8. Helterbrand, J.D.: One pixel-wide closed boundary identification. IEEE Transactions on Image Processing 5(5), 780–783 (1996)
9. Jiao, L.: Evolutionary-based image segmentation methods. Image Segmentation (10), 180–224 (2011)
10. Lai, C.-C., Chang, C.-Y.: A hierarchical evolutionary algorithm for automatic medical image segmentation. Expert Syst. Appl. 36(1), 248–259 (2009)
11. Leung, T., Malik, J.: Contour Continuity in Region Based Image Segmentation. In: Burkhardt, H.-J., Neumann, B. (eds.) ECCV 1998. LNCS, vol. 1406, pp. 544–559. Springer, Heidelberg (1998)
12. Merzougui, M., Allaoui, A.E., Nasri, M., Hitmy, M.E., Ouariachi, H.: Evolutionary image segmentation by pixel classification and the evolutionary Xie and Beni criterion - application to quality control. International Journal of Computational Intelligence and Information Security 2(8), 4–13 (2011)
13. Pappas, T.N.: An adaptive clustering algorithms for image segmentation. IEEE Transactions on Signal Processing 40(4), 901–914 (1992)
14. Park, Y.J., Song, M.S.: A genetic algorithm for clustering problems. In: Proc. of 3rd Annual Conference on Genetic Algorithms, pp. 2–9 (1989)
15. Paulinas, M., Uinskas, A.: A survey of genetic algorithms applications for image enhancement and segmentation. Information Technology And Control, Kaunas, Technologija 36(3), 278–284 (2007)
16. Sahoo, P.K., Soltani, S., Wong, A.K.C.: A survey of thresholding techniques. CVGIP 41, 233–260 (1988)
17. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(8), 888–905 (2000)
18. Urquhart, R.: Graph theoretical clustering based on limited neighborhood sets. Pattern Recognition 15(3), 173–187 (1982)
19. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: Theory and applications to image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(11), 1101–1113 (1993)
20. Xu, Y., Olman, V., Uberbacher, E.C.: A segmentation algorithm for noisy images: Design and evaluation. Pattern Recognition Letters 19, 1213–1224 (1998)
21. Zahn, C.T.: Graph theoretical methods for detecting and describing gestalt clusters. IEEE Transactions on Computers 20(1), 68–86 (1971)

Real-Time GPU Based Road Sign Detection and Classification

Roberto Ugolotti, Youssef S.G. Nashed, and Stefano Cagnoni

Department of Information Engineering, University of Parma,
Viale G.P. Usberti 181/A, 43124 Parma, Italy

Abstract. This paper presents a system for detecting and classifying road signs from video sequences in real time. A model-based approach is used in which a prototype of the sign to be detected is transformed and matched to the image using evolutionary techniques. Then, the sign detected in the previous phase is classified by a neural network. Our system makes extensive use of the parallel computing capabilities offered by modern graphics cards and the CUDA architecture for both detection and classification. We compare detection results achieved by GPU-based parallel versions of Differential Evolution and Particle Swarm Optimization, and classification results obtained by Learning Vector Quantization and Multi-layer Perceptron. The method was tested over two real sequences taken from a camera mounted on-board a car and was able to correctly detect and classify around 70% of the signs at 17.5 fps, a similar result in shorter time, compared to the best results obtained on the same sequences so far.

Keywords: Road Sign Classification, Differential Evolution, Particle Swarm Optimization, Learning Vector Quantization, Neural Networks, GPGPU.

1 Introduction

Automatic road sign detection and classification is a task that can help drivers and increase road safety. For this reason, this problem has been frequently tackled [2], up to systems mounted on recent car models, with limited functionalities.

Solutions to this problem usually include two different stages: the presence of a sign is first detected in the image, then it is classified to precisely recognize its meaning and possibly activate some driving system control. In the detection phase, the features used most frequently to recognize a sign are shape and color. Detection based on RGB color is usually fast but performance degrades when dealing with illumination changes [17] or other artifacts related to image acquisition. These problems can be reduced by performing conversion to other color-spaces like HSV/HSI [13]. Shape-based detection is generally more robust against these problems, besides allowing one to work with gray-scale images. However, it has to struggle against occlusions, different viewing angles or the presence of other artificial objects like commercial signs or buildings. For these reasons, a combination of color and shape information is usually preferred [5,15].

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 153–162, 2012.

The term "road sign classification" is not used consistently in the literature: in fact, some authors reduce the classification problem to what, in this paper, we call detection [15], i.e. distinguishing between a sign and a different object in the scene. In other papers [16], the classification problem takes into account only the classification between different categories of signs (e.g. prohibitory versus warning signs). The classification task considered here is concerned with the distinction of signs within the same category (prohibitory, warning, mandatory), as the category is implicitly determined in the detection phase.

The techniques most frequently used for classification are artificial neural networks [10,13] and support vector machines [9].

In the detection phase of our method a model representing a category of signs is rigidly transformed and then reprojected onto the image using an Inverse Perspective transform: this model-based approach shows good results against several problems that can affect the images, like partial occlusions or color unbalancing. After doing so, a fitness function is calculated to assess the degree of matching between the reprojected model and the image, turning detection into an optimization problem, in which a sign is considered to have been detected when the similarity is above a pre-defined value. To perform such an optimization we considered and compared Differential Evolution (DE) [18] and Particle Swarm Optimization (PSO) [7]. For the classification stage, we used Multi-layer Perceptrons (MLP) [4] and learning vector quantization (LVQ) [8] neural networks. In both phases we relied on GPUs to increase processing speed and to reach real-time performance, implementing our methods in CUDA-C [14], a C language extension for developing parallel routines (*kernels*) that run on GPU.

2 Sign Recognition System

A first implementation of the system, limited to the detection stage of three categories of signs (priority, prohibitory and warning), has been first presented by Mussi et al in [11]. The work described in this paper completed the system, adding the detection of mandatory signs and the classification stage. In addition, a DE-based detection stage has been developed and compared to the one based on PSO.

2.1 Sign Detection

The sign detection stage is based on a generally-applicable object detection algorithm that includes the following steps:

1. Consider some sets of key points, of known coordinates with respect to a reference position, and representative of the shape and colors of specific regions of the object to detect.
2. Translate and rotate the sets to a hypothesized position visible by the camera and project them onto the image.
3. Verify that the color histograms of the sets match those of their projection on the image to assess the presence of the object being sought.

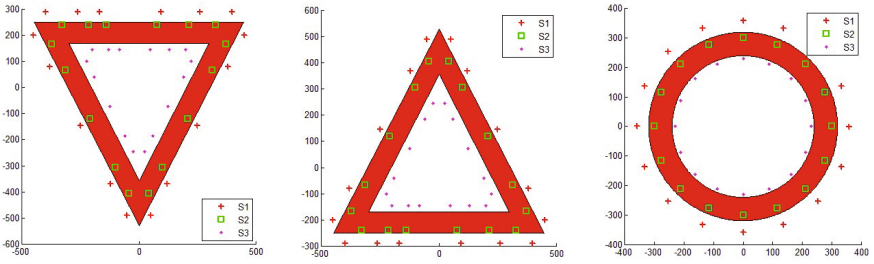


Fig. 1. The model used to recognize priority, warning and prohibitory signs. Each model consists of three sets of points. The first (S_1) lies just outside the sign; the second (S_2) on the red band, while the third (S_3) inside the sign.

In [11], PSO generates location estimates for a sign as each particle in the swarm encodes the candidate sign pose as four values: its offsets along the x, y and z axes, and its rotation around the vertical axis (yaw) in the camera reference frame. Rotation around the camera optic axis (roll) and the horizontal axis (pitch) have been ignored after some preliminary tests showed that they have little relevance. The image region located by the model is then rectified via an inverse perspective transform in order to obtain a frontal view. Then, the three sets of points (see Figure 1) are evaluated according to the fitness function described below; a sign is considered to have been detected when the fitness value is below a fixed threshold.

For every set of points, three color histograms in the HSV color space are computed. Then, the Bhattacharyya coefficient [6] $B(x, y)$, which estimates the overlap between two statistical samples, is used to compare the histograms. The fitness function can be expressed as follows:

$$f = \frac{k_0(1 - B(h_1, h_2)) + k_1(1 - B(h_2, h_3)) + k_2 B(h_1, h_r)}{k_0 + k_1 + k_2}$$

where h_i is the histogram of set S_i, and h_r is a reference histogram centered on red; k_0, k_1 and $k_2 \in \Re^+$ are used to weigh the elements of the equation. This means that a sign is detected when:

- the histogram of the set outside the sign is different from the histogram of the set located on the red band;
- the histogram of the points in the red band is as different as possible from the one computed on the inner area of the sign;
- the histogram of the points in the red band is similar to the reference histogram we defined for the red hue (a Gaussian centered on pure red).

This operation is repeated twice for every frame to permit recognition of more than one sign of the same category. This method has shown good robustness against illumination changes.

Mandatory signs (see figure 2) are detected similarly to the other three categories, although some changes have been introduced to improve performance.

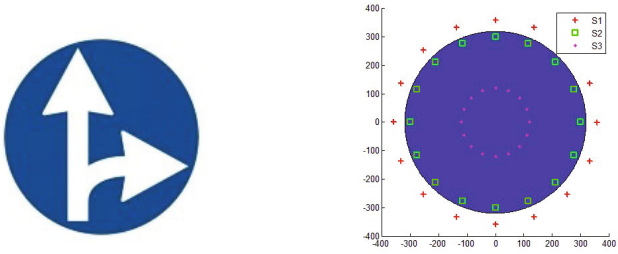


Fig. 2. An example of a mandatory sign, along with the corresponding model

We still consider three similarly arranged sets of points: the only difference in their location is that set S_3 is located closer to the center of the sign, in order to get more information about the white foreground, which usually includes the main information useful for sign recognition. The fitness function has also been changed to:

$$f = \frac{\bar{k}_0(1 - B(h_1, h_2)) + \bar{k}_1(B(h_3, h'_r)) + \bar{k}_2 B(h_2, h''_r) + \bar{k}_3(1 - B(h_2, h_r))}{\bar{k}_0 + \bar{k}_1 + \bar{k}_2 + \bar{k}_3}$$

where h'_r is a reference histogram with peaks corresponding to blue and white, h''_r a reference histogram centered on blue, and $\bar{k}_i$, $i = 1, \ldots, 4$ are positive weights.

2.2 Sign Classification

Before performing classification, the rectified image of the detected sign is pre-processed to reduce the complexity of this step. The following pre-processing steps are performed:

- the image is re-sampled to 50×50 pixels, converted to gray scale, and the pixels outside the sign are removed;
- for mandatory signs, in which the foreground is brighter than the background, gray-scale values are inverted, in order for the inputs to the final classifier to have similar contrast features;
- the histogram is calculated, average-filtered to remove isolated peaks and stretched between the mean intensities of the sign background and foreground.

In our test set we considered 27 different classes for prohibitory, 30 classes for warning, and 27 for mandatory signs. The training sets contained 9 instances of each sign, obtained by applying the pre-processing phase over synthetic noiseless images of the sign and adding some noise in terms of position and rotation. Since there is only one possible instance of priority signs, we implemented a binary classifier and created a test set which includes priority signs along with signs of different categories and other elements that are not signs (cars, trees, guard rails . . .).

3 Implementation Details

Both the detection and classification phases have been implemented on a Graphical Processing Unit (GPU) within the CUDA (Compute Unified Distributed Architecture) environment available from nVIDIA. CUDA requires that a problem be divided into groups of cooperating threads (each group is called a *thread block*), organized in one-, two- or three-dimensional grids. The threads within a block are also organized in similar grids. The performance of CUDA code largely depends on the grid configurations and memory access schemes. As in CPUs, also in GPUs, memory is arranged in a hierarchy, in which each thread has its own private local memory, thread blocks use shared memory that is visible only to the threads of the block, while all threads have access to global device, texture, and constant memory spaces [14]. To maximize efficiency, algorithms developed in CUDA-C should rely mainly on fast local and shared memory, avoiding frequent accesses to global memory locations.

The details about the parallel design and implementation of the methods used for detection and classification are discussed in the following subsections.

3.1 Parallel Particle Swarm Optimization

Our parallel version of PSO [11] is divided into three kernels: position update, fitness evaluation, and bests update. Two-dimensional thread-block grids represent different swarms (one for each type of sign) along one dimension while, along the other, every block represents a particle. Position update and fitness evaluation are performed by the threads of a block working on particle data loaded in shared memory for faster performance. Kernels follow a common procedure that: i) loads particle data into shared memory from global device memory, ii) processes the data in local and shared memory, iii) stores the results back to global memory, at the end of its execution, to make them visible to other kernels. We use four swarms (one for each sign category), each consisting of 64 particles arranged in a ring topology, that run for 250 generations. As for PSO parameters, we set C_1 and C_2 to 1.19, and the inertia factor w to 0.72.

3.2 Parallel Differential Evolution

Differential Evolution (DE) [18] is a powerful stochastic real-parameter optimization algorithm [1]. New solutions are generated by performing a crossover operation between one element of the current population (*parent*) and a *donor*, which is created by the combination of some randomly chosen solutions from the population. This new element, called *trial*, is then evaluated and can replace the parent if it has a better fitness than the parent's. The parallel implementation of DE [12] resembles parallel PSO, except that, instead of the position update kernel, DE uses a kernel to generate trial vectors for every element in the solution group, and another kernel to evaluate the fitness of the trial solutions to possibly replace the parent with them. There are several flavors of DE, depending on the method for choosing the individuals that are combined to form donor solutions,

and the type of crossover used to generate the trial solutions. In our experiments, we adopt random mutation and binomial crossover. The same PSO swarm configuration was used for the DE population, also run for 250 generations, with DE parameters: F set to 0.5 and C_r to 0.9.

3.3 Parallel Multi-Layer Perceptron

The Multi-layer Perceptron (MLP) is the most commonly used artificial neural network [4], in which only feedforward connections between neurons are allowed and a supervised training algorithm (backpropagation) is used.

The number of operations needed to perform the classification using MLP is very high. For instance, the MLP we use for the classification of warning signs has four fully-connected layers whose sizes are respectively 2500, 180, 90, 30. This means that, for the first layer, the total number of products that has to be computed is 2500×180, as each of the 2500 inputs is multiplied by the weight of the connections between it and all neurons of the next layer, and must then be summed. These operations amount to a product between a 2500-elements vector and a 180×2500 matrix. In our parallel implementation the computation of each layer of the network is a CUDA kernel. The operations performed by each kernel are:

1. loading layer inputs in shared memory;
2. computing, partly in parallel and partly sequentially, the products (as suggested by [3]);
3. summing the products by means of a parallel reduction;
4. computing the activation function over the sum of the products.

3.4 Parallel Learning Vector Quantization

Learning Vector Quantization (LVQ) is a supervised classification algorithm developed by Kohonen [8]. The basic idea is to find a small set (called *codebook*) of prototypes (called *codebook vectors*) representative of all possible inputs. New data are then classified according to the most similar codebook vector. The training phase is performed by iterating over the examples in a training set: if an example is correctly classified, the codebook vector closest to the data sample is attracted towards it, otherwise the codebook vector is moved in the opposite direction. The CUDA implementation of LVQ is straightforward: a kernel computes the distance between the input vector and the codebook vectors that compose the network. Then, a parallel reduction is performed to compute the classification result, that is the label corresponding to the most similar codebook vector. Table 1 shows the topologies and sizes of the networks that have obtained the best results during our experiments.

4 Experimental Results

In this section we will describe the data and the experiments performed to evaluate our system, for both detection and classification.

Table 1. MLP and LVQ structures for the four categories of signs

Category	MLP Topology	LVQ size
Prohibitory	2500×180×90×27	176
Warning	2500×180×90×30	246
Mandatory	2500×100×50×27	128
Priority	2500×120×16×2	128

4.1 Detection

The benchmark used to evaluate the results in a real environment is composed of two sequences [10]. The first one, which includes 10000 frames at a resolution of 750×480 pixels, was acquired at 7.5 fps in Parma on a sunny day. The sequence contains images featuring all possible light orientations. The second sequence is about 5000 frames long and was acquired at 7.5 fps in Turin on a cloudy day. Images in this sequence feature more constant lighting but lower contrast.

We compared the results of DE and PSO in terms of correct/incorrect detections of the signs. DE was able to detect more signs (scoring more true and false positives); this suggests that DE has a greater exploitation ability, and is able to refine solutions better than PSO. However, if fitness values are compared, PSO has a better average, with a lower standard deviation, showing a more consistent behavior. Table 2 reports the best and worst detection results obtained over 10 runs on each sequence.

4.2 Classification

The evaluation of the classification system was firstly performed over a test set that comprises synthetic, good quality and noisy or deformed images of signs, as described in [10], then on the two real sequences. The first rows of table 3 show the percentage of correct classification for the four categories on the test set. The same table (second and third row) shows the results of the classification of all signs detected in the experiments reported in table 2. We take all detections into account: this means that the same sign can be detected and classified in more than one frame.

Table 2. Results of the detection phase (min-max) for the four categories of signs (detections): worst and best result in 10 independent runs

		Parma Sequence			Turin Sequence		
		Total	False Positives	Detections	Total	False Positives	Detections
Warning	DE	51	0-1	27-31	53	0-2	39-43
	PSO	51	0-0	27-30	53	0-1	35-40
Prohibitory	DE	44	2-6	26-30	47	2-4	39-42
	PSO	44	0-1	22-27	47	0-1	39-40
Priority	DE	30	5-11	18-22	15	2-4	13-15
	PSO	30	0-2	15-18	15	0-1	7-12
Mandatory	DE	62	2-4	40-41	39	0-1	27-29
	PSO	62	0-1	35-39	39	0-1	24-27

Table 3. Percentage of correct sign classifications in the test set and in the two sequences for the four categories of signs

Set		Warning	Prohibitory	Priority	Mandatory	Total
Test Set	LVQ	74.3	78.7	98.7	89.3	81.6
	MLP	71.5	80.5	99.4	94.3	83.1
Parma Sequence	LVQ	91.2	81.9	97.0	99.4	92.4
	MLP	69.0	69.2	95.4	99.4	83.2
Turin Sequence	LVQ	75.3	94.8	100	100	92.5
	MLP	77.1	77.8	98.9	99.6	88.4

The table shows that the two methods have similar performance (with slightly better performances for the MLP) on the test set, but LVQ achieves better results over the real sequences. This probably happens because the test set contains several deformed or noisy images, while the images produced by the detection are usually good in terms of quality and positioning. The conclusion can be that LVQ is able to yield better results when operating on good quality images, while MLPs have better generalization ability. Finally, in table 4, we present the results of the entire system using DE and LVQ. Since, in the actual implementation, there is no way to track signs during the flow of the sequence (and, consequently, each sign can be detected multiple times), we consider a sign to have been correctly classified by evaluating off-line if at least half of its classifications are correct. Figure 3 shows some examples of correct classifications, correct detections with wrong classification and wrong detections.

4.3 Computational Efficiency

Experiments were run on a PC equipped with a 64-bit Intel® Core(TM) i7 CPU running at 2.67 GHz, combined with a Quadro FX5800 graphics card by nVIDIA, having 4Gb of video RAM and 240 processing cores. All the operations performed on each frame (two repetitions of detection plus classification for each sign category) require an average time of 57 ms, which corresponds to a frame rate of 17.5 fps. A sequential version of the same algorithm could reach only a frame rate of 4-5 fps [11], a processing speed which is not acceptable for this kind of application.

Table 4. Final results of the system using DE and LVQ

	Parma Sequence		Turin Sequence	
	Total	Correct	Total	Correct
Warning	51	25-28	53	37-43
Prohibitory	44	24-27	47	38-41
Priority	30	17-20	15	11-15
Mandatory	62	38-41	39	26-29

Fig. 3. Some results of the system. The first three columns show signs that have been correctly detected and classified, the fourth shows misclassified signs, while the last one shows wrong detections. Our method is robust against differences in light conditions and partial occlusions.

5 Conclusions

We presented a system for detecting and classifying road signs. Detection is performed using a method in which a model of the sign to detect is translated and projected onto the image. Candidate solutions are created by means of swarm intelligence techniques. Differential Evolution and Particle Swarm Optimization have been compared in this phase, showing that PSO has better average results than DE, while DE exhibits a better exploitation ability, which produces a larger number of detections. Classification have been performed using Learning Vector Quantization and Multi-layer Perceptrons. The results showed that LVQ has better performance when working on good quality images, while MLPs have greater generalization ability. The system has been implemented on GPU using CUDA and is able to correctly detect and classify around 70% of the signs at 17.5 fps, a similar result in shorter time, compared to the best results obtained on the same sequences so far [10].

Acknowledgments. Youssef S. G. Nashed is funded by the European Commission (MIBISOC Marie Curie Initial Training Network, FP7 PEOPLE-ITN-2008, GA n. 238819). Roberto Ugolotti is funded by Compagnia di San Paolo and Fondazione Cariparma. The authors wish to thank Luca Donati for his help in the development of the system and his support regarding the library used for the training of MLP[1], and VisLab for providing the sequences.

[1] Libcudann is freely available at `http://sourceforge.net/projects/libcudann`

References

1. Das, S., Suganthan, P.: Differential Evolution: A survey of the state-of-the-art. IEEE Transactions on Evolutionary Computation 15(1), 4–31 (2011)
2. Escalera, S., Baró, X., Pujol, O., Vitrià, J., Radeva, P.: Background on traffic sign detection and recognition. In: Traffic-Sign Recognition Systems. SpringerBriefs in Computer Science, pp. 5–13. Springer, London (2011)
3. Harris, M.: Optimizing parallel reduction in CUDA. NVIDIA Developer Technology (2008)
4. Haykin, S.: Neural Networks: a comprehensive foundation. Prentice Hall (1999)
5. Jiang, Y., Zhou, S., Jiang, Y., Gong, J., Xiong, G., Chen, H.: Traffic sign recognition using ridge regression and Otsu method. In: IEEE Intelligent Vehicles Symposium (IV), pp. 613–618 (2011)
6. Kailath, T.: The divergence and Bhattacharyya distance measures in signal selection. IEEE Transactions on Communication Technology 15(1), 52–60 (1967)
7. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
8. Kohonen, T.: Learning Vector Quantization. Neural Networks 1(supplement 1) (1988)
9. Maldonado-Bascon, S., Lafuente-Arroyo, S., Gil-Jimenez, P., Gomez-Moreno, H., Lopez-Ferreras, F.: Road-sign detection and recognition based on support vector machines. IEEE Trans. on Intelligent Transportation Systems 8(2), 264–278 (2007)
10. Medici, P., Caraffi, C., Cardarelli, E., Porta, P., Ghisio, G.: Real time road signs classification. In: IEEE International Conference on Vehicular Electronics and Safety, ICVES 2008, pp. 253–258 (2008)
11. Mussi, L., Cagnoni, S., Cardarelli, E., Daolio, F., Medici, P., Porta, P.: GPU implementation of a road sign detector based on Particle Swarm Optimization. Evolutionary Intelligence 3(3), 155–169 (2010)
12. Nashed, Y.S., Ugolotti, R., Mesejo, P., Cagnoni, S.: libCudaOptimize: an open source library of GPU-based metaheuristics. In: Proc. Genetic and Evolutionary Computation Conference, GECCO 2012 (2012)
13. Nguwi, Y.Y., Kouzani, A.: Detection and classification of road signs in natural environments. Neural Computing & Applications 17(3), 265–289 (2008)
14. nVIDIA Corporation: nVIDIA CUDA programming guide v. 4.0 (May 2011)
15. Ohara, H., Nishikawa, I., Miki, S., Yabuki, N.: Detection and recognition of road signs using simple layered neural networks. In: Proceedings of the 9th International Conference on Neural Information Processing, ICONIP 2002, vol. 2, pp. 626–630 (2002)
16. Paulo, C., Correia, P.: Automatic detection and classification of traffic signs. In: Workshop on Image Analysis for Multimedia Interactive Services (2007)
17. Prieto, M.S., Allen, A.R.: Using self-organising maps in the detection and recognition of road signs. Image and Vision Computing 27(6), 673–683 (2009)
18. Storn, R., Price, K.: Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute (1995)

Acceleration of Evolutionary Image Filter Design Using Coevolution in Cartesian GP

Michaela Sikulova and Lukas 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. The aim of this work is to accelerate the task of evolutionary image filter design using coevolution of candidate filters and training vectors subsets. Two coevolutionary methods are implemented and compared for this task in the framework of Cartesian Genetic Programming (CGP). Experimental results show that only 15–20% of original training vectors are needed to find an image filter which provides the same quality of filtering as the best filter evolved using the standard CGP which utilizes the whole training set. Moreover, the median time of evolution was reduced 2.99 times in comparison with the standard CGP.

1 Introduction

Evolutionary design based on genetic programming is a very computationally-intensive design method. For example, for 36 tasks solved using Koza's genetic programming, the average population size was 3,350,000 individuals, 128.7 generations were produced in average and the average time to reaching a solution was 81.9 hours [3]. It also holds for the evolutionary design of image filters which has been performed by Cartesian Genetic Programming (CGP). The most time consuming procedure is the fitness calculation where tens of thousands of pixels in training set (the so-called target objective vectors, TOVs) have to be evaluated in order to obtain a single fitness value. A single run is typically finished after 200 thousands candidate filter evaluations. However, for the cost of runtime, very efficient image filters were evolved, often beating conventional designs in terms of the filtering quality as well as the area required on a chip [9]. In order to reduce the time of evolution, various CGP accelerators have been introduced including GPU-based and FPGA-based machines [12, 11, 1].

In this paper, we propose to employ a coevolutionary algorithm running on an ordinary processor to accelerate the image filter evolution. Various coevolutionary methods have been introduced that can evolve suitable subsets of TOVs for evaluation of candidate solutions. The aim of this type of coevolution is to allow both candidate programs and TOVs subsets to improve each other automatically until a satisfactory problem solution is found. Coevolutionary algorithms with interactions between two independently evolving populations, in the "hosts" and "parasites" type relationships, were studied in many application domains [2, 7, 6, 4].

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 163–172, 2012.

In our previous work, inspired by *coevolution of fitness predictors* (CFP) [8], we applied coevolution of TOVs in CGP in order to accelerate the task of symbolic regression [10]. In this paper, we will show that the subsets of TOVs can be (substantially) smaller than original training sets in the task of image filter evolution. Consequently, the overall time of filter evolution can be significantly reduced. This paper also proposes and evaluates two strategies for top-ranked TOVs subset selection. The proposed coevolutionary algorithms will be compared with the standard CGP in the task of evolutionary design of image filters suppressing a salt-and-pepper noise.

2 CGP for Image Filter Design

The state of the art of CGP has recently been summarized in a monograph [5] which also surveys the evolutionary image filter design using CGP [9]. In CGP, candidate programs are represented in the form of directed acyclic graph, which is modeled as a matrix of $n_c \times n_r$ programmable elements (nodes). The number of primary inputs, n_i, and outputs, n_o, of the program is defined for a particular task. Each node input can be connected either to the output of a node placed in previous l columns or to one of the program inputs. 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.

As the considered filters operate over a filtering window consisting of 3×3 pixels, each candidate filter can utilize up to nine 8-bit inputs, i.e. $n_i = 9$. The filters produce a single pixel, i.e. $n_o = 1$. Table 1 gives a set of functions working over two pixels i_1 and i_2 that are typically used for image filter evolution. Figure 1 shows an example of a candidate filter and its encoding in CGP.

In this task, the search is usually performed using a simple $(1+\lambda)$ evolutionary algorithm, where $\lambda = 7$. Every new population consists of the best individual of the previous population and its λ offspring created using a mutation operator which modifies up to h integers of the chromosome. The initial population is generated randomly. The algorithm is terminated when the maximum number

Table 1. List of node functions

Code	Function	Description	Code	Function	Description
0	255	constant	8	$i_1 \gg 1$	right shift by 1
1	i_1	identity	9	$i_1 \gg 2$	right shift by 2
2	$255 - i_1$	inversion	A	$swap\,(i_1, i_2)$	swap nibbles
3	$i_1 \vee i_2$	bitwise OR	B	$i_1 + i_2$	+ (addition)
4	$\overline{i_1} \vee i_2$	bitwise $\overline{i_1}$ OR i_2	C	$\overline{i_1} +^S i_2$	+ with saturation
5	$i_1 \wedge i_2$	bitwise AND	D	$(i_1 + i_2) \gg 1$	average
6	$\overline{i_1 \wedge i_2}$	bitwise NAND	E	$max\,(i_1, i_2)$	maximum
7	$i_1 \oplus i_2$	bitwise XOR	F	$min\,(i_1, i_2)$	minimum

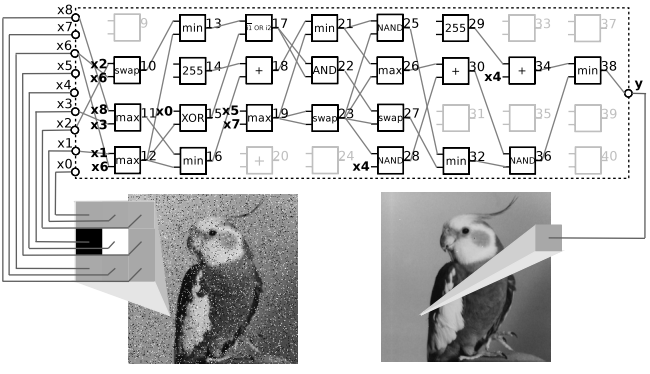


Fig. 1. A candidate filter in CGP, where $l = 1$, $n_c = 8$, $n_r = 4$, $n_i = 9$, $n_o = 1$, $n_a = 2$, Γ is according to Table 1 and the chromosome is: $6, 7, 14;\ 2, 6, 10;\ 8, 3, 14;\ 1, 6, 14;\ 10, 12, 15;\ 12, 8, 0;\ 0, 12, 7;\ 11, 12, 15;\ 13, 15, 4;\ 14, 16, 11;\ 5, 7, 14;\ 16, 5, 10;\ 18, 19, 15;\ 17, 17, 5;\ 19, 19, 10;\ 19, 18, 7;\ 21, 23, 6;\ 21, 23, 14;\ 22, 23, 10;\ 4, 23, 6;\ 25, 28, 0;\ 28, 26, 11;\ 1, 25, 11;\ 27, 25, 15;\ 31, 30, 9;\ 4, 29, 11;\ 6, 30, 8;\ 30, 32, 6;\ 33, 33, 2;\ 34, 36, 15;\ 33, 35, 11;\ 33, 34, 1;\ 38.$

of generations is exhausted, typically after 30,000 generations [9]. In the fitness function, the goal is to minimize the mean absolute error between uncorrupted version of the training image and the result of filtering of a candidate filter. This can be expressed in terms of the mean difference per pixel (MDPP) as

$$\mathrm{MDPP} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |v\,(i,j) - w\,(i,j)|\,. \tag{1}$$

where $M \times N$ is the image size, $v(i,j)$ is a pixel value in the filtered image and $w(i,j)$ is a pixel value in the uncorrupted image.

3 Coevolution of TOVs in CGP

In the proposed coevolutionary algorithm, there are two concurrently working populations: (1) candidate programs (filters) evolving using CGP and (2) TOVs subsets evolving using a genetic algorithm. Figure 2 shows that both populations evolve simultaneously, interacting through the fitness function (using top-ranked individuals).

3.1 Population of Candidate Filters

Evolution of candidate filters is based on principles of CGP as introduced in Section 2. The fitness function for CGP is defined in terms of MDPP. There are, in fact, two fitness functions for a candidate filter. While the exact fitness function $\mathrm{MDPP}_{\mathrm{exact}}$ utilizes the complete training set (i.e. all training vectors from the training images), the partial fitness function $\mathrm{MDPP}_{\mathrm{partial}}$ uses only a selected subset. Formally,

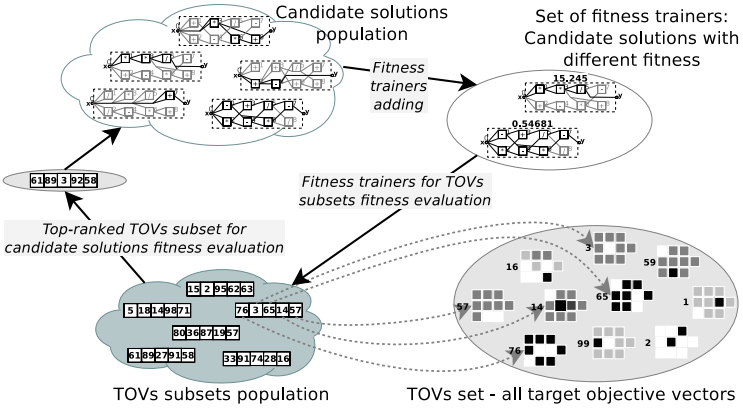


Fig. 2. Populations in coevolution of TOVs in CGP

$$\mathrm{MDPP}_{\mathrm{exact}} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |v\ \ (i,j) - w\,(i,j)| \tag{2}$$

$$\mathrm{MDPP}_{\mathrm{partial}} = \frac{1}{K} \sum_{l=1}^{K} |v\,(l) - w\,(l)| \tag{3}$$

where $M \times N$ is the total number of TOVs in the training set and K is the number of TOVs in a training subset and l is index in the list of pointers to pixel at possition (i, j).

The set of *fitness trainers* contains several candidate filters and is used to evaluate the fitness of TOVs subsets. If the top-ranked candidate filter has a different fitness value than the top-ranked candidate filter in the previous generation, the top-ranked candidate filter replaces the oldest trainer in a circular list of trainers.

3.2 Population of TOVs Subsets

The most useful subset of TOVs is sought using a simple genetic algorithm (GA) which operates with a population of TOVs subsets. Every TOVs subset is encoded as a constant-size array of pointers to elements (i, j) in the training set. In addition to one-point crossover and mutation, a randomly generated TOVs subset replacing the worst-scored TOVs subset in each generation has been introduced as a new genetic operator of GA.

We will compare two approaches to fitness calculation. In the first one, the fitness value of a TOVs subset (i.e. the fitness predictor) is calculated using the mean absolute error of the exact fitness and partial fitness of fitness trainers s. This fitness value, which is based on the CFP approach [8], can be expressed as

$$\mathrm{f}_{\mathrm{CFP}} = \frac{1}{T} \sum_{t=1}^{T} |\mathrm{MDPP}_{\mathrm{partial}}\,(s(t)) - \mathrm{MDPP}_{\mathrm{exact}}\,(s(t))|\,, \tag{4}$$

where T is the number of trainers in the trainers set. The goal of TOVs subsets evolution is to minimize this f_{CFP} value, i.e. to ensure that the TOVs subset can determine (predict) the solutions' fitness values as exactly as possible.

Another approach exploits the *competitive coevolution* (CC) scheme [2]. The population of candidate filters can be viewed as "hosts" and the population of test cases as "parasites". The fitness of each candidate filter is measured by its ability to correctly solve training cases (therefore, the eq. 3 is applicable) while the fitness of the training cases is higher for those that cannot be solved well by currently evolved filters. Then the fitness value of a TOVs subset derived from Eq. 3 is defined as

$$f_{CC} = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{K} \sum_{l=1}^{K} |v(l) - w(l)| \tag{5}$$

and the goal of evolution is to maximize the f_{CC} value. This type of fitness function should ensure that the TOVs subset includes TOVs that cannot be solved exactly by currently evolved filters.

3.3 Implementation

There are two threads in the coevolution implementation. One thread is responsible for candidate filters evolution using CGP, the other one for TOVs subsets evolution. This two thread model is described in Figure 3.

In the first step, the candidate filters, the trainers and the TOVs subsets are randomly initialized. Then the CGP thread waits for the first evolved TOVs subset to load it from shared memory, which is done at the beginning of every iteration of the CGP main loop. This loop continues with evaluating fitness values ($MDPP_{partial}$) of each candidate filter in a current population and selecting the top-ranked candidate filter. Next, there is a possibility of storing a new trainer in trainers circular list. After deciding whether to store the new trainer or not, a new generation is created and the evolution loop continues with the next iteration. The evolution loop terminates when the predefined count of generations is reached.

The TOVs subsets evolution loop begins with loading the trainers from shared memory. Depending on the selected method the exact fitness is or is not evaluated. In CFP, the exact fitness $MDPP_{exact}$ is calculated for each trainer. Then the TOVs subset fitness is evaluated using f_{CFP} (Eq. 4). In CC, the exact fitness values of trainers are not evaluated, and the TOVs subset fitness is calculated using f_{CC} (Eq. 5). As the top-ranked TOVs subset is then taken the subset with the maximal f_{CC} value, which means that this TOVs subset filtered using trainers has the worst (maximal) mean quality measured by $MDPP_{predicted}$ over trainers. This can help the candidate filters to improve filtering those training cases, which cannot be solved yet.

```
BEGIN Filters Thread
    Randomize trainers-circular-list
    Randomize filters population
    FOR 1 TO generation-total-count DO BEGIN
        Load top-ranked-TOVs-subset from shared memory
        Evaluate partial fitnesses for filters using top-ranked-TOVs-subset
        Select top-ranked-filter
        IF actual-parent-fitness <> previous-parent-fitness THEN BEGIN
            Store parent to trainers-circular-list to shared memory
        END
        Create new generation of filters using top-ranked-filter
    END
    Set terminating-flag
    Evaluate exact fitness of last-top-ranked-filter
    RETURN last-top-ranked-filter
END

BEGIN TOVs Thread
    Randomize TOVs population
    REPEAT forever
        Load trainers-circular-list from shared memory
        [OPTIONALLY] Evaluate exact fitnesses of trainers     // depending on selected method
        Evaluate fitnesses for TOVs-subsets using trainers
        Select top-ranked-TOVs-subset
        Store top-ranked-TOVs-subset to shared memory
        Create new generation of TOVs-subsets using 2-tournament selection
                                              and single point crossover
        IF terminating-flag THEN BEGIN
            EXIT thread
        END
    END
END
```

Fig. 3. Pseudocode for coevolution of the population of filters and the TOVs subsets population

After trainers processing, the fitness values of TOVs subsets are evaluated and the top-ranked TOVs subset is selected and stored to shared memory. The worst-ranked TOVs subset is replaced by a new randomly generated one, which is involved in TOVs subset reproduction to ensure TOVs subsets population diversity. A new generation is then created and the evolution loop continues with the next iteration, while the terminating flag is not set up.

4 Results

This section presents benchmark problems, experimental setup and experimental evaluation of the proposed coevolutionary approach and its comparison with the standard CGP.

4.1 Benchmark Problems

In order to evaluate the proposed approach, salt and pepper noise filters will be designed using CGP. This type of noise is characterized by noisy pixels with the value of either 0 or 255 (for the 8-bit gray-scaled images) typically caused by

errors in data transmission, faulty memory locations in hardware or malfunctioning pixels in camera sensors. Conventionally designed filters for this type of noise are based on the median function. Two noise intensities have been applied, i.e. the Lena training image with resolution 256×256 pixels was corrupted by 5 % and 10 % salt-and-pepper type of noise. Evolved filters are tested using five different images containing the same type of noise.

4.2 Experimental Setup

CGP is used according to literature [5], i.e. $n_c = 8$, $n_r = 4$, $l = 1$, $n_i = 9$, $n_o = 1$, $\lambda = 7$, every node has two inputs, the number of mutations per new individual is $h = 5$ and Γ contains the functions from Table 1. The trainers set is represented as a circular list with eigth elements.

TOVs subsets are evolved using a simple GA, where 2-tournament selection, single point crossover (with the probability 100%, but the top-ranked individual is always involved in next generation) and mutation up to 2% of chromosome are used. For the GA, various chromosome lengths are tested, particularly, 2.5 %, 5 %, 10 %, 15 %, 20 % and 25 % of total size of TOVs in the training set (which contains 64,516 TOVs because boundary pixels are not considered in the 256×256 training images). For each TOVs subset size, 25 independent runs were performed and the evolution/coevolution was terminated after 30,000 generations of CGP.

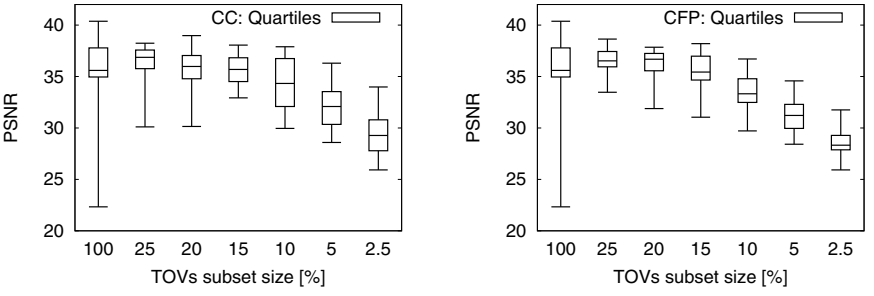
4.3 Comparison of Coevolving CGP with Standard CGP

The proposed coevolutionary algorithms were compared with the standard CGP in terms of filtering quality of evolved filters and the execution time.

The quality of filtering is expressed as a peak signal-to-noise ratio (PSNR) which is a measure typically used in the image processing community. It can be seen in Figure 4 that the proposed coevolutionary algorithm is capable of evolving image filters of satisfactory quality even if only 15% of TOVs are utilized.

Table 2 gives PSNR for five test images filtered by the best filters evolved using standard CGP, coevolutionary CGP (CC, the TOVs subset size is 20 %) and conventional median filter. The PNSR results of coevolutionary CGP are comparable or better with respect to the standard CGP for both noise intensities. The PSNR results for the median filter are not as good, however, we expected this on the basis of our previous work [9]. The best filter for the 5% noise evolved using CGP with coevolution is shown in Figure 1.

Figure 5 shows one of test images corrupted with the 5 % salt-and-pepper noise and then filtered using the median filter, the best filter evolved using the standard CGP and the best filter evolved using CGP with coevolution. The median filter provides smudged images in comparison to evolved filters.



(a) CC PSNR **(b)** CFP PSNR

Fig. 4. PSNR statistics calculated from 25 independent runs of CC and CFP for the Lena image (5% noise). The 100 % result is for the standard CGP.

It can be seen in Figure 6 that for the 15% TOVs subset size, the evolutionary design is accelerated 2.99-times in comparison to the standard CGP. Note that the execution time is given as the sum of execution times of both threads. The speedup was measured on the 2× 8-thread Intel® Xeon® E5640 machine.

Figures 6 and 4 show that the CC and CFP fitness interacting strategies are similar in terms of filtering quality of evolved filters and execution time. However, profiles of evolved TOVs subsets differ. The CFP fitness interacting strategy leads to TOVs subsets that have a similar ratio of corrupted pixels as the total TOVs set, while the CC fitness interacting strategy leads to a little higher ratio of corrupted pixels in TOVs subsets (Table 3).

Table 2. PSNR for test images filtered by the best filters evolved using standard CGP, coevolutionary CGP (CC, the TOVs subset size is 20 %) and conventional median filter.

Test image	5 % noise			10 % noise		
	std CGP	coevolution	median 3×3	std CGP	coevolution	median 3×3
Airplane	38.008	37.747	29.303	32.370	34.053	28.557
Bird	46.113	44.706	38.242	35.735	40.054	36.990
Bridge	35.117	33.707	26.040	30.051	30.246	25.662
Camera	35.299	36.075	26.823	30.590	32.800	26.245
Goldhill	37.799	37.906	27.927	32.284	34.012	27.524
Lena	38.233	38.357	30.381	33.332	35.301	29.739

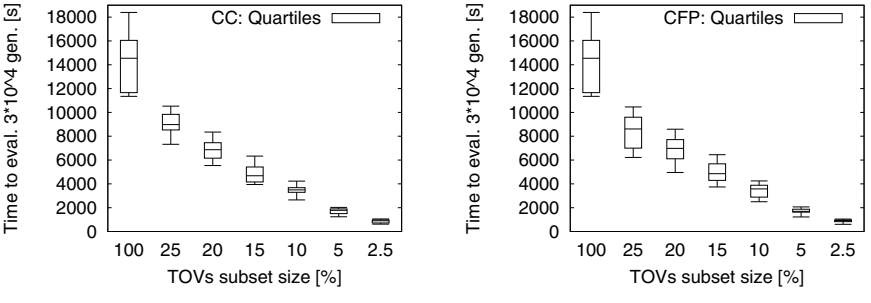
Table 3. Comparison of a mean ratio of corrupted pixels in the top-ranked TOVs subsets in the CFP and CC strategy (5 % noise).

Subset size	25 %	20 %	15 %	10 %	5 %	2.5 %
CFP	4.973 %	5.009 %	4.971 %	4.960 %	4.961 %	5.087 %
CC	5.328 %	5.391 %	5.456 %	5.658 %	5.519 %	6.114 %

(a) Noise 5 % **(b)** Median filter

(c) Std CGP filter **(d)** Coe CGP filter

Fig. 5. Comparison of images filtered by median filter (b), the best filter evolved using the standard CGP (c), and CGP with coevolution (d)



(a) CC speed **(b)** CFP speed

Fig. 6. Execution time statistics calculated from 25 independent runs for CC and CFP. The 100% result is for the standard CGP running in one thread. Other values are for coevolution using two threads (the execution time is the sum of both threads).

5 Conclusions

In this paper, we proposed two coevolutionary methods to CGP in order to accelerate the evolutionary design of image filters. No significant differences were observed between the cooperative coevolution and coevolution of fitness predictors. It was shown that only 15–20% of original test vectors are needed to find

an image filter which provides the same quality of filtering as the best filter evolved using the standard CGP which utilizes the whole training set. The median time of evolution was reduced 2.99 times in comparison with the standard CGP. Future work will be devoted to further parallelization of the whole concept.

Acknowledgments. This work was supported by the Czech science foundation projects P103/10/1517 and GD102/09/H042, 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] Harding, S.L., Banzhaf, W.: Hardware acceleration for cgp: Graphics processing units. In: Cartesian Genetic Programming, pp. 231–253. Springer (2011)
[2] Hillis, D.W.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D: Nonlinear Phenomena 42(1–3), 228–234 (1990)
[3] Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers (2003)
[4] Lohn, J., Kraus, W., Haith, G.: Comparing a coevolutionary genetic algorithm for multiobjective optimization. In: Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002, vol. 2, pp. 1157–1162 (2002)
[5] Miller, J.F.: Cartesian Genetic Programming. Springer (2011)
[6] Pagie, L., Hogeweg, P.: Evolutionary consequences of coevolving targets. Evolutionary Computation 5(4), 401–418 (1997)
[7] Rosin, C.D., Bellew, R.K.: New methods for competitive coevolution. Tech. Rep. CS96-491, Department of Computer Science and Engineering, University of California, San Diego (1996)
[8] Schmidt, M.D., Lipson, H.: Coevolution of Fitness Predictors. IEEE Transactions on Evolutionary Computation 12(6), 736–749 (2008)
[9] Sekanina, L., Harding, L.S., Banzhaf, W., Kowaliw, T.: Image processing and cgp. In: Cartesian Genetic Programming, pp. 181–215. Springer (2011)
[10] Šikulová, M., Sekanina, L.: Coevolution in Cartesian Genetic Programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 182–193. Springer, Heidelberg (2012)
[11] Vasicek, Z., Sekanina, L.: Hardware accelerator of cartesian genetic programming with multiple fitness units. Computing and Informatics 29(6), 1359–1371 (2010)
[12] Wang, J., Chen, Q.S., Lee, C.H.: Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware. IET Computers and Digital Techniques 2(5), 386–400 (2008)

Transfer Learning, Soft Distance-Based Bias, and the Hierarchical BOA

Martin Pelikan[1], Mark W. Hauschild[1], and Pier Luca Lanzi[2]

[1] Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), Department of Mathematics and Computer Science, University of Missouri, St. Louis, MO 63121
martin@martinpelikan.net, mwh308@umsl.edu, http://medal-lab.org/

[2] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci, 32, I-20133 Milano, Italy
pierluca.lanzi@polimi.it, http://www.pierlucalanzi.net/

Abstract. An automated technique has recently been proposed to transfer learning in the hierarchical Bayesian optimization algorithm (hBOA) based on distance-based statistics. The technique enables practitioners to improve hBOA efficiency by collecting statistics from probabilistic models obtained in previous hBOA runs and using the obtained statistics to bias future hBOA runs on similar problems. The purpose of this paper is threefold: (1) test the technique on several classes of NP-complete problems, including MAXSAT, spin glasses and minimum vertex cover; (2) demonstrate that the technique is effective even when previous runs were done on problems of different size; (3) provide empirical evidence that combining transfer learning with other efficiency enhancement techniques can often yield nearly multiplicative speedups.

Keywords: Transfer learning, inductive transfer, learning from experience, estimation of distribution algorithms, hierarchical Bayesian optimization algorithm, decomposable problems, efficiency enhancement.

1 Introduction

Estimation of distribution algorithms (EDAs) [1,2,3] guide the search for the optimum by building and sampling probabilistic models of candidate solutions. The use of probabilistic models in EDAs provides a basis for incorporating prior knowledge about the problem and learning from previous runs in order to solve new problem instances of similar type with increased speed, accuracy and reliability [4,5]. However, much prior work in this area was based on hand-crafted constraints on probabilistic models [6,7,8,9] which may be difficult to design or even detrimental to EDA efficiency and scalability [10]. Recently, Pelikan and Hauschild [11] proposed an automated technique capable of learning from previous runs of the hierarchical Bayesian optimization algorithm (hBOA) in order to improve efficiency of future hBOA runs on problems of similar type. The

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 173–183, 2012.

basic idea of the approach was to (1) design a distance metric on problem variables that correlates with the expected strength of dependencies between the variables, (2) collect statistics on hBOA models with respect to the values of the distance metric, and (3) use the collected statistics to bias model building in hBOA when solving future problem instances of similar type. While the distance metric is strongly related to the problem being solved, the aforementioned study [11] described a rather general metric that can be applied to practically any problem with the objective function represented by an additively decomposable function. However, the prior study [11] evaluated the proposed technique on only two classes of problems and it did not demonstrate several key features of this technique.

The purpose of this paper is threefold: (1) Demonstrate the technique from ref. [11] on other classes of challenging optimization problems, (2) demonstrate the ability of this technique to learn from problem instances of one size in order to introduce bias for instances of another size, and (3) demonstrate the potential benefits of combining this technique with other efficiency enhancement techniques, such as sporadic model building [12]. As test problems the paper considers several classes of NP-complete additively decomposable problems, including MAXSAT, three-dimensional Ising spin glass, and minimum vertex cover. The new results together with the results published in prior work [11] provide strong evidence of the broad applicability and great potential of this technique for learning from experience (transfer learning) in EDAs.

The paper is organized as follows. Section 2 outlines hBOA. Section 3 discusses efficiency enhancement of estimation of distribution algorithms using inductive transfer with main focus on hBOA and the distance-based bias [11]. Section 4 presents and discusses experimental results. Section 5 summarizes and concludes the paper.

2 Hierarchical BOA

The hierarchical Bayesian optimization algorithm (hBOA) [4,13] works with a population of candidate solutions represented by fixed-length strings over a finite alphabet. In this paper, candidate solutions are represented by n-bit binary strings. The initial population of binary strings is generated at random according to the uniform distribution over candidate solutions. Each iteration starts by selecting promising solutions from the current population; here binary tournament selection without replacement is used. Next, hBOA (1) learns a Bayesian network with local structures [14] for the selected solutions and (2) generates new candidate solutions by sampling the distribution encoded by the built network. To maintain useful diversity in the population, the new candidate solutions are incorporated into the original population using restricted tournament selection (RTS) [15]. The run is terminated when termination criteria are met. In this paper, each run is terminated either when the global optimum is found or when a maximum number of iterations is reached.

hBOA represents probabilistic models of candidate solutions by Bayesian networks with local structures [14,16]. A Bayesian network is defined by two

components: (1) an acyclic directed graph over problem variables specifying direct dependencies between variables and (2) conditional probabilities specifying the probability distribution of each variable given the values of the variable's parents. A Bayesian network encodes a joint probability distribution as $p(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i|\Pi_i)$ where X_i is the ith variable (string position) and Π_i are the parents of X_i in the underlying graph.

To represent conditional probabilities of each variable given the variable's parents, hBOA uses decision trees [13,14]. Each internal node of a decision tree specifies a variable, and the subtrees of the node correspond to the different values of the variable. Each leaf of the decision tree for a particular variable defines the probability distribution of the variable given a condition specified by the constraints given by the path from the root of the tree to this leaf (constraints are given by the assignments of the variables along this path).

To build probabilistic models, hBOA typically uses a greedy algorithm that initializes the decision tree for each problem variable X_i to a single-node tree that encodes the unconditional probability distribution of X_i. In each iteration, the model building algorithm tests how much a model would improve after splitting each leaf of each decision tree on each variable that is not already located on the path to the leaf. The algorithm executes the split that provides the most improvement, and the process is repeated until no more improvement is possible. Models are evaluated using the Bayesian-Dirichlet (BDe) metric with penalty for model complexity, which estimates the goodness of a Bayesian network structure given data D and background knowledge ξ as $p(B|D, \xi) = cp(B|\xi)p(D|B, \xi)$, where c is a normalization constant [14,17]. The Bayesian-Dirichlet metric estimates the term $p(D|B, \xi)$ by combining the observed and prior statistics for relevant combinations of variables [14]. To favor simpler networks to the more complex ones, the prior probability $p(B|\xi)$ is often set to decrease exponentially fast with respect to the description length of the network's parameters [4,16].

3 Learning from Experience Using Distance-Based Bias

In hBOA and other EDAs based on complex probabilistic models, building an accurate probabilistic model is crucial to the success [2,3,10,18]. However, building complex probabilistic models can be time consuming and it may require rather large populations of solutions [2,3]. That is why much effort has been put into enhancing efficiency of model building in EDAs and improving quality of EDA models even with smaller populations [5,7,8,19,20]. Learning from experience [4,5,11,19,20] represents one approach to addressing this issue.

The basic idea of learning from experience is to gather information about the problem by examining previous runs of the optimization algorithm and to use the obtained information to bias the search on new problem instances. The use of bias based on the results of other learning tasks is also commonplace in machine learning where it is referred to as *inductive transfer* or *transfer learning* [21,22]. Since learning model structure is often the most computationally expensive task in model building, learning from experience often focuses on identifying regularities in model structure and using these regularities to bias structural learning in future runs.

Analyzing probabilistic models built by hBOA and other EDAs is straightforward. The more challenging facet of implementing learning from experience in practice is that one must make sure that the collected statistics are meaningful with respect to the problem being solved. The key to make the learning from experience work is to ensure that the pairs of variables are classified into a set of *categories* so that the pairs in each category have a lot in common and can be expected to be either correlated or independent simultaneously [11]. This section describes one approach to doing that [11], in which pairs of variables are classified into categories based on a predefined distance metric on variables.

3.1 Distance Metric for Additively Decomposable Functions

For many optimization problems, the objective function (fitness function) can be expressed as or approximated by an additively decomposable function (ADF):

$$f(X_1, \ldots, X_n) = \sum_{i=1}^{m} f_i(S_i), \tag{1}$$

where $(X_1, \ldots, X_n)$ are problem's decision variables (string positions), f_i is the ith subfunction, and $S_i \subset \{X_1, X_2, \ldots, X_n\}$ is the subset of variables contributing to f_i. Typically, the sets $\{S_i\}$ are *not* disjoint. While there may often exist multiple ways of decomposing the problem using additive decomposition, one would typically prefer decompositions that minimize the sizes of subsets $\{S_i\}$. Note that the difficulty of ADFs is not fully determined by the order of subproblems, but also by the definition of the subproblems and their interaction; even with subproblems of order only 2 or 3, the problem can be NP-complete.

The definition of a distance between two variables of an ADF used in this paper as well as ref. [11] follows the work of Hauschild et al. [5,10,19]. Given an ADF, we define the distance between two variables using a graph G of n nodes, one node per variable. For any two variables X_i and X_j in the same subset S_k, we create an edge in G between the nodes X_i and X_j. Denoting by $l_{i,j}$ the number of edges along the shortest path between X_i and X_j in G (in terms of the number of edges), we define the distance between two variables as

$$D(X_i, X_j) = \begin{cases} l_{i,j} & \text{if a path between } X_i \text{ and } X_j \text{ exists,} \\ n & \text{otherwise.} \end{cases}$$

The above distance measure makes variables in the same subproblem close to each other, whereas for the remaining variables, the distances correspond to the length of the chain of subproblems that relate the two variables. The distance is maximal for variables that are completely independent (the value of the first variable does not influence the contribution of the second variable in any way).

Since interactions between problem variables are encoded mainly in the subproblems of the additive problem decomposition, the above distance metric should typically correspond closely to the likelihood of dependencies between problem variables in probabilistic models discovered by EDAs. Specifically, the variables located closer with respect to the metric should more likely interact

with each other. This observation has been confirmed with numerous experimental studies across a number of important problem domains from spin glasses distributed on a finite-dimensional lattice [10,11] to NK landscapes [11].

3.2 Distance-Based Bias Based on Previous Runs of hBOA

This section describes the approach to learning from experience developed by Pelikan and Hauschild [11] inspired mainly by the work of Hauschild et al. [5,19,20]. Let us assume a set M of hBOA models from prior hBOA runs on similar problems; in the experiments we use *all* models obtained from prior runs as the starting point. Before applying the bias based on prior runs, the models in M are first processed to generate data that will serve as the basis for introducing the bias. The processing starts by analyzing the models in M to determine the number $s(m, d, j)$ of splits on any variable X_i such that $D(X_i, X_j) = d$ in a decision tree T_j for variable X_j in a model $m \in M$. Then, the values $s(m, d, j)$ are used to compute the probability $P_k(d, j)$ of a kth split on a variable at distance d from X_j in a dependency tree T_j given that $k - 1$ such splits were already performed in T_j:

$$P_k(d, j) = \frac{|\{m \in M : s(m, d, j) \geq k\}|}{|\{m \in M : s(m, d, j) \geq k - 1\}|}. \tag{2}$$

Recall that the BDe metric for evaluating the quality of probabilistic models in hBOA contains two parts: (1) the prior probability $p(B|\xi)$ of the network structure B, and (2) the posterior probability $p(D|B, \xi)$ of the data (population of selected solutions) given B. Pelikan and Hauschild [11] proposed to use the prior probability distribution $p(B|\xi)$ to introduce a bias based on distance-based statistics from previous hBOA runs represented by $P_k(d, j)$ by setting

$$p(B|\xi) = c \prod_{d=1}^{n} \prod_{j=1}^{n} \prod_{k=1}^{n_s(d,j)} P_k^{\kappa}(d, j), \tag{3}$$

where $n_s(d, j)$ denotes the number of splits on any variable X_i in T_j such that $D(X_i, X_j) = d$, $\kappa > 0$ is used to tune the strength of bias (the strength of bias increases with κ), and c is a normalization constant. Since log-likelihood is typically used to evaluate model quality, when evaluating the contribution of any particular split, the change of the prior probability of the network structure can still be done in constant time.

4 Experiments

4.1 Test Problems and Experimental Setup

The experiments were done for three problem classes known to be difficult for most genetic and evolutionary algorithms: (1) Three-dimensional Ising spin glasses were considered with $\pm J$ couplings and periodic boundary conditions [23,24]; two problem sizes were used, $n = 6 \times 6 \times 6 = 216$ spins and

$n = 7 \times 7 \times 7 = 343$ spins with 1,000 unique problem instances for each n. (2) Minimum vertex cover was considered for random graphs with a fixed ratio c of the number of edges and number of nodes [25,26]; two ratios ($c = 2$ and $c = 4$) and two problem sizes ($n = 150$ and $n = 200$) were used with 1,000 unique problem instances for each combination of c and n. (3) MAXSAT was considered for mapped instances of graph coloring with graphs created by combining regular ring lattices (with probability $1 - p$) and random graphs (with probability p) [27,28]; 100 unique problem instances of $n = 500$ bits (propositions) were used for each considered value of p, from $p = 2^{-8}$ (graphs nearly identical to a regular ring lattice) to $p = 2^{-1}$ (graphs with half of the edges random). For more information about the test problems, we refer the reader to refs. [23,25,27].

The maximum number of iterations for each problem instance was set to the number of bits in the problem; according to preliminary experiments, this upper bound was sufficient. Each run was terminated either when the global optimum was found or when the maximum number of iterations was reached. For each problem instance, we used bisection [4,29] to ensure that the population size was within 5% of the minimum population size to find the optimum in 10 out of 10 independent runs. Bit-flip hill climbing (HC) [4] was incorporated into hBOA to improve its performance on all test problems except for the minimum vertex cover; HC was used to improve every solution in the population. For minimum vertex cover, a repair operator based on ref. [25] was incorporated instead. The strength of the distance-based bias was tweaked using $\kappa \in \{1, 3, 5, 7, 9\}$; the greater the value of κ, the stronger the bias.

To ensure that the same problem instances were not used for defining the bias as well as for testing it, 10-fold crossvalidation was used when evaluating the effects of distance-based bias derived from problem instances of the same size. For each set of problems (by a set of problems we mean a set of random problem instances generated with one specific set of parameters), problem instances were randomly split into 10 equally sized subsets. In each round of crossvalidation, 1 subset of instances was left out and hBOA was run on the remaining 9 subsets of instances. The runs on the 9 subsets produced models that were analyzed in order to obtain the probabilities $P_k(d, j)$ for all d, j, and k. The bias based on the obtained values of $P_k(d, j)$ was then used in hBOA runs on the remaining subset of instances. The same procedure was repeated for each subset; overall, 10 rounds of crossvalidation were performed for each set of instances. When evaluating the effects of distance-based bias derived from problem instances of *smaller* size, we did not use crossvalidation because in this case all runs had to be done on different problem instances (of different size). Most importantly, in every experiment, models used to generate statistics for hBOA bias were obtained from hBOA runs on *different* problem instances. While the experiments were performed across a variety of computer architectures and configurations, the base case with no bias and the case with bias were always both run on the same computational node; the results of the two runs could therefore be compared against each other with respect to the actual CPU (execution) time.

To evaluate hBOA performance, we focus on the multiplicative speedup with respect to the execution time per run; the speedup is defined as a multiplicative factor by which the execution time improves with the distance-based bias compared to the base case. For example, an execution-time speedup of 2 indicates that the bias allowed hBOA to find the optimum using only half the execution time compared to the base case without the bias. We also report the percentage of runs for which the execution time was strictly improved (shown in parentheses after the corresponding average multiplicative speedup).

In addition to the speedups achieved for various values of κ, we examine the ability of the distance-based bias based on prior runs to apply across a range of problem sizes; this is done by using previous runs on instances of one size to bias runs on instances of another size. Since for MAXSAT, we only used instances of one size, this facet was only examined for the other two problem classes.

Finally, we examine the combination of the distance-based bias based on prior runs and the sporadic model building [12]. Specifically, we apply sporadic model building on its own using the model-building delay of $\sqrt{n}/2$ as suggested by ref. [12], and then we carry out a similar experiment using both the distance-based bias as well as the sporadic model building, recording the speedups with respect to the base case. Ideally, we would expect the speedups from the two sources to multiply. Due to the time requirements of solving MAXSAT, the combined effects were studied only for the remaining two problem classes.

4.2 Results

The results presented in tables 1, 2 and 3 confirm the observation from ref. [11] that the stronger the bias the greater the benefits, at least for the examined range of $\kappa \in \{1, 3, 5, 7, 9\}$ and most problem settings; that is why in the remainder of this discussion we focus on $\kappa = 9$. In all cases, the distance-based bias yielded substantial speedups of about 1.2 to 3.1. Best speedups were obtained for the minimum vertex cover. In all cases, performance on at least about 70% problem instances was strictly improved in terms of execution time; in most cases,

Table 1. Results for 3D spin glass

(a) Results for 10-fold crossvalidation with priors from other instances of the same size.

κ	CPU speedup	
	$n = 216$	$n = 343$
1	0.40 (0%)	0.43 (0%)
3	1.00 (43%)	1.08 (60%)
5	1.23 (71%)	1.32 (85%)
7	1.24 (70%)	1.34 (81%)
9	1.21 (66%)	1.20 (67%)

(b) Results for $n = 343$ with priors based on models obtained on problem instances of *smaller* size, $n = 216$.

κ	CPU speedup
1	0.43 (1%)
3	1.05 (61%)
5	1.33 (85%)
7	1.34 (82%)
9	1.26 (75%)

(c) Results for a combination of distance-based bias (DBB) and sporadic model building (SMB) for $n = 343$. 10-fold crossvalidation was used.

κ	CPU speedup	
	DBB+SMB	SMB
1	1.85 (99%)	3.20 (99%)
3	3.29 (99%)	3.20 (99%)
5	4.04 (99%)	3.20 (99%)
7	4.23 (99%)	3.20 (99%)
9	4.03 (99%)	3.20 (99%)

Table 2. Results for minimum vertex cover

(a) Results for 10-fold crossvalidation with priors from other instances of the same size.

$c = 2$		
κ	CPU speedup	
	$n = 150$	$n = 200$
1	0.57 (2%)	0.45 (0%)
3	1.95 (91%)	1.63 (87%)
5	2.78 (96%)	2.69 (94%)
7	3.04 (95%)	2.98 (94%)
9	3.10 (93%)	2.95 (92%)

$c = 4$		
κ	CPU speedup	
	$n = 150$	$n = 200$
1	0.28 (0%)	0.17 (0%)
3	0.97 (39%)	0.53 (4%)
5	1.56 (82%)	1.16 (62%)
7	1.97 (88%)	1.65 (81%)
9	2.27 (89%)	1.91 (85%)

(b) Results for $n = 200$ with priors based on models obtained on problem instances of *smaller* size, $n = 150$.

$c = 2$	
κ	CPU speedup
1	0.53 (2%)
3	1.95 (91%)
5	2.79 (95%)
7	2.99 (94%)
9	3.02 (91%)

$c = 4$	
κ	CPU speedup
1	0.23 (0%)
3	0.86 (27%)
5	1.50 (79%)
7	1.89 (85%)
9	2.12 (84%)

(c) Results for a combination of distance-based bias (DBB) and sporadic model building (SMB) for $n = 200$. 10-fold crossvalidation was used.

$c = 2$		
κ	CPU speedup	
	DBB+SMB	SMB
1	3.12 (99%)	4.89
3	6.89 (100%)	4.89
5	10.25 (100%)	4.89
7	11.38 (100%)	4.89
9	11.29 (99%)	4.89

$c = 4$		
κ	CPU speedup	
	DBB+SMB	SMB
1	1.88 (82%)	4.54
3	3.24 (96%)	4.54
5	5.00 (99%)	4.54
7	6.15 (99%)	4.54
9	6.60 (99%)	4.54

Table 3. Results for MAXSAT

κ	CPU speedup			
	$p = 2^{-1}$	$p = 2^{-2}$	$p = 2^{-4}$	$p = 2^{-8}$
1	0.13 (0%)	0.22 (0%)	0.22 (0%)	0.38 (0%)
3	0.41 (0%)	0.53 (0%)	0.48 (0%)	1.01 (49%)
5	0.81 (25%)	0.82 (18%)	0.74 (4%)	1.63 (100%)
7	1.38 (69%)	1.09 (55%)	1.03 (54%)	1.84 (100%)
9	2.31 (94%)	1.38 (81%)	1.28 (89%)	1.90 (100%)

the improvements were observed in a much greater majority of instances. The speedups were substantial even when the bias was based on prior runs on problem instances of different, smaller size; in fact, the speedups obtained with such a bias were nearly identical to the speedups with the bias based on the instances of the same size. The results thus provide clear empirical evidence that the distance-based bias is applicable even when the problem instances vary in size, which was argued [11] to be one of the main advantages of the distance-based bias over prior work in the area but was not demonstrated. Finally, the results show the nearly multiplicative effect of the distance-based bias and sporadic model building, providing further support for the importance of the distance-based bias; the combined speedups ranged from about 4 to more than 11.

5 Summary and Conclusions

This paper extended the prior work on efficiency enhancement of the hierarchical Bayesian optimization algorithm (hBOA) using a distance-based bias derived from prior hBOA runs [11]. The paper demonstrated that (1) the distance-based bias yields substantial speedups on several previously untested classes of challenging, NP-complete problems, (2) the approach is applicable even when prior runs were executed on problem instances of different size, and (3) the approach can yield nearly multiplicative speedups when combined with other efficiency enhancement techniques. In summary, the results presented in this paper together with the prior work [11] provide clear evidence that learning from experience using a distance-based bias has a great potential to improve efficiency of hBOA in particular and estimation of distribution algorithms (EDAs) in general. It is of note that thus far we have *not* found a single problem class for which the distance-based bias based on prior runs would fail to yield speedups.

In future work, the approach should be adapted to other model-directed optimization techniques, including other EDAs and genetic algorithms with linkage learning. The approach should also be modified to introduce bias on problems that cannot be formulated using an additive decomposition in a straightforward manner or such a decomposition is not practical. It would also be interesting to investigate the effects of a bias based on problem instances that are not as similar, for example, by learning a bias on NK landscapes and applying the learned bias to MAXSAT or 3D spin glass. Finally, it is important to study the limitations of the proposed approach, and create theoretical models to automatically tune the strength of the bias and predict expected speedups.

Acknowledgments. This project was sponsored by the National Science Foundation under grants ECS-0547013 and IIS-1115352, and by the Univ. of Missouri–St. Louis through the High Performance Computing Collaboratory sponsored by Information Technology Services. Most experiments were performed on the Beowulf cluster maintained by ITS at the Univ. of Missouri in St. Louis and the HPC resources at the University of Missouri Bioinformatics Consortium. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Hauschild, M.W., Pelikan, M.: An introduction and survey of estimation of distribution algorithms. Swarm and Evolutionary Computation 1(3), 111–128 (2011)
2. Larrañaga, P., Lozano, J.A. (eds.): Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer, Boston (2002)
3. Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications 21(1), 5–20 (2002)

4. Pelikan, M.: Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms. Springer (2005)
5. Hauschild, M.W., Pelikan, M., Sastry, K., Goldberg, D.E.: Using previous models to bias structural learning in the hierarchical BOA. Evolutionary Computation 20(1), 135–160 (2012)
6. Mühlenbein, H., Mahnig, T., Rodriguez, A.O.: Schemata, distributions and graphical models in evolutionary optimization. Journal of Heuristics 5, 215–247 (1999)
7. Mühlenbein, H., Mahnig, T.: Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. International Journal of Approximate Reasoning 31(3), 157–192 (2002)
8. Baluja, S.: Incorporating a priori knowledge in probabilistic-model based optimization. In: Pelikan, Sastry, Cantú-Paz (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, pp. 205–219. Springer (2006)
9. Schwarz, J., Ocenasek, J.: A problem-knowledge based evolutionary algorithm KBOA for hypergraph partitioning. In: Proc. of the Fourth Joint Conf. on Knowledge-Based Software Engineering, Brno, Czech Rep., pp. 51–58 (2000)
10. Hauschild, M.W., Pelikan, M., Sastry, K., Lima, C.F.: Analyzing probabilistic models in hierarchical BOA. IEEE Transactions on Evolutionary Computation 13(6), 1199–1217 (2009)
11. Pelikan, M., Hauschild, M.: Distance-based bias in model-directed optimization of additively decomposable problems. MEDAL Report No. 2012001, Missouri Estimation of Distribution Algorithms Laboratory, University of Missouri–St. Louis, St. Louis, MO (2012)
12. Pelikan, M., Sastry, K., Goldberg, D.E.: Sporadic model building for efficiency enhancement of the hierarchical BOA. Genetic Programming and Evolvable Machines 9(1), 53–84 (2008)
13. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. In: Genetic and Evol. Comp. Conf (GECCO 2001), pp. 511–518 (2001)
14. Chickering, D.M., Heckerman, D., Meek, C.: A Bayesian approach to learning Bayesian networks with local structure. Technical Report MSR-TR-97-07, Microsoft Research, Redmond, WA (1997)
15. Harik, G.R.: Finding multimodal solutions using restricted tournament selection. In: Proc. of the Int. Conf. on Genetic Algorithms (ICGA 1995), pp. 24–31 (1995)
16. Friedman, N., Goldszmidt, M.: Learning Bayesian networks with local structure. In: Jordan, M.I. (ed.) Graphical Models, pp. 421–459. MIT Press (1999)
17. Cooper, G.F., Herskovits, E.H.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning 9, 309–347 (1992)
18. Lima, C.F., Lobo, F.G., Pelikan, M., Goldberg, D.E.: Model accuracy in the Bayesian optimization algorithm. Soft Computing 15(7), 1351–1371 (2011)
19. Hauschild, M., Pelikan, M.: Enhancing Efficiency of Hierarchical BOA Via Distance-Based Model Restrictions. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN X. LNCS, vol. 5199, pp. 417–427. Springer, Heidelberg (2008)
20. Hauschild, M.W., Pelikan, M.: Intelligent bias of network structures in the hierarchical BOA. In: Genetic and Evol. Comp. Conf. (GECCO 2009), pp. 413–420 (2009)
21. Pratt, L.Y., Mostow, J., Kamm, C.A., Kamm, A.A.: Direct transfer of learned information among neural networks. In: Proceedings of the Ninth National Conference on Artificial Intelligence, pp. 584–589 (1991)
22. Caruana, R.: Multitask learning. Machine Learning 28, 41–75 (1997)

23. Pelikan, M., Hartmann, A.K.: Searching for ground states of Ising spin glasses with hierarchical BOA and cluster exact approximation. In: Pelikan, M., Sastry, K., Cantú-Paz, E. (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications. Springer (2006)
24. Young, A. (ed.): Spin glasses and random fields. World Scientific, Singapore (1998)
25. Pelikan, M., Kalapala, R., Hartmann, A.K.: Hybrid evolutionary algorithms on minimum vertex cover for random graphs. In: Genetic and Evol. Comp. Conf. (GECCO 2007), pp. 547–554 (2007)
26. Weigt, M., Hartmann, A.K.: Minimal vertex covers on finite-connectivity random graphs: A hard-sphere lattice-gas picture. Physical Review E 63, 056127 (2001)
27. Pelikan, M., Goldberg, D.E.: Hierarchical BOA Solves Ising Spin Glasses and Maxsat. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2724, pp. 1271–1282. Springer, Heidelberg (2003)
28. Gent, I., Hoos, H.H., Prosser, P., Walsh, T.: Morphing: Combining structure and randomness. In: Proc. of the American Association of Artificial Intelligence (AAAI 1999), pp. 654–660 (1999)
29. Sastry, K.: Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL (2001)

Reinforcement Learning with N-tuples on the Game Connect-4

Markus Thill, Patrick Koch, and Wolfgang Konen*

Department of Computer Science, Cologne University of Applied Sciences,
51643 Gummersbach, Germany
{patrick.koch,wolfgang.konen}@fh-koeln.de

Abstract. Learning complex game functions is still a difficult task. We apply temporal difference learning (TDL), a well-known variant of the reinforcement learning approach, in combination with n-tuple networks to the game Connect-4. Our agent is trained just by self-play. It is able, for the first time, to consistently beat the optimal-playing Minimax agent (in game situations where a win is possible). The n-tuple network induces a mighty feature space: It is not necessary to design certain features, but the agent learns to select the right ones. We believe that the n-tuple network is an important ingredient for the overall success and identify several aspects that are relevant for achieving high-quality results. The architecture is sufficiently general to be applied to similar reinforcement learning tasks as well.

Keywords: Machine learning, reinforcement learning, TDL, self-play, n-tuple systems, feature generation, board games.

1 Introduction

1.1 Learning

Our understanding of learning processes is still limited, especially in complex decision-making situations, where the payoff for a particular action occurs only later, probably a long time after the action is executed. The fact that the payoff occurs only after a number of subsequent actions leads to the well known *credit assignment problem*: decide which action should get which credit for a certain payoff. The most advanced methods in machine learning to address this problem are reinforcement learning (RL), e. g., the well-known temporal difference learning (TDL), and evolutionary algorithms, namely evolution strategies (ES) and co-evolution.

TDL was applied as early as 1957 by Samuel [9] to checkers and gained more popularity through Sutton's work in 1984 and 1988 [13,14]. It became very famous in 1994 with Tesauro's TD-Gammon [15], which learned to play backgammon at expert level.

* This work has been partially supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grant SOMA ("Ingenieurnachwuchs" 2009).

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 184–194, 2012.

Learning to play board games has a long tradition in AI. This is due to most board games having simple rules, nevertheless, they encode surprisingly complex decision-making situations including the above mentioned credit assignment problem. The search for an optimal playing agent constitutes an interesting branch of optimization problems: In normal optimization problems, an objective function is known, i. e. a function to assess the quality of a solution. Contrariwise, no such objective function exists for many board games, or it is computationally too expensive to calculate. For example, most board positions in chess are too complex to be analyzed in full depth. Moreover, the strength of an agent is the quality of its move in *all* relevant board positions. Instead of this inaccessible objective function, we often use certain *interactions* as primary driver of the search, which serve as a surrogate for the objective function [7]. On a simple level, interaction can take place between a solution and itself: A well-known example is self-play, a technique used for game strategy learning, where an agent plays many games against itself. In population-based methods like co-evolutionary algorithms, interaction involves two or more different solutions from one or from several populations.

In particular, we are interested in such learning problems, where a 'true' objective function is not accessible for learning. Many problems in practice are like this: We need to 'play well' in a certain environment, but the objective function is either not known or it may not be invoked often enough for all the learning trials needed. Given the right learning architecture, how much can we learn from self-play? Nature itself provides us with incredibly convincing examples, for instance, children, who learn in complex situations from only few interactions with the environment. It is an open question, how much internal self-play contributes to such learning successes. Our goal is to mimic at least some of these awesome learning successes observed in nature with machine learning architectures.

1.2 Related Work

In our previous work concerned with the learning of game functions, we compared CMA-ES and TDL [6] and investigated the role of features and feature generation on learning and self-organization [5].

In this paper we consider the game Connect-4 as a specific example, which is solved on the AI-level (see Sec. 2.1). Only rather few attempts to *learn* Connect-4 (whether by self-play or by learning from teachers) are found in the literature: Schneider et al. [10] tried to learn Connect-4 with a neural network, using an archive of saved games as teaching information. Sommerlund [11] applied TDL to Connect-4 but obtained rather discouraging results. Stenmark [12] compared TDL against a knowledge-based approach from Automatic Programming and found TDL to be slightly better. Curran et al. [3] used a cultural learning approach for evolving populations of neural networks in self-play to play Connect-4. All the above works gave no clear answer on the playing strength of the agents, since they did not compare their agents with a perfect-playing Minimax agent.

Some preliminary work in our institution on learning Connect-4 with neural nets or TDL also did not lead to convincing success. Only small subsets of Connect-4 could be learned to some extent.

Then Lucas showed with the n-tuple-approach [8] that the game of Othello, having a somewhat greater complexity than Connect-4, could be learned by TDL within a few thousand training games. The n-tuple-approach is basically a clever approach to introduce a rich variety of features into board games. Krawiec et al. [7] applied the n-tuple-approach in (Co-) Evolutionary TDL and outperformed TDL in the Othello League. This stirred new interest in our Connect-4 project and gave rise to the following research questions:

Q1 Is the n-tuple approach also applicable to Connect-4, i. e. can we, for the first time, get good results from self-play using n-tuples?

Q2 Can we do so with relatively *few training games* when using enough n-tuples?

Q3 Which ingredients are crucial for the success of learning?

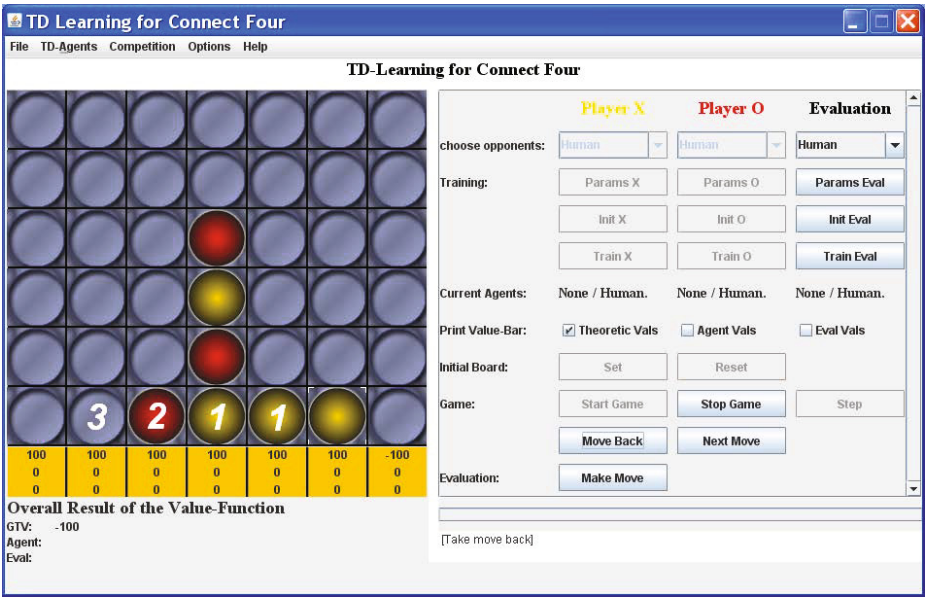


Fig. 1. Connect-4 board with an example 4-tuple '3-2-1-1' (see Sec. 2.2)

2 Methods

2.1 Connect-4

The game of Connect-4 is a two-player game played on a board with 7 vertical slots containing 6 positions each (Fig. 1). Player Yellow (1st) and player Red (2nd) place one piece per turn in one of the available slots and each piece falls down under the force of gravity into the lowest free position of the slot. Each player attempts to create horizontal, vertical or diagonal piece-lines of length

four. Fig. 1 shows an example position where Yellow would win if Red does not block this by placing a red piece into the right slot.

Connect-4 has a medium state space complexity of $4.5{\cdot}10^{12}$ board positions [4]. It can be solved by a combination of game tree search and clever heuristics with a few days of computing time. Connect-4 was solved in 1988 independently by Allen and by Allis [1]: Yellow (the 1st player) wins, if she places her piece in the middle slot.

We developed a Minimax agent combined with a pre-calculated 8-ply or 12-ply-opening database [16] and it finds the perfect next move (or moves, if several moves are equally well) for each board position within fractions of a second. This agent will be used in our experiments only as referee or as evaluating agent. It is by no means used for any training purpose.

2.2 N-tuples and LUTs

N-tuples in General. N-tuple systems have first been introduced in 1959 by Bledsoe and Browning [2] for character recognition. Their main advantages include conceptual simplicity and capability of realizing non-linear mappings to spaces of higher dimensionality [7]. Recently, Lucas proposed employing the n-tuple architecture also for game-playing purposes [8]. The n-tuple approach works in a way similar to the kernel trick used in support vector machines (SVM): The low dimensional board is projected into a high dimensional sample space by the n-tuple indexing process [8].

N-tuples in Connect-4. Each n-tuple T_i is a sequence $[a_{i0}, \ldots, a_{in-1}]$ of n different board locations where each a_{ij} codes a specific cell of the board. For example, the four white digits in Fig. 1 mark the board locations of a 4-tuple. Each location possesses one of P possible states $z[a_{ij}] \in \{0, \ldots, P-1\}$ (the value of the digits in our example). An n-tuple of length n thus has P^n possible states $k \in \{0, \ldots, P^n-1\}$. The number represented by the state of T_i can be used as an index into an associated look-up table LUT_i, which contains parameters $w_{i,t}[k]$ equivalent to weights in standard neural networks. For a given board position z, the output of the n-tuple network can be calculated as:

$$f(\boldsymbol{w}_t, z_t) = \sum_{i=0}^{m} w_{i,t}\,[k] \quad \text{with} \quad k = \sum_{j=0}^{n-1} z_t[a_{ij}]P^j. \tag{1}$$

Here, $z_t[a_{ij}]$ is the state of board location a_{ij} at time t. Likewise, $w_{i,t}[k]$ is the weight for state k of n-tuple T_i, $i = 1, \ldots, m$. It is also a function of time t since it will be modified by the TD learning rule, see Sec. 2.3. The vector $\boldsymbol{w}_t$ combines all weights from all LUTs at time t.

Position Encoding. We consider two alternative approaches for Connect-4:

- **P=3:** Each board location has one of the 3 z-values: 0=empty, 1=Yellow, 2=Red.
- **P=4:** Each board location has one of the 4 z-values: 0=empty and not reachable, 1=Yellow, 2=Red, 3=empty and reachable.

By *reachable* we mean an empty cell that can be occupied in the next move. The reason behind the (P=4)-encoding is that it makes a difference whether e. g. three yellow pieces in a row have a reachable empty cell adjacent to them (a direct threat for Red) or a non-reachable cell (only indirect threat).

Fig. 1 shows an example board position with a 4-tuple in the numbered cells. The state of the 4-tuple is $k = 3 \cdot 4^0 + 2 \cdot 4^1 + 1 \cdot 4^2 + 1 \cdot 4^3 = 91$.

Symmetry. Board games often have several symmetries in the board position. In case of Connect-4, there is only one symmetry, the mirror reflection of the board along the middle column. If k denotes an n-tuple state for a certain board position, we denote with $M(k)$ its state in the mirror-reflected board. As proposed by Lucas [8] we can augment the n-tuple network output of Eq. (1) to

$$f(\boldsymbol{w}_t, z_t) = \sum_{i=0}^{m} (w_{i,t}[k] + w_{i,t}[M(k)]). \tag{2}$$

N-tuple Creation. There are different methods how n-tuples can be created: The purely random sampling of board cells is possible but not very sensible for game feature search. It is more advisable to select adjacent locations because the goal of the game is to place 4 adjecent pieces. Therefore, we propose a random walk very similar to the snake method introduced by Lucas [8]: To create an n-tuple of length K, we start at a random cell, which is the first cell of the n-tuple. Afterwards, we visit one of its 8 neighbors and add it to the n-tuple (if not already in there). We continue to one of its neighbors, and so on, until we have K cells in the n-tuple. The only difference to Lucas' snake method is our n-tuple all having a fixed length of K, while snakes can have lengths $2, \ldots, K$.

2.3 TDL

The goal of the agent is to predict the ideal *value function*, which usually is 1.0 if the board position is a win for Yellow, and -1.0 if it is a win for Red. The TD algorithm aims at *learning* the value function. It does so by setting up an (initially inexperienced) agent, who plays a sequence of games against itself. It learns from the environment, which gives a reward $r \in \{-1.0, 0.0, 1.0\}$ for { Yellow-win, Draw, Red-win } at the end of each game . The main ingredient is the *temporal difference* (TD) error signal [14]

$$\delta_t = V(\boldsymbol{w}_t, z_{t+1}) - V(\boldsymbol{w}_t, z_t). \tag{3}$$

Here, $V(\boldsymbol{w}_t, z_t) = \sigma\left(f(\boldsymbol{w}_t, z_t)\right)$ is the agent's current approximation of the value function on the basis of Eq. (2) and a nonlinear sigmoid function σ (we choose $\sigma = \tanh$). If the state observed at time t + 1 is terminal, the exact value r of the game is used in Eq. (3) instead of the prediction $V(\boldsymbol{w}_t, z_{t+1})$. The weights are trained with the usual δ-rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \delta_t \boldsymbol{\nabla}_w V(\boldsymbol{w}_t, z_t), \tag{4}$$

which aims at making the preceding prediction match the current prediction more closely. More details on TDL in games can be found in our previous work [5].

Table 1. Computation time, number of weights and (for a specific n-tuple set) non-zero weights. The count of non-zero weights varies a bit with different n-tuple sets, but always has the same order of magnitude.

position encoding	time (10×10^7 games)	weights	non-zero weights
P=3	6h 45min	918.540	304.444
P=4	7h 50min	9.175.040	646.693

2.4 Agent Evaluation

A fair agent evaluation for board games is not trivial. There is no closed-form objective function for 'agent playing strength' since the evaluation of all board positions is infeasible and it is not clear, which relevance has to be assigned to each position. The most common approach is to assess an agent's strength by observing its *interactions* with other agents, either in a tournament or against a referee agent. We choose the Minimax agent to be the ultimate referee. Note that all the approaches to Connect-4 found in the literature (cf. Sec. 1) fail to provide a common reference point for the strength of the agents generated.

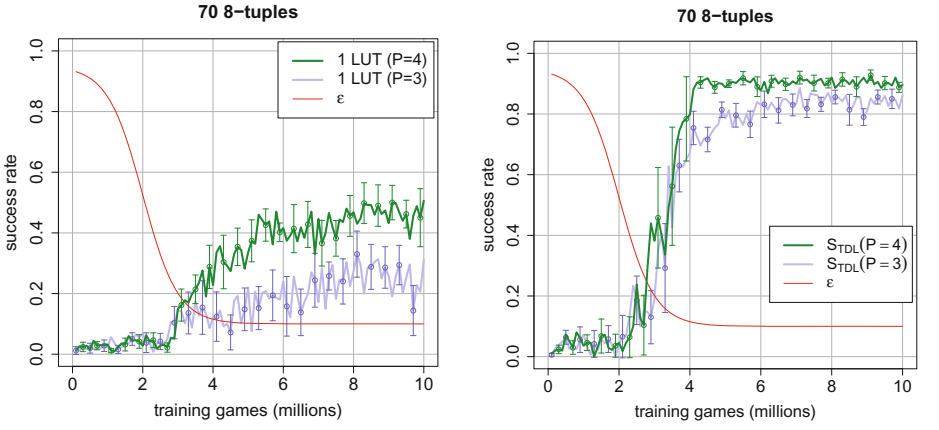
Our approach to agent evaluation in Connect-4 is as follows: When TDL training is finished, TDL (Yellow) and Minimax (Red) play a tournament of 50 games. (Since Minimax will always win playing Yellow, we consider only games with Minimax playing Red.) The ideal TDL agent is expected to win every game. If both agents act fully deterministically, each game would be identical. We introduce a source of randomness without sacrificing any agent's strength as follows: If Minimax has several optimal moves at its disposal, it chooses one of them randomly. The TDL agent gets a score of 1 for a win, 0.5 for a draw and 0 for a loss. The overall success rate $S_{TDL} \in [0.0, 1.0]$ is the mean of the 50 individual scores. A perfect TDL agent receives a success rate of 1.0.

3 Experimental Setup

We established a software framework in Java for conducting learning experiments (see Fig. 1) on the basis of the elements described above: TDL agent with n-tuples, Connect-4, Minimax, agent evaluation and visual inspection capabilities. Each n-tuple is allowed to have one or two LUTs (see Sec. 4).[1]

During training and play, the TDL agent does not use any game-tree search (0-ply look-ahead), instead it just inspects the board positions of its legal moves and chooses the best one. Initially, we tested different n-tuple architectures and decided to conduct all our experiments with 70 n-tuples of length 8 afterwards. If not stated otherwise, the learning rate α followed an exponential decay scheme with $\alpha_{init} = 0.01$ and $\alpha_{final} = 0.001$. Each TDL agent was initialized with weights = 0 and trained during 10 million games of self-play and evaluated

[1] The code is available from the authors on request.



(a) No success with 1 LUT for each n-tuple (b) 2 LUTS, one for each player

Fig. 2. Success rate S_{TDL} of the n-tuple-based TDL agent playing Connect-4

every 100,000 games. Table 1 shows the computation time needed on a Q9550 quad-core PC (2.83 GHz on each of the 4 cores) for 10 TDL training experiments.

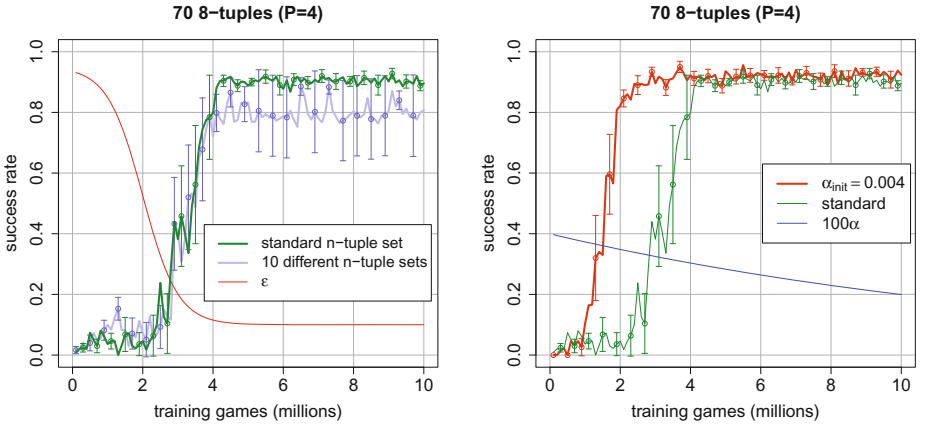
If we have 70 n-tuples of length 8 with 2 LUTs each, the number of LUT-entries in (P=4)-encoding is $2 \cdot 70 \cdot 4^8 = 9.175 \cdot 10^6$. Note that not every state corresponding to a LUT-entry is a realizable state during Connect-4 games: If a column's lower cell is 0 or 3, all cells above must be 0. Combinations like (0,1), (0,1,1,2), ... (from bottom to top) are not possible. Thus we expect a large number of weights to stay at zero since they are never updated during training. Tab. 1 confirms this for an example n-tuple set: After training, the number of non-zero weights is only a fraction of the total number of weights.[2]

A final important ingredient is the exploration rate ϵ: Connect-4 is a deterministic game and deterministic players would always conduct the same game during self-play. Although a TDL agent might slowly change during learning and thus behaves not fully deterministically, it is nearly deterministic and learning would become ineffective. Therefore, a source of randomness is introduced to aid the exploration of the game tree: With probability ϵ the TDL agent chooses its next move randomly. After such a random move, *no* weight update (Eq. (4)) is performed. The parameter ϵ follows a sigmoidal decay scheme, usually with $\epsilon_{init} = 0.95$ and $\epsilon_{final} = 0.1$.

4 Results

First results, cf. Fig. 2(a), were rather discouraging: The TDL agent rarely won against Minimax, even utilizing different n-tuple creation schemes (random walk, random snake, random sample) and both types of position encoding, i.e. P=3

[2] The number of non-zero weights usually amounts to 98-100% of the realizable states.



(a) 10 different n-tuple sets (b) Improved learning rate decay scheme

Fig. 3. Further results with TDL agent playing Connect-4

and P=4. The results show the mean from 10 experiments and even the best experiment has only a mediocre success rate $S_{TDL} < 58\%$.

Two LUTs. The major breakthrough was achieved when we changed the n-tuple architecture from 1 LUT per n-tuple to 2 LUTs, one for each player. In contrast to Othello, a certain position in a subpart of the board can have a rather different value whether it is Yellow's turn or Red's turn. Therefore, it is advisable to learn both concepts separately. As the light blue curve in Fig. 2(b) shows, we, for the first time, receive an 80% success rate. The results are comparatively stable: If we repeat the experiment ten times with the same n-tuple set but different random moves, we get very similar results, cf. Fig. 2(b). The error bars represent the standard deviation from the ten experiments.

(P=4)-Encoding. The darker green curve in Fig. 2(b) shows the beneficial effects of (P=4)-encoding (cf. Sec. 2.2): Compared to the (P=3)-encoding we reach a strong-playing agent much faster (the 80% success rate line is crossed after 4 million games instead of 5 million games), the mean success rate between 6 and 10 million training games is 6% higher (90% instead of 84%), and the variance is lower. We refer to the darker green curve as "standard" in the following experiments.

Different N-tuple Sets. The n-tuple creation process is completely by chance, no game specific considerations are taken into account. We tested whether different randomly selected sets would produce a large variance. The results with 10 different sets are shown in Fig. 3(a): The light blue curve is considerably lower (10% in the range beyond $6 \cdot 10^6$ games) and has a larger variance.

When analyzing this result, we found that half of the variance can be attributed to *one* 'bad' n-tuple set, which completely failed on the task (success rate < 0.1). All others sets were very similar to the standard n-tuple set (approx.

4.5% lower in the range beyond $6 \cdot 10^6$ games). The results are consistent: A 'bad' n-tuple set will always be bad when repeating the training with other random moves, a 'good' set will always be good.

Learning Rate Tuning. We observed that an α-decay scheme with $\alpha_{final} \geq 0.004$ does not learn anything. A possible reason is that, due to conflicting signals, individual weights oscillate up and down without the ability to average over different situations. This is often the case in TDL game learning. The sharp onset in success rate in Fig. 2(b) and Fig. 3(a) is with 4 million training games exactly at the point in training where α drops below 0.004. It seems natural to change the α-decay scheme from $[\alpha_{init}, \alpha_{final}] = [0.01, 0.001]$ (standard) to $[\alpha_{init}, \alpha_{final}] = [0.004, 0.002]$. Having implemented this, we observe a much faster training (the 80% success rate line is crossed after 2 instead of 4 million games) and a slight increase in final performance (92% instead of 90%), cf. Fig. 3(b).

We also tested variations in parameter ϵ (exploration decay scheme), but this does not seem to change the results very much. However, a thorough tuning was not performed yet.

5 Conclusion

We summarize by answering our research questions as stated in Sec. 1.2:

Q1. The n-tuple approach has proven to be very successful for the game of Connect-4. For the first time, we generated a strong-playing agent trained just by self-play, i. e. it had no access to teaching information other than the mere rules of the game. The shapes of the n-tuples were not designed by the programmer, instead the system selected them by random walk.

Q2. Lucas found that 1250 training games were enough to produce a strong-playing TDL agent applying the n-tuple approach to Othello [8]. We cannot confirm this for Connect-4. So far we need 2–4 million training games to produce strong-playing Connect-4 agents. One possible reason, among others, is that Othello has an 8-fold symmetry, while it is only a 2-fold symmetry in Connect-4. Thus, it takes longer to visit all branches of the game tree. It is an open question, whether more or better n-tuples will shorten the training time.

Q3. Among the ingredients crucial for the success of learning there is first of all the architecture {*n-tuple + TDL*} itself: The game Connect-4, despite the fact that it is heuristically solved, seems somewhat hard to code for learning architectures. To the knowledge of the authors, there is no other result in literature, where a learning agent, just trained from self-play, was able to consistently win Connect-4 when playing as starting player against Minimax. Given this architecture, some other ingredients are essential to get the n-tuple approach to work in Connect-4:

- *Two* LUTs per n-tuple, one for each player. Using only one LUT, conflicting signals hinder the TDL architecture to learn.

– It is essential to select the right decay scheme for the learning rate α: With too large α values, the agent fails to learn anything, while a too small α slows down the learning process.

Future work. Given the large impact seen in the n-tuple creation process, it is natural to ask whether learning is depending on the characteristics of the n-tuple sets (number, size, and shape of n-tuples). An interesting area of future work is the question whether it is possible to *evolve* n-tuple sets by keeping successful n-tuples in a population and removing bad or redundant ones.

We presented the n-tuple approach to Connect-4 in conjunction with TD learning. It might as well be interesting to compare it with other learning strategies such as co-evolutionary learning (CEL) [7].

We conclude by emphasizing the impressiveness that such a simple and general architecture is able to learn near-perfect decision-making in a complex surrounding *completely from self-play* without a teacher. Here, 'simple' means that no domain-specific knowledge about the tactics of the game is coded, neither explicitly nor implicitly. It is our impression that the mighty feature space, implicitly induced by the large LUTs of the n-tuple approach, plays a large role here.

References

1. Allis, V.: A knowledge-based approach of Connect-4. The game is solved: White wins. Master's thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands (1988)
2. Bledsoe, W.W., Browning, I.: Pattern recognition and reading by machine. In: Proc. Eastern Joint Computer Conference, New York, pp. 225–232 (1959)
3. Curran, D., O'Riordan, C.: Evolving Connect-4 playing neural networks using cultural learning. NUIG-IT-081204, National University of Ireland, Galway (2004)
4. Edelkamp, S., Kissmann, P.: Symbolic classication of general two-player games. Technical report, Technische Universität Dortmund (2008)
5. Konen, W., Bartz–Beielstein, T.: Reinforcement Learning: Insights from Interesting Failures in Parameter Selection. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 478–487. Springer, Heidelberg (2008)
6. Konen, W., Bartz-Beielstein, T.: Reinforcement learning for games: failures and successes – CMA-ES and TDL in comparison. In: Proc. GECCO 2009, Montreal, pp. 2641–2648. ACM, New York (2009)
7. Krawiec, K., Szubert, M.G.: Learning n-tuple networks for Othello by coevolutionary gradient search. In: Proc. GECCO 2011, Dublin, pp. 355–362. ACM, New York (2011)
8. Lucas, S.M.: Learning to play Othello with n-tuple systems. Australian Journal of Intelligent Information Processing 4, 1–20 (2008)
9. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development 3(3), 210–229 (1959)
10. Schneider, M., Garcia Rosa, J.: Neural Connect-4 - a connectionist approach. In: Proc. VII. Brazilian Symposium on Neural Networks, pp. 236–241 (2002)

11. Sommerlund, P.: Artificial neural nets applied to strategic games (1996) (unpublished), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.4690 (last access: June 05, 2012)
12. Stenmark, M.: Synthesizing board evaluation functions for Connect-4 using machine learning techniques. Master's thesis, Østfold University College, Norway (2005)
13. Sutton, R.S.: Temporal Credit Assignment in Reinforcement Learning. PhD thesis, University of Massachusetts, Amherst, MA (1984)
14. Sutton, R.S.: Learning to predict by the method of temporal differences. Machine Learning 3, 9–44 (1988)
15. Tesauro, G.: TD-gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation 6, 215–219 (1994)
16. Thill, M.: Using n-tuple systems with TD learning for strategic board games. CIOP Report 01/12, Cologne University of Applied Science (2012) (in German)

Efficient Sampling and Handling of Variance in Tuning Data Mining Models

Patrick Koch and Wolfgang Konen*

Department of Computer Science, Cologne University of Applied Sciences,
51643 Gummersbach, Germany
{patrick.koch,wolfgang.konen}@fh-koeln.de

Abstract. Computational Intelligence (CI) provides good and robust working solutions for global optimization. CI is especially suited for solving difficult tasks in parameter optimization when the fitness function is noisy. Such situations and fitness landscapes frequently arise in real-world applications like Data Mining (DM). Unfortunately, parameter tuning in DM is computationally expensive and CI-based methods often require lots of function evaluations until they finally converge in good solutions. Earlier studies have shown that surrogate models can lead to a decrease of real function evaluations. However, each function evaluation remains time-consuming. In this paper we investigate if and how the fitness landscape of the parameter space changes, when only fewer observations are used for the model trainings during tuning. A representative study on seven DM tasks shows that the results are nevertheless competitive. On all these tasks, a fraction of 10-15% of the training data is sufficient. With this the computation time can be reduced by a factor of 6-10.

Keywords: Machine learning, parameter tuning, sampling, SVM, sequential parameter optimization.

1 Introduction

Data Mining (DM) is an interesting field for applying Computational Intelligence (CI) techniques. Although CI methods can generate good and robust solutions for global optimization problems, it is known that they sometimes require a large number of function evaluations. Unfortunately, data mining tasks are usually very expensive to evaluate and quick solutions are requested by the users. In this paper we investigate how computation time can be saved in order to make CI methods more applicative for DM tasks.

We claim the following hypotheses:

H1. Tuning results are more subject to noise when smaller fractions X of the training data are used.

* This work has been partially supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grant SOMA ("Ingenieurnachwuchs" 2009).

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 195–205, 2012.

H2. Tuning with smaller fractions X will usually lead to increased prediction errors, but the optimal design points found by a robust optimizer will nevertheless be competitive (as long as X is not *too* small).

If we rewrite **H1** a little, we can come to the conclusion that the variance should be higher when the training set size is smaller. If **H1** holds, we should be able to measure this effect directly by an increased variance of the model error. If we can confirm **H2**, considerable computation speedups in tuning DM models are possible.

Previous work in analyzing the effects of the chosen sample size has been mainly done in fields like statistics and machine learning. A good overview about different strategies to sample data can be found in Cochran [3]. A description of resampling methods for optimization in data mining has been given by Bischl *et al.* [2]. Raudis and Jain [14] and Jain and Zongker [7] discuss the influence of sample sizes on the results of feature selection, which is in a certain way related to the parameter optimization task in this article. In statistics, sample sizes are frequently discussed in terms of statistical studies like significance tests [11]. In machine learning, sampling strategies like the bootstrap [5] have been well analyzed and are often applied in practice. However, to our knowledge no study exists where the size of the underlying training data is diminished during parameter tuning.

2 Methods

2.1 Learning Algorithms

As a learning algorithm we experimented with the Support Vector Machine (SVM) [16], since SVM is known to be very sensitive to its parameter settings. We used the *libsvm* implementation in `R` from the *e1071* package[1]. However, all experiments can be also performed with any other (supervised) machine learning algorithm. Here, we restricted ourselves to SVM, since it appeared to be best suited for our experiments.

SVM needs several hyperparameters to be set to work properly. First of all it requires a kernel function to allow for non-linear class boundaries. The choice of the kernel function is a crucial decision in machine learning and must be considered carefully. However, some kernel functions are good-working for several problems. For instance, the radial basis function (RBF) kernel belongs to the most popular kernel functions and defines the similarity of inputs x and z by

$$k(x, z) = exp(-\gamma(||x - z||)) \tag{1}$$

For our needs the RBF kernel function is well-suited, since it comes along with the hyperparameter γ which has to be set anew for each data. Small values of γ indicate large influences of given data points, whereas large γ values mean

[1] Software available from `http://cran.r-project.org/web/packages/e1071/index.html`

that the influence of the data points is more restricted. Besides the γ kernel parameter SVM regularizes points which do not lie in the hyperplane fitted by the algorithm. Therefore it uses a so-called regularization parameter C (for cost), which is important for finding a good balance of the underlying optimization problem of the SVM and correct classification of training examples. For more details on SVM we refer the interested reader to the literature [15].

2.2 Tuning Algorithms

Parameter tuning tasks in machine learning can be modelled as noisy single-objective optimization problems. Any parameter setting of the learning parameters is called a *design*. For each design we can measure the quality by training a model (running the learning algorithm using some training data) and applying the trained model to new test data. Note that although the underlying learning algorithm may be deterministic, the tuning task can be nevertheless stochastic, because the training and test samples of the data are drawn at random. Hence, the robustness and generalization ability of the optimization algorithm is an important criterion for the tuning task. Konen *et al.* [10] therefore compared tuning algorithms like the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [6] with the Sequential Parameter Optimization Toolbox (SPOT) [1]. SPOT is a heuristic which employs surrogate models of the objective function. The advantage of such strategies is that the number of real function evaluations can be reduced, since parts of the optimization can be performed on the surrogate model. In a comparative study [10], SPOT performed best under strongly limited budgets. It was noted that CMA-ES can give similar results, presuming that enough function evaluations are allowed. Other statistical methods like Latin hypercube sampling (LHD) [12] uniformly distribute design points over the search space: their performance and accuracy usually diminishes for larger search space dimensions.

2.3 Objective Function

We perform a tuning of all parameters which are relevant for our machine learning task. Any single-objective optimizer requires an objective function, in order to evaluate the quality of the parameter designs. In our case we distinguish between

(a) the model error on validation data during tuning, and
(b) the error obtained with the best parameters from tuning on independent test data (data which has not been used throughout the whole tuning process)

During tuning the objective function value is the fraction of wrongly classified pattern in the validation set. The unbiased estimator for the model's error on new data is th fraction of wrongly classified pattern in the test set.

3 Experimental Setup

Parameter optimization in machine learning can be challenging: the training time of SVM often grows quadratically with the number of training patterns [13]. Since this might be one reason why parameter optimization is seldom done in practice, we try to find new ways to make tuning of DM tasks less costly: We vary the number of training patterns used during parameter tuning. In our optimization task the most time consuming part is the evaluation of the objective function, because in each evaluation a complete SVM training is performed (the runtime of optimizers like SPOT can be also expensive, but are neglectable in our case, since we only use fast surrogate models and small designs). We investigate how much time can be saved by reducing the training data used for parameter tuning and if parameters with small training fractions are competitive with parameters tuned on the total training set.

3.1 Datasets

All experiments presented in this study were performed using datasets from the UCI website,[2] which can be regarded as the simpler datasets, the DMC 2007 data from the data mining cup 2007, and a real-world dataset from water ressource management (AppAcid). In Tab. 1 an overview about the datasets and their sizes is given. We present the minimal training set size for each dataset, and also the sizes of the test and validation sets. Each model is evaluated a) during tuning on the validation data and b) after tuning on the independent test data. The sizes for the test and validation sets are equal and stay constant all the time at 20% of the total dataset size. In Tab. 1 these sizes are given as *Validation and Test Size*.

Table 1. Datasets used for the training set size experiments

Dataset	Records	Min. Training Size	Max. Training Size	Validation and Test Size	Number of Parameters
Sonar	208	8	124	41	2
Glass	214	8	128	42	2
Liver	345	13	207	69	2
Ionosphere	351	14	210	70	2
Pima	768	30	460	153	2
AppAcid	4400	176	2640	880	12
DMC-2007	50000	2000	30000	10000	7

Every time a fixed set of 20% of the patterns was set aside prior to tuning for testing purposes. From the remaining 80% of the data (subset D_{train}), we use a fraction X from $X_{min} = 5\%$ to $X_{max} = 75\%$ for training and a fraction of

[2] http://archive.ics.uci.edu/ml/

Table 2. Splitting of data. We use fractions from 4% to 60% of the data for model training, 20% for validation during the tuning and the remaining 20% for independent testing.

Training ⟷	not used	Validation	Test

25% for validation (which is 20% of all data). See Tab. 2 for illustration. At the end of tuning a best design point is returned. Using this best design point, we ran a final 'full' training with all data in D_{train} (80%) and evaluated the trained model on the test set (20%). Since the training, validation and test sets were drawn at random we repeated all of our experiments ten times.

While a first benchmark of tuning algorithms has been performed by Konen *et al.* [10], it remained unclear, if the results also hold for smaller training set sizes. Now we also compare SPOT as a tuning algorithm with LHD as a simple, but robust sampling heuristic. LHD is based on the following procedure: We chose random and roughly equally distributed design points from the region of interest and evaluated the design 3 times. Again, the best point is taken for the 'full' training as above.

3.2 SPOT Setup

SPOT can be controlled through various settings, which have to be adapted slightly for each task. We briefly present our settings for the experimental study here (Tab. 3). With 150 function evaluations for the UCI experiments we chose a rather large number of evaluations compared to the other (real-world) datasets. Our aim was to achieve a good and clear convergence to an optimum. Out of this reason we considered to analyze simpler datasets first, since complex datasets require much more time for such experiments. Nevertheless we also set a number of 200 function evaluations for AppAcid, which proved to be a good and sufficient number in one of our last studies [10]. It has to be noted that the dimensionality of the parameter space for AppAcid is 12, while it is only 2 for the UCI benchmarks.

As region of interest (ROI) for the UCI datasets we set quite large ranges (as we have enough evaluations for these benchmarks). We vary the range of γ between $[0.0, 1.0]$ and the range of cost C between $[0.0, 10.0]$. For the other applications (AppAcid, DMC2007) we relied on the same settings as in our previous experiments, see [10] for more information.

Table 3. SPOT Configuration

Setting	UCI	AppAcid
function evaluations	150	200
initial design size	10	24
sequential design points	3	3
sequential design repeats	$\{1, 3\}$	2

3.3 Sampling Strategies

First of all, k subsets $D_1, ..., D_k \subset D_{train}$ of the complete training data D_{train} lead to models $M_1, .., M_k$ which are presumably not equal when the training data subsets are not equal, although hyperparameters γ and C are identical (on the same training data SVM is deterministic). Thus we have $D_1 \neq D_2 \neq ... \neq D_k \rightarrow M_1 \neq M_2 \neq ... \neq M_k$. Different strategies are possible for sampling the training subsets during tuning:

(A) Choose a subset $D_1 \subset D_{train}$ *once* and use it for the whole parameter optimization process. We call this option *parameter tuning without resampling.*
(B) In each call i of the objective function, choose a new subset $D_i \subset D_{train}$ for training. If a design point is evaluated repeatedly, e. g. n times, we choose different subsets $D_1, D_2, ..., D_n$, train models for them and use an aggregation function (here: the mean) to return an objective function value. We call this option *parameter tuning with resampling.*

4 Results

4.1 Tuning Results

Exemplarily we present boxplots of the SPOT hyperparameter tuning for the Sonar and AppAcid datasets in Fig. 1.[3] We distinguish between the error achieved on the validation data when using only a smaller training set fraction X (*Error VALI Set*) and the independent test set error when a complete re-training with the optimal parameter setting is performed (*Error TEST Set*). All results in Fig. 1 were optimized using SPOT with sampling strategy (B) and a total number of 3 repeated evaluations for each design point. The validation error in both plots is clearly increasing if the training set size X is reduced from $X = 75\%$ to $X = 5\%$. However, the same does not necessarily hold for the test data. While the mean errors and their variances are large for small training fractions, they are roughly constant and small for all $X \geq 15\%$.

This is a promising result, as it may allow us to use only a subsample of the complete data during tuning. As we can see in Tab. 4, good tuning results with a very small number of training data (here X=10%) were observed as well for all other datasets. If we take the best parameters from such a tuning process and re-train *once* with the complete training data, we obtain results on independent test data which are competitive with a much more time-consuming 'full' tuning.

We observed on all considered datasets, that the prediction accuracies of single models on different data are very noisy. Thus, the chosen sampling strategy (see Sec. 3.3) has an impact on the final results. A comparison of the sampling strategies (A) and (B) showed that we can achieve the most stable results using strategy (B) and a number of repeated evaluations greater than 1 (from now on

[3] A complete set of all boxplot results and accompanying material is available for the interested reader from http://gociop.de/about/people/koch/ppsn2012/.

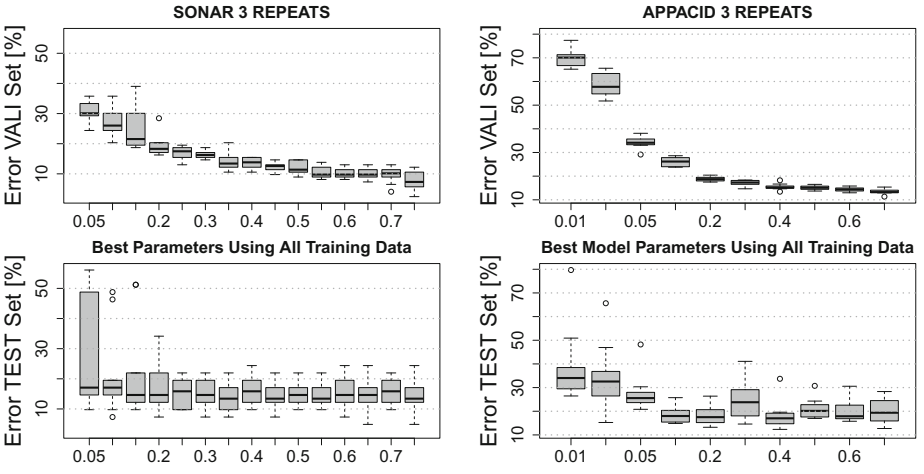


Fig. 1. Tuning results with SPOT for Sonar and AppAcid datasets using 3 repeats. The x-axis shows the training set fraction X.

Table 4. Test-set error rates (mean and standard deviation of ten runs) for all datasets investigated. We show results when using small and full training data during tuning, after re-training once with best parameters found. In case of DMC-2007 we show relative gain instead of error rate.

Dataset	X=10% median (std.dev.)	X=75% median (std.dev.)
Sonar	17.07 (14.3)	13.41 (5.5)
Glass	28.57 (7.7)	28.57 (7.6)
Liver	36.23 (4.3)	31.88 (6.5)
Ionosphere	5.71 (2.4)	5.71 (2.4)
Pima	23.20 (2.5)	23.20 (3.9)
AppAcid	17.99 (3.3)	19.38 (5.5)
DMC-2007	14.73 (2.0)	15.66 (1.0)

we always set the repeats to 3 in our study). We think that repeated evaluations are an important factor to circumvent wrong decisions of the optimizer operating in a noisy environment.

Regarding the comparison of SPOT and LHD we show in Fig. 2 that SPOT usually yields in better parameter settings when trained with a fraction of the data, especially for $X \leq 10\%$. Results degrade for very low training set sizes (1% or 2%). They are however competitive for all $X \geq 10\%$ (SPOT) and $X \geq 20\%$ (LHD): The tuning results are as good as with a tuning on the 'full' training data and the models generalize well on unseen data.

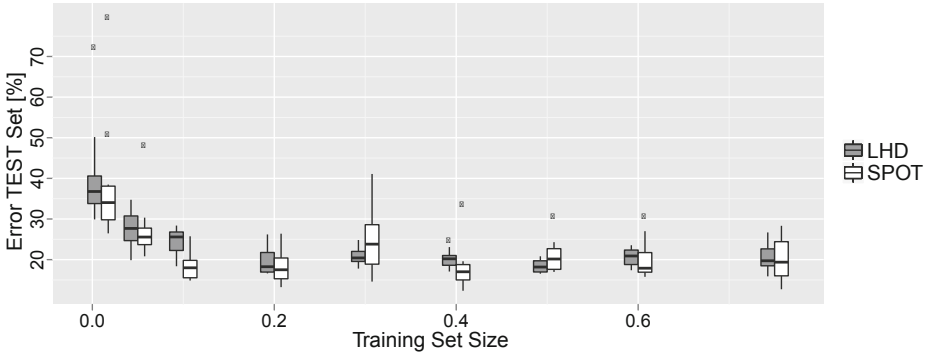


Fig. 2. Comparison of SPOT and LHD algorithms on AppAcid

4.2 Landscape Analysis

Fig. 3 shows a comparison of the surrogate surfaces for a small (X=10%) and large training set size on the Sonar dataset. We used a surrogate model based on Gaussian Processes (GP) [8]. When only few training data were used (left plots), the SPOT and LHD landscapes are both relatively flat, with shallow minima near $\gamma \approx 0$. These minima are however a good indicator for the minima obtained when we perform the tuning with the complete training set size (right plots). These plots both exhibit a clear and deep minimum near $\gamma \approx 0$, relatively independent of the cost term. The landscape for $\gamma > 0$ is however very different. With SPOT, we obtain very spiky surrogate models (Fig. 3, upper right). This especially occurs in regions where only few design points are sampled (these are the regions presumably not containing the optimum). We think that when there is a region with a high density of points but large noise in the objective function, GP assumes a very small correlation length leading to spikes in the low-density regions. Overall this leads to a less robust surrogate model. We will show in a forthcoming contribution [9] that slightly different SPOT initial design settings can lead to instable results.

LHD with its equal-density sampling in design space does not have this problem: The landscape (Fig. 3, lower right) exhibits a low curvature and is stable in all experiments. Nevertheless the main issue of LHD sampling, which is the bad scalability for higher dimensions, remains, and this is the reason why this sampling method is less preferable for higher-dimensional search spaces.

We can conclude that if we use a robust surrogate model and a sufficient initial design size of the tuning algorithm, we can obtain good parameter settings very quickly.

4.3 Computation Time

The characteristics of the computation times for different training set sizes are shown in Fig. 4 for the AppAcid dataset. Note that the curve which appears

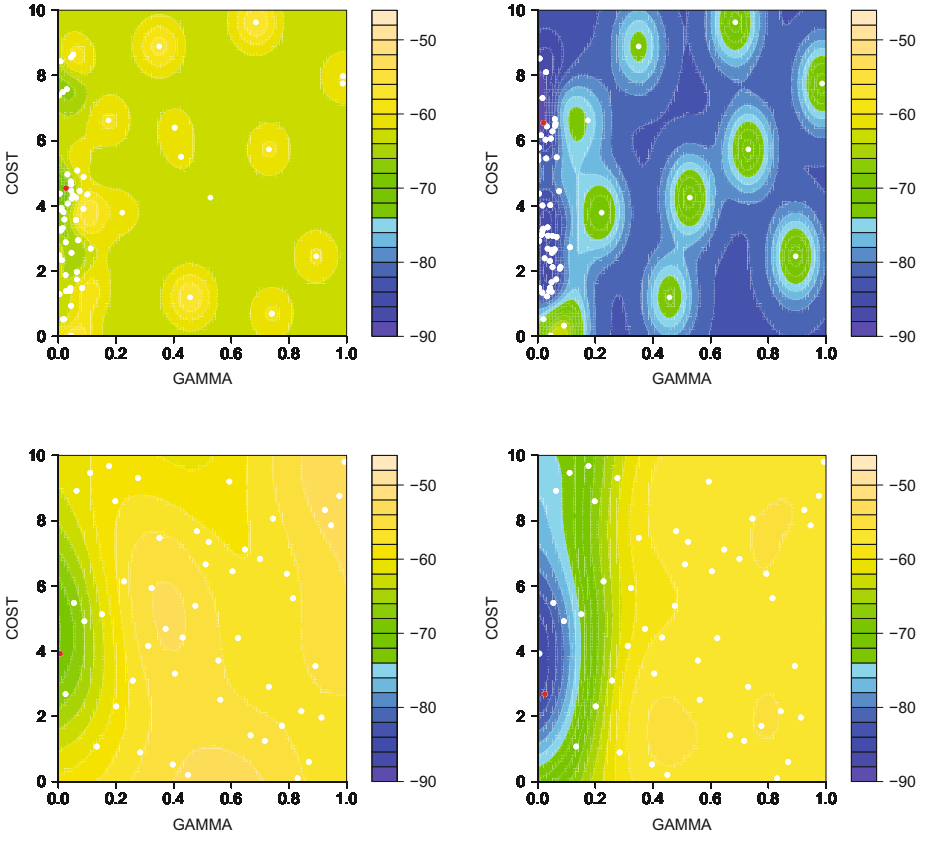


Fig. 3. Contour plots for Sonar. SPOT (top plots) and LHD (bottom plots), using 3 repeats, GP for the contour surface and small (X=10%, left) and large (X=75%, right) training set sizes. White points are design points, the red point is the optimum found by the optimization procedure.

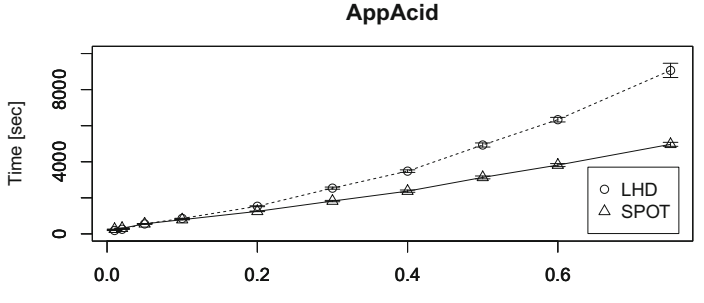


Fig. 4. Computation time on AppAcid as a function of training set size X

to be linear here can have a different look on other datasets. The roughly linear slope has its reason in the model building process for the AppAcid task. This process includes several other operators beneath SVM training, e.g., preprocessing (principal component analysis and feature selection). In other cases the pure SVM training might grow quadratically with the number of training samples, leading to even larger computation time savings.

5 Conclusion

We showed that tuning with a low training set size very often leads to good results: An astonishing small fraction of the training set (10-15%) was sufficient to find with high probability a good DM model when performing one 'full' training with the best design point parameters. The resampling strategy (B) (see Sec. 3.3) might be a crucial ingredient for this success with small training samples, but further research is needed to justify this claim.[4]

This study investigated seven different data sets with sizes from 208 to 50000 records. Especially for the bigger datasets large speedups (factor 6 to 10) are possible as Fig. 4 shows. Summarizing the results, we can conclude that both hypotheses **H1** and **H2** hold for the datasets used in this study. In the future we plan to search strategies for selecting the right sample sizes automatically.

References

1. Bartz-Beielstein, T., Lasarczyk, C., Preuss, M.: The SPO Toolbox. In: Bartz-Beielstein, et al. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 337–360. Springer, Heidelberg (2010)
2. Bischl, B., Mersmann, O., Trautmann, H.: Resampling methods in model validation. In: Proc. WEMACS 2010, Joint to PPSN 2010, Krakow, p. 14 (2010)
3. Cochran, W.G.: Sampling techniques. Wiley-India (2007)
4. Daelemans, W., Hoste, V., De Meulder, F., Naudts, B.: Combined Optimization of Feature Selection and Algorithm Parameters in Machine Learning of Language. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) ECML 2003. LNCS (LNAI), vol. 2837, pp. 84–95. Springer, Heidelberg (2003)
5. Efron, B.: Bootstrap methods: another look at the jackknife. The Annals of Statistics 7(1), 1–26 (1979)
6. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9, 159–195 (2001)
7. Jain, A., Zongker, D.: Feature selection: Evaluation, application, and small sample performance. IEEE Transactions on PAMI 19(2), 153–158 (1997)
8. Karatzoglou, A., Smola, A., Hornik, K., Zeileis, A.: Kernlab — An S4 package for kernel methods in R. Technical report, Department of Statistics and Mathematics, WU Vienna University of Economics and Business, Vienna (2004)
9. Koch, P., Konen, W.: Stability issues in tuning very noisy functions. Submitted to PPSN 2012 Workshop on Automated Selection and Tuning of Algorithms (2012)

[4] We note in passing that Daelemans et al. [4] got negative results with a small training sample, but only on a specific word disambiguation task and without resampling.

10. Konen, W., Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M., Naujoks, B.: Tuned data mining: A benchmark study on different tuners. In: Proc. GECCO 2011, Dublin, pp. 1995–2002. ACM (2011)
11. Lenth, R.V.: Some practical guidelines for effective sample size determination. The American Statistician 55(3), 187–193 (2001)
12. McKay, M.D., Beckman, R.J., Conover, W.J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics, 239–245 (1979)
13. Osuna, E., Freund, R., Girosi, F.: Training support vector machines: an application to face detection. In: IEEE Proc. CVPR 1997, pp. 130–136 (1997)
14. Raudys, S.J., Jain, A.K.: Small sample size effects in statistical pattern recognition. IEEE Transactions on PAMI 13(3), 252–264 (1991)
15. Schölkopf, B., Smola, A.J.: Learning with kernels: Support vector machines, regularization, optimization, and beyond. The MIT Press (2002)
16. Vapnik, V.: Statistical learning theory (1998)

A Spatial EA Framework for Parallelizing Machine Learning Methods

Uday Kamath[1], Johan Kaers[2], Amarda Shehu[1], and Kenneth A. De Jong[1]

[1] George Mason University, Fairfax VA 22003, USA
{ukamath,ashehu,kdejong}@gmu.edu
[2] Shaman Research, Heverlee 3001, Belgium
johankaers@telenet.be

Abstract. The scalability of machine learning (ML) algorithms has become increasingly important due to the ever increasing size of datasets and increasing complexity of the models induced. Standard approaches for dealing with this issue generally involve developing parallel and distributed versions of the ML algorithms and/or reducing the dataset sizes via sampling techniques. In this paper we describe an alternative approach that combines features of spatially-structured evolutionary algorithms (SSEAs) with the well-known machine learning techniques of ensemble learning and boosting. The result is a powerful and robust framework for parallelizing ML methods in a way that does not require changes to the ML methods. We first describe the framework and illustrate its behavior on a simple synthetic problem, and then evaluate its scalability and robustness using several different ML methods on a set of benchmark problems from the UC Irvine ML database.

Keywords: Spatially-structured evolutionary algorithms, machine learning, ensemble learning, boosting.

1 Introduction

The most common applications of machine learning involve supervised learning in which a training set of labeled examples (or instances) is used to learn a model that can be subsequently used to make predictions about previously unseen examples. The scalability of such ML algorithms has become increasingly important as datasets become larger and the complexity of the induced models increases. Many current ML techniques scale poorly either because they require the entire training set to be in memory simultaneously, or because the running time of the model induction code grows non-linearly with the size of the training data, or both [1]. Basic solutions like reducing the size of the training datasets via sampling can be used but can introduce sampling errors. ML boosting techniques are designed to deal with hard-to-classify examples, but do so by making multiple passes over the training data [2]. More complex approaches involve changing the basic structure of ML methods into parallel and distributed versions.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 206–215, 2012.

In this paper we describe an alternative approach that combines features of spatially-structured evolutionary algorithms (SSEAs) with the well-known machine learning techniques of ensemble learning and boosting. The result is a powerful and robust framework for parallelizing ML methods in a way that does not require changes to the underlying ML methods. We refer to this as our PSBML framework, shorthand for Parallel Spatial Boosting Machine Learning. We first describe our framework and illustrate its behavior on a simple synthetic problem. We then evaluate its scalability and robustness using several different standard ML methods on a selected subset of the benchmark problems in the UC Irvine ML database.

2 Related Work

As noted above, our framework combines features from both the evolutionary computing (EC) and ML communities. In this section we briefly summarize them.

Spatially-structured evolutionary algorithms (SSEAs) which use topologically distributed populations and local neighborhood selection have been well analyzed in the EC literature [3]. SSEAs have been shown to maintain a diverse set of better individuals longer, resulting in improved performance in many applications [4]. However, the key feature that we want to take advantage of is its "embarrassingly parallel" architecture in that at each topological grid point a local algorithm is running that has only local interactions with its immediate neighbors. In that sense our approach has much in common with cellular EAs and cellular automata.

One of the interesting ML developments is that a collection (ensemble) of simpler classifiers can often be more accurate than a single more complex classifier, and of course much easier to parallelize [5]. This maps nicely onto SSEAs in the sense that an ensemble of classifiers can be distributed across the topological grid points in an SSEA. However, standard ensemble techniques require each classifier to look at the entire (possibly sampled) set of training data. The hypothesis we are exploring is that, by distributing the training data across the grid points as well, the emergent capability of an ensemble of ML classifiers that only have access to local subsets of training data will be comparable in classification performance to that of standard ML techniques and significantly more scalable via parallelization.

A second interesting ML development is the awareness of the important distinction between easy and hard training examples. Intuitively, the hard examples are those close to classification decision boundaries. A classic example of these are the "support vectors" on which support vector ML techniques are based. Since the decision boundaries are not known *a priori*, in general, multiple passes over the training data are required to identify these examples often via "boosting" techniques that increase the frequency and/or weights of such training instances [2]. Beyond just improving classification accuracy, the identification of these difficult training examples on the boundaries often leads to additional problem insights used for finding interesting features for classification [6].

Our PSBML framework incorporates this idea as well by introducing a local neighborhood notion of hardness, using it as a measure of fitness, and boosting the harder instances via fitness-proportional selection. The result is a parallel ML technique in which both classification accuracy and the identification of hard instances improves as the system evolves.

3 The PSBML Framework

Our PSBML framework for parallelizing machine learning methods has at its core an SSEA [3] in which individuals and algorithms are distributed over a two-dimensional torroidal grid with a common algorithm running locally on each node in the grid and only local interactions with nearby grid points. In our case the common algorithm is a replicator EA (i.e., no reproductive variation) that is manipulating a population of training examples. Selection pressure in an SSEA is determined by two design choices: the selection method used by the local EAs running on each grid point, and the size and shape of the neighborhood structure. In the experiments described in this paper, the local EA selection method used was fitness-proportional selection (our boosting technique). We did, however, experiment with different standard cellular neighborhood structures (Fig. 1) in order to study the effects of varying the overall selection pressure.

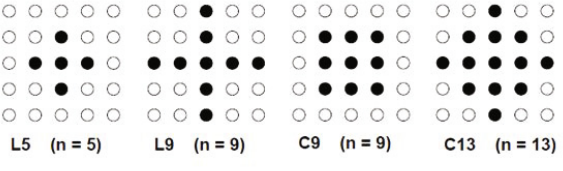


Fig. 1. 2D-grid with various neighborhood structures (Source: LNCS 1141, p 237) [3]

Each node in the grid has a local EA running maintaining a population of ML training examples that gets updated each generational cycle. The fitness of each training example in the population is assessed via is a local ML technique (e.g., a naive Bayesian classifier, a decision tree learner, etc.) in the following manner. On each generational cycle, a node performs a standard ML train-test procedure using the training examples on its node for training, and using the training examples on neighboring nodes for testing. As is the case with standard ML boosting methods, in addition to classifying the test examples, the learners output a confidence value for each decision. The confidence values are used to assign a fitness to each of the neighborhood test examples, allowing the local replicator EA to subsequently select (boost) the more difficult examples. Since each member of the overall set of training data is a member of the neighborhood of multiple local ML methods, the result is an ensemble assessment of difficulty similar to the classification margin concept used in boosting [2]; namely, the smallest confidence from any node, for any class is taken as the fitness w of the instance.

$$w = \min_{i \in class} (\min_{n \in neighbor} c_{ni})$$

Experimentally, we determined that a non-overlapping-generation model for the local EAs was much less effective than an overlapping one, in which only a fraction of the local population was replaced each generational cycle. We implemented this feature through a replacement probability parameter p_r, and found that values around 0.20 were most effective (i.e., replacing about 20% each generation). See section 4.3 for more details.

The overall pseudo-code of PSBML is as follows:

- Initialization: distribute the training dataset uniformly over all the nodes in the grid.
- For every EA generation:
  - On each node:
    - Use the local ML technique and the current local population of training examples to produce a candidate classifier.
    - Test this classifier on all the population members in the neighboring nodes, assigning confidence values to each population member.
    - Create a selection pool consisting of all population members of the node and its neighbor nodes.
    - For each member in the current node population, replace it with probability p_r with an individual from the selection pool using fitness-proportional selection.

4 Analysis of the PSBML Framework

As stated in the introduction, the hypothesis for this research is that the emergence behavior of ML techniques embedded locally in this spatially distributed framework will be comparable in classification performance to the corresponding monolithic ML versions with the significant additional benefit of significant improvements in scalability via parallelization. The two important emergent properties are the effects of local boosting and local classifier training and testing. We analyze both effects in this section.

Since local boosting is done by a replicator EA using fitness-proportional selection from a pool that includes neighboring populations, the formal analysis is identical to that of how the emergent selection pressure in SSEAs changes as a function of the neighborhood topology [3,7] as illustrated in Figure 2. In this case, increased selection pressure corresponds to increased boosting rates. As with standard boosting techniques, one must find a growth rate in PSBML that facilities the learning process by gradually propagating the more difficult training examples throughout the grid via evolutionary boosting.

The second emergent property is the overall classification accuracy of PSBML. Unless its local boosting and training elements result in an ensemble performance comparable to monolithic ML techniques, there is no virtue in PSBML's scalability.

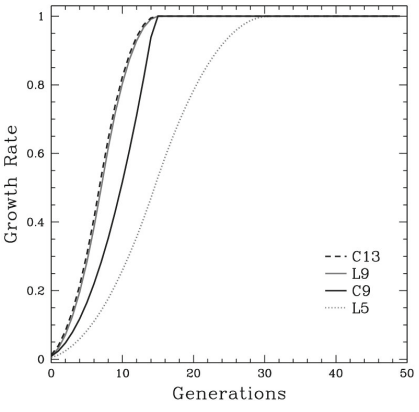


Fig. 2. Growth curves for C13, L9, C9 and L5 neighborhoods

In the following sections we describe an initial set of experiments to assess both of these emergent properties. We start with a series of tuning experiments to obtain a rough estimate of the more important PSBML design parameters that affect these emergent properties. Then, using this as the default PSBML configuration, we evaluate its performance on a set of standard ML benchmark problems and assess the robustness of the approach using a variety of standard ML methods.

4.1 Experiment 1: A Simple Circle Classification Problem

As a first step in analyzing the behavior of PSBML, we designed a simple synthetic ML problem to illustrate its behavior. The underlying binary classification problem was a 2-dimensional space in which points inside a circle centered at the origin were designated as negative examples and the rest as positive examples. In this case, a simple ML learner is trying to infer the radius of the circle from the training examples it is given by choosing a radius equal to the average distance from the origin of the largest negative example and the smallest positive example. Classification confidence is then based on the distance of an instance from the edge of the circle with the hypothesized radius.

Figure 3 illustrates the results involving a circle of radius 0.4. The underlying spatial topology was a 5x5 torroidal grid in which 10,000 sample points are equally distributed over the 25 grid nodes. In these experiments, the C9 neighborhood structure was used. We ran the PSBML framework on this setup for 100 generations and collected two pieces of behavioral data: the average distance of the instances from the origin, and the number of distinct instances over all nodes. As hypothesized, the emergent global behavior of PSBML was to steadily reduce the number of distinct training instances to a subset that was "on the margin", i.e. close to the decision boundary of 0.4 and comparable to single margin-based classifiers like SVMs.

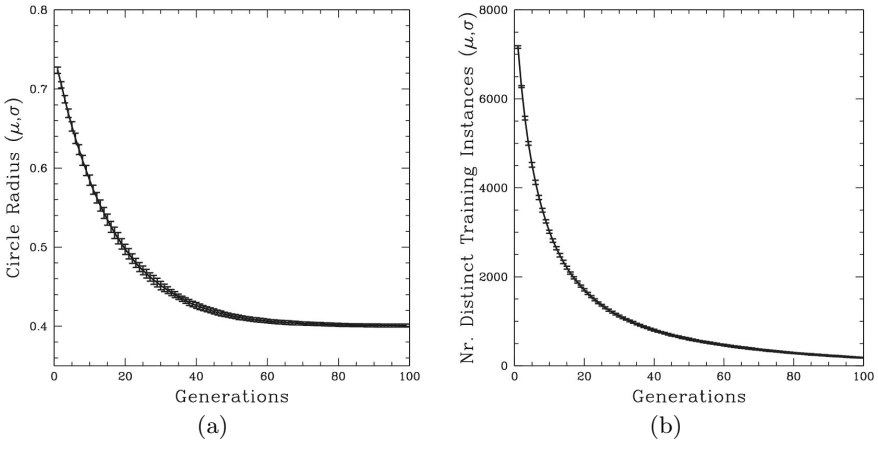


Fig. 3. (a) Mean values with 95% confidence intervals from 30 independent PSBML runs. (b) The number of distinct training distances decreases with the generations.

4.2 Experiment 2: Neighborhood Effects

The next step was to study the effects that SSEA neighborhood structure has on the performance of PSBML. We chose the UCI Chess (King-Rook vs. King-Pawn) dataset for these experiments. It has 3196 instances, 36 attributes and 2 classes. We ran PSBML on this problem using various neighborhood structures as shown in Fig. 4(a). We used a 5X5 grid with a naive Bayesian classifier as the ML method with discretization for numeric features. The ensemble classifier is evaluated by combining the reduced datasets from all the nodes, training a single classifier with these and comparing the test set predictions for classification accuracy or the error-rate. Although the average reduction in the training data was quite similar for all the neighborhoods, their classic "over fitting curves" were different. The stronger selection pressures of L9 and C13 produced the more rapid initial decrease in test classification error rates, which subsequently increased more rapidly as the training data became too sparse. The simplest L5 neighborhood reduced classification error rates too slowly. The best results were obtained with C9.

4.3 Experiment 3: Impact of p_r

In this set of experiments, we ran PSBML on the UCI Chess dataset with different p_r values to observe the effect that different rates of replacement have on the performance of PSBML. Figure 4(b)-(c) illustrates that increasing p_r results in faster convergence but a less accurate learner, with the best results obtained when p_r is about 0.2.

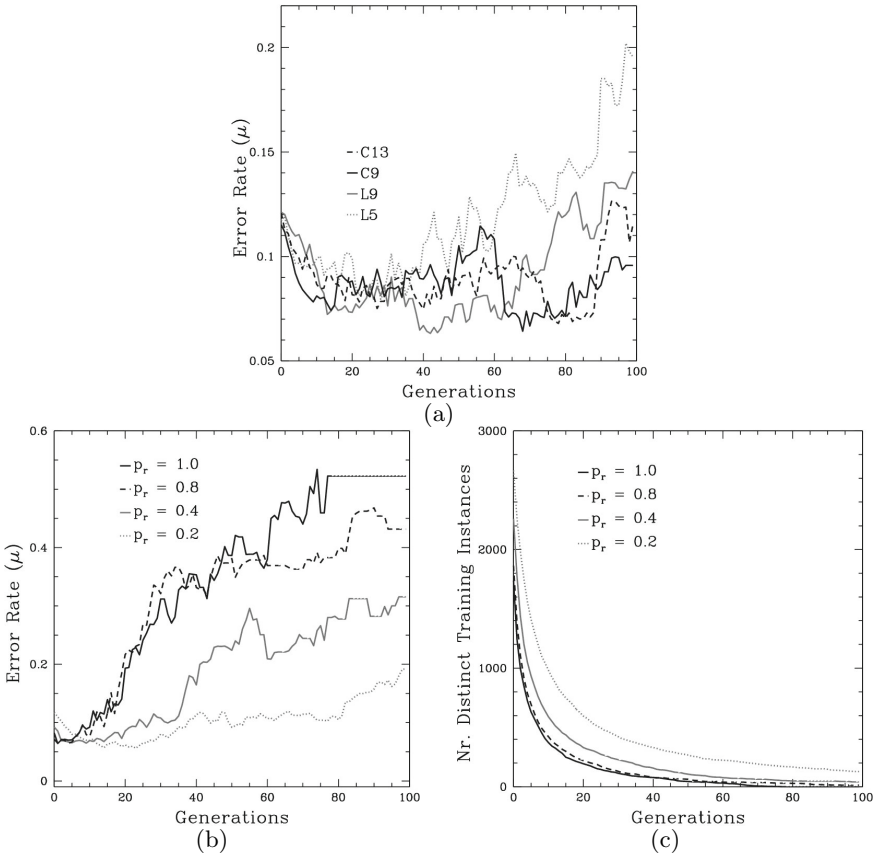


Fig. 4. Results are shown on the UCI chess dataset. The error rate is shown as a function of the neighborhood structure in (a) and probability p_r in (b). The number of distinct training instances is shown as a function of p_r in (c).

4.4 Experimental Analyses on Benchmark Datasets

Using the rough tuning parameters of the previous sections, we evaluated the performance of PSBML on nine classification problems with medium-to-large datasets from the UCI ML repository [8]. The datasets are shown in Table 1 in terms of the number of training instances, number of testing instances, number of features, and number of classes.

We employed a 5x5 grid with C9 neighborhood configuration and p_r of 0.2. To evaluate the robustness and meta learning capability, we tested PSBML with 3 standard ML methods: a naive Bayesian (NB) classifier, a decision tree (DT) learner and a support vector machine (SVM), each employed with the standard implementations available in Weka [9]. The classification accuracy obtained by each ML method is compared to the classification accuracy obtained when embedding that method within PSBML. Results are shown in Table 2. Rows labeled

Table 1. UCI Benchmark datasets

Dataset	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
#Train	3196	4600	10992	19020	32560	49749	271617	581012	4000000
#Test	319	460	1099	1902	16279	14951	59535	58102	311000
#Feat	36	57	256	10	14	300	8	54	42
#Class	2	2	10	2	2	2	2	7	24

"#Hard" show the reduced size of the data set resulting from the evolutionary boosting in PSBML. All reported results are averages over 30 runs (ceiling values reported for "#Hard"). The standard deviation for most runs was below 0.1 and so is not shown. Fields labeled NA correspond to experiments that could not be performed due to algorithmic constraints or very long training times required by the base classifier. Comparisons between methods are done by performing 30 runs and using t-tests for statistical significance with 95% confidence intervals. Runs that show improvements are highlighted in bold.

Table 2. Comparison of PSBML with NB, DT, and SVM on UCI Benchmark datasets

Dataset	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
NB	88.32	79.52	84.41	78.21	83.19	96.7	78.11	79.15	98.89
PNB	**93**	**94**	**90**	**83.1**	**89.01**	**98.1**	**90.01**	**85.1**	**99.65**
#Hard	191	752	484	4544	5625	7234	9157	47234	45034
DT	99.65	97.17	79.51	85.49	85.83	NA	95.12	NA	NA
PDT	99.64	96.1	**80.12**	**86.1**	85.61	NA	**96.34**	NA	NA
#Hard	2678	2667	302	7699	9163	NA	45001	NA	NA
SVM	96.24	90.76	87.97	79.33	85.26	NA	93.9	NA	NA
PSVM	**97.1**	78	**88.45**	**80.12**	**86.1**	NA	84.1	NA	NA
#Hard	2001	3078	1297	3715	23409	NA	47234	NA	NA

Recent research has shown that parallelizing boosting algorithms results in efficient learning [10,11]. So we also compared the performance of PSBML to the ensemble-based meta-learners AdaBoost and ParallelBoost. Table 3 summarizes the result using NB as the base classifier.

The column labeled PNB shows the classification accuracies obtained by PSBML running an NB classifier, the column labeled AB-NB shows the classification accuracies obtained when employing AdaBoost with NB, and the column labeled PB-NB shows the classification accuracies obtained when employing ParallelBoost with NB. Again we see that PSBML produces comparable or better results.

Table 3. Comparison of PSBML with AdaBoost and ParallelBoost

Dataset	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
AB-NB	92.83	93.89	86.9	83.22	85.13	97.45	92.43	78.72	99.14
PB-NB	92.9	93.8	77.21	83.9	85.7	97.9	93.21	82.1	99.18
PNB	**93**	**94**	**90**	83.1	**89.01**	**98.1**	90.01	**85.1**	**99.65**

4.5 Scalability Experiments

The experiments of the previous section support the hypothesis that PSBML achieves comparable classification in comparison with other single classifiers, while providing a significant scalability potential via parallel local learning on local subsets of training data. These experiments were run on single machines using single computation threads. The next obvious step is a systematic study of the scalability of PSBML on a variety of parallel and distributed computational architectures. In general, although cellular models can map onto loosely-coupled Beowulf-style clusters, a better fit is a multi-threaded shared memory architecture with each local EA running on a separate thread.

A principled port of PSBML to our GPU server environment is in progress. To date, our multi-threading experiments consist of measuring PSBML speedup on a single machine with multiple cores and multi-threading support. As an example, Fig. 5 shows the running time in milliseconds of PSBML as a function of the number of threads employed, suggesting there is indeed significant potential for speedup and hence scalability. These particular results were obtained running under Linux OS on a 2GHz 2x4 core Intel machine.

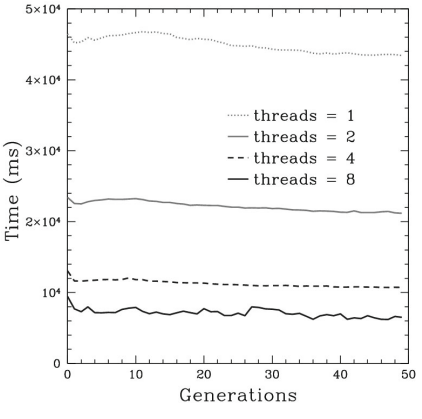


Fig. 5. Training time when using 1, 2, 4, and 8 threads

5 Conclusion and Future Work

We have described a novel approach for parallelizing machine learning methods that combines the features of spatially-structured evolutionary algorithms with the well-known machine learning techniques of ensemble learning and boosting. It does so in a way that does not require changes to the underlying machine learning methods, maintains or improves classification accuracy, and can achieve significant speedup in running times via a straightforward mapping to multi-threaded shared-memory architectures.

Although our experiments to date have been on machines that have significantly fewer parallel threads than the number of grid points of the underlying

SSEA, we plan to continue our evaluation of PSBML in the context of a GPU server environment in which this is not the case.

We are also exploring the use of PSBML as the first stage of multi-stage experiments in which subsequent stages take advantage of the reduction of the dataset to both a more manageable size and containing the most critical exemplars.

References

1. Bordes, A., Bottou, L., Gallinari, P.: Sgd-qn: Careful quasi-newton stochastic gradient descent. Journal of Machine Learning Research 10, 1737–1754 (2009)
2. Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the margin: A new explanation for the effectiveness of voting methods (1997)
3. Sarma, J., De Jong, K.: An Analysis of the Effects of Neighborhood Size and Shape on Local Selection Algorithms. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN IV. LNCS, vol. 1141, pp. 236–244. Springer, Heidelberg (1996)
4. Tomassini, M.: Spatially structured evolutionary algorithms: artificial evolution in space and time. Natural computing series. Springer (2005)
5. Opitz, D., Maclin, R.: Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research 11, 169–198 (1999)
6. Pamuk, B., Can, T.: Coevolution based prediction of protein-protein interactions with reduced training data. In: 2010 5th International Symposium on Health Informatics and Bioinformatics (HIBIT), pp. 187–193 (April 2010)
7. Banks, R.B.: Growth and Diffusion Phenomena: Mathematical Frameworks and Applications. Springer (1993)
8. Asuncion, A., Newman, D.J.: UCI machine learning repository (2007)
9. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. 11(1), 10–18 (2009)
10. Yu, C., Skillicorn, D.B.: Parallelizing boosting and bagging (2001)
11. Favre, B., Hakkani-Tür, D., Cuendet, S.: Icsiboost (2007), http://code.google.come/p/icsiboost

Competing Mutating Agents for Bayesian Network Structure Learning

Olivier Regnier-Coudert and John McCall

IDEAS Research Institute, Robert Gordon University, Aberdeen, UK
{o.regnier-coudert,j.mccall}@rgu.ac.uk

Abstract. Search and score techniques have been widely applied to the problem of learning Bayesian Networks (BNs) from data. Many implementations focus on finding an ordering of variables from which edges can be inferred. Although varying across data, most search spaces for such tasks exhibit many optima and plateaus. Such characteristics represent a trap for population-based algorithms as the diversity decreases and the search converges prematurely. In this paper, we study the impact of a distance mutation operator and propose a novel method using a population of agents that mutate their solutions according to their respective positions in the population. Experiments on a set of benchmark BNs confirm that diversity is maintained throughout the search. The proposed technique shows improvement on most of the datasets by obtaining BNs of similar of higher quality than those obtained by Genetic Algorithm methods.

Keywords: Bayesian network, permutations, distance mutation, genetic algorithm, island model, diversity maintainance.

1 Introduction

Bayesian Networks (BN) are probabilistic graphical models composed of a Directed Acyclic Graph (DAG) and a parameter set. Combination of both allows factorization of the joint probability distribution P of a set of variables X_i, according to their respective parents $Pa(X_i)$ in the DAG as expressed in (1).

$$P = P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)) \tag{1}$$

Learning BN structure from data is a NP-hard problem and thus a challenging task for the machine learning community. The number of possible DAGs that can be drawn grows super-exponentially with the number of variables and is quantified as $O(n!2^{\frac{n}{2}})$. Typically, two families of methods are considered for BN structure learning, respectively based on conditional independence tests and on search and score approaches. While conditional independent methods focus on determining relationships that exist between variables using statistical tests, this paper makes use of the search and score approach. Solutions are generated and

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 216–225, 2012.

scored against a fitness function and are improved until reaching a satisfactory level. In recent years, nature-inspired meta-heuristics have proved successful in efficiently learning structures from data on various problems whether using Genetic Algorithms (GA) [1], Island Models (IM) [2], Ant Colony Optimization (ACO) [3] or Particle Swarm Optimization [4]. Yet, most approaches suffer from the multi-optimal and plateaued nature of the search space, which leads to consequent loss of diversity within the set of solutions and early convergence [2]. In this paper, a novel technique is presented that takes advantage of the permutation representation of the solutions. By considering a population of agents and by assigning different roles to each according to their positions in the population, diversity is maintained throughout the search.

The paper is structured as follows. In sections 2 and 3, we respectively provide background information on BN structure learning and describe a novel approach based on agents and mutation. The experimental approach is introduced in section 4. We finally present and discuss the findings in section 5.

2 Background

2.1 Search and Score Bayesian Network Structure Learning

In recent years, K2GA [5] has been used as a reference for comparison with many algorithms [1,3]. K2GA uses the greedy K2 [6] algorithm to score solutions within a GA framework. Scoring solutions using K2 is not singular to K2GA and it has been extended to other metaheuristics such as ACO [3] or IM GA [2]. In all K2-based methods, solutions are represented as permutations denoting variable orderings. Given an ordering, K2 only allows a variable to be parent of another variable if the latter is at a further position. This constraint ensures that no cycle is found in solutions produced by K2, hence, that all structures are DAGs. In K2, each network is evaluated using the CH score given in (2), which is a measure of the likelihood of a particular structure to represent the data [7]. q_i represents the number of possible different combinations the parents of the node X_i can take and r_i, the arrity of X_i, that is its number of possible states. From a dataset D, the CH score of the BN structure B_s takes into consideration for each of the n variables the number N_{ijk} of instances where X_i is set to its k-th state while its parents are in their j-th state. N_{ij} is the sum of N_{ijk} over the different states of X_i. K2 starts by assuming that all nodes are independent. Edges are added one at a time and kept if the score of the network is improved.

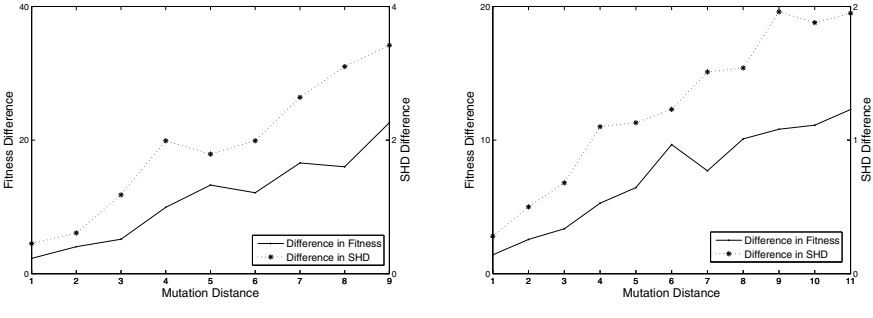
$$P(B_s, D) = P(B_s) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{2}$$

2.2 Diversity Enhancements

In [2], IMK2GA, an IM version of K2GA, is compared to K2GA on five benchmarks. In IMK2GA, several evolutions are run in parallel and paused in order to

allow exchange of solutions. The study highlights difficulties for K2GA to avoid early convergence. By migrating solutions between populations, IMK2GA is able to increase the diversity in each population and prevent the search to stop. In addition, the structures learned using IMK2GA were overall better than those obtained by K2GA, illustrating how local optima can be brought together.

In [8], distance mutation (DM), an operator for variable orderings, is introduced. Several Evolutionary Algorithms (EAs) are implemented and tested against each other on benchmark data. Results suggest that DM can be used as the only operator to produce new solutions within EAs and that crossover does not help when used in conjunction of DM. A common way to perform mutation in permutation representations is to swap two genes of an individual. However, due to the ordered nature of the representation and on K2 behavior, the distance between two variables being swapped in an ordering has an influence on how different the mutated solution is from its predecessor. Mutation is often seen as a local change in GA. With regards to BN structure learning, this statement is true when adjacent variables are mutated. However, the locality breaks down as the distance between two swapped variables increases. Structural Hamming Distance (SHD) can be measured to evaluate differences between two BN structures. In order to observe the effect of the distance parameter in the mutation of orderings, 100 random solutions were generated and mutated once for each distance, that is one gene is selected at random in the ordering and is swapped with another at a position related to the mutation distance. Differences in fitness and SHD to the true structure were recorded for the networks obtained after running K2 on these orderings. These are plotted in Figure 1 for two selected benchmarks and show that changes in the BN structures are more important when the distance is increased. In [8], mutation distance varies along the search in order to cope with the loss of diversity in the population and allow exploration in early generations and exploitation in later ones.



(a) Car **(b)** Boerlage

Fig. 1. Effect of mutation distance on mutated BNs

3 Competing Mutating Agents Using Distance Mutation

3.1 Competing Mutating Agents

Since early convergence is an issue for many methods, a new algorithm has been implemented that aims at performing both exploratory and exploitative tasks throughout the search. A population of agents is considered in which each agent aims at improving its assigned solution by means of DM. The use of adaptive mutation is not a new concept in EAs as has been seen with mutation rate in previous works including [9]. In this paper, DM distances differ between agents and are set according to the agents' positions in the population. Large distances are allowed for agents in low positions while best agents, that is those with the highest quality solutions, are constrained with smaller distances. Since each agent only uses mutation to improve its solution and because each agent aims at reaching the best positions in the population, we call this approach COMpeting Mutating Agents (COMMA). Algorithm 1 presents the outline of COMMA used for maximization optimization. For each position pos_j in the population pop sorted in ascending order, a DM distance d_j is set such that for two agents at positions e and f, $d_e \leq d_f$ if $e < f$. Since it can be beneficial to allow degrading solutions to be accepted, as seen in simulated annealing [10], a probability p_j is also set for each pos_j. Each agent a_i is initially assigned a random solution s_i. The population is then sorted by fitness. At each generation, each agent mutates s_i using the distance $dist_i \in [1, d_r]$ defined according to its position r in the population. If the mutated solution s_{new} has a better fitness than s_i, a_i replaces s_i with s_{new}. If s_{new} has a poorer fitness than s_i, s_{new} only replaces s_i with probability p_r.

Algorithm 1. *COMMA*

Initialize pop of σ agents with random solutions, distance vector d of size σ and probability vector p of size σ
repeat
  Sort pop by fitness in ascending order
  for each agent $a_i, i \in [0, \sigma - 1]$ **do**
    Get position r of a_i in pop
    Generate new solution s_{new} with fitness fit_{new} by mutating s_i with distance $dist_i$ selected with uniform probability from $[1, d_r]$
    if $fit_{new} > fit_i$ **then**
      Assign $s_i = s_{new}$
    else
      Assign $s_i = s_{new}$ with probability p_r
    end if
  end for
until Stopping condition met

3.2 Island Model Competing Mutating Agents

The use of IM showed improvements over its serial counterparts when learning BN structures [2] by maintaining diversity. Hence, IM was also implemented for COMMA and is referred to as IM-COMMA. The outline of IM-COMMA is presented in Algorithm 2 for n islands, m migrations and migration intervals of size $migInterval$. The search is split into evolution stages stg_l where $migInterval$ generations are performed. At each stg_l, the COMMA process is performed and paused to allow migration. Solutions from the ρ best agents from each island are sent to the neighbouring island at migration and assigned to the ρ worse agents in its population following a ring topology with a best-worse policy.

Algorithm 2. $IM - COMMA$

```
Initialize k populations of σ agents with random solutions, distance vector d of size
σ and probability vector p of size σ
for each evolution stage stg_l, l ∈ [0, m − 1] do
  for each island isl_k, k ∈ [0, n − 1] do
    gen_k = 0
    repeat
      Sort pop_k by fitness in ascending order
      for each agent a_i, i ∈ [0, σ − 1] do
        Get position r of a_i in pop_k
        Generate new solution s_new with fitness fit_new by mutating s_i with dis-
        tance dist_i selected with uniform probability from [1, d_r]
        if fit_new > fit_i then
          Assign s_i = s_new
        else
          Assign s_i = s_new with probability p_r
        end if
      end for
      gen_k + +
    until gen_k = migInterval
  end for
  if l ≠ m then
    Select subpopulation mig_k of size ρ for migration
    if i ≠ 0 then
      Replace ρ worse orderings in pop_k by mig_{i−1}
    else
      Replace ρ worse orderings in pop_k by mig_{n−1}
    end if
  end if
end for
```

4 Experimental Approach

In order to evaluate the abilities of the different methods in learning BN structures we selected some known benchmark BNs from which we sampled datasets.

These include *asia* [11], *tank* [12], *credit* [12], *car* [13] and *boerlage* [14]. Characteristics of the benchmarks differ and are summarized in Table 1. K2GA and IMK2GA, were set following [2]. IMK2GA was set with 4 islands and 3 migrations of size $\rho = 2$. Population sizes in K2GA and IMK2GA were set with similar values as described in Table 1. Migration intervals for IMK2GA were set in a way that all migrations occur within a maximum number of 1000 individual fitness evaluations (FEs). Two versions of COMMA and IM-COMMA were implemented in order to observe the effect of degradation. We set one of the COMMA and one of the IM-COMMA with degradation probabilities equal to zero while the two other versions were set with degradation probabilities of [0.2, 0.2, 0.4, 0.4, 0.6, 0.6, 0.6, 0.8, 0.8, 0.8], respectively stated from the best to the worse position in the population of agents. For the time of the experiments, we respectively call these methods $COMMA$, $IM-COMMA$, $COMMA_d$ and $IM-COMMA_d$ where d stands for degradation. As the population size for $COMMA$ and $COMMA_d$ was set to 10, using 4 distinct DM distances and probabilities helped observing how agents evolve. DM distances were set relative to the number of nodes in each benchmark. $IM-COMMA$ and $IM-COMMA_d$ were both set with 4 islands, 3 migrations and 7 generations were chosen as migration interval in order to reach 1000 FEs.

In order to assess the behavior and efficiency of the algorithms, measurements are taken at each generation. Fitness of the best solution is measured, as it helps understanding how the algorithms converge. Kendall Tau Distance (KTD) in the population is also calculated in order to evaluate diversity. The KTD is obtained by averaging the KTD between every pair of orderings in the population. The approach is described in [2]. In addition, it has been shown that increase in the fitness value does not always correlate with improvement in the actual BN structure. Since the true structure of the BNs is known, it is possible to measure the number of correct (C), reversed (R), added (A) and omitted (O) edges of a solution. Based on C, R, A and O, several metrics can be used to describe the distance of a solution to an optimal structure [15]. In the current paper, algorithms are compared according to their respective SHD, representing the number of changes needed to be performed to retrieve the true structure, that is the sum of R, A and O. In addition, R is often regarded as an approximation of low importance. For this reason, the number or relevant edges, that is C + R, is also expressed. On the other hand, it is important not to omit edges, nor to add

Table 1. Dataset characteristics and algorithm settings

Dataset	**Nodes /Edges**	**K2GA pop**	**IMK2GA migInterval**	**COMMA pop**	**mutation distances**
Asia	8/8	100	150	10	1,1,2,2,3,3,3,4,4,4
Tank	14/20	50	250	10	1,1,3,3,5,5,5,7,7,7
Credit	12/12	50	300	10	1,1,2,2,4,4,4,6,6,6
Car	18/17	20	1200	10	1,1,3,3,6,6,6,9,9,9
Boerlage	23/36	20	800	10	1,1,3,3,7,7,7,11,11,11

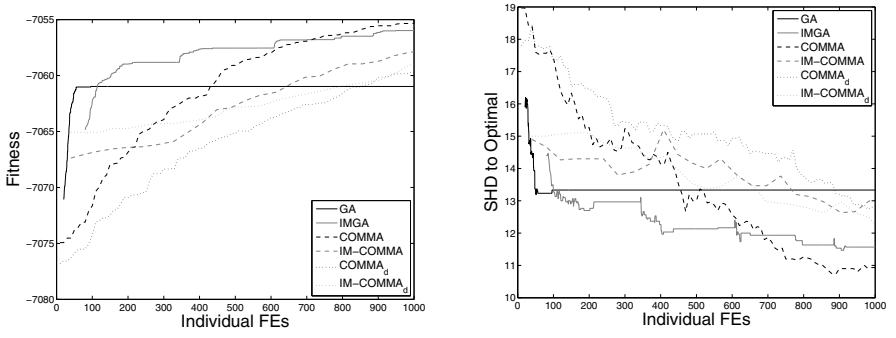
spurious edges to the structure. Consequently, erroneous edges are measured as A + 0. Finally, a greyscale representation was used to map agents' positions in the population over time and illustrate the state of convergence of the method.

5 Results and Discussion

For brevity only a sample of all results is presented although similar patterns are observable across benchmarks[1]. Table 2 summarizes the empirical results in terms of C, SHD, relevant (*Rel.*) and erroneous edges (*Err.*) obtained after 1000 FEs. 30 runs were performed for each algorithm on each dataset and unpaired t-tests carried out. Best values over all methods appear in bold while those not statistically significant from the best (p-value>0.003 after Bonferroni correction) are marked with a $*$ symbol. On small problems, K2GA exhibits results that are either the best or not significantly different from it. BNs obtained by K2GA on larger problems such as *car* and *boerlage* suffer from poorer quality in comparisons with other methods as illustrated by the corresponding C and SHD values. The use of IM generally improves the BNs obtained by the GA as the dimension of the data grows and confirms results foreseen in [2]. On the other hand, the standard COMMA is always competitive with the GA-based methods in terms

Table 2. Characteristics of best BNs obtained by each algorithm after 1000 FEs

		Asia	Tank	Credit	Car	Boerlage
K2GA	C	6.43 (0.56)	**14.77** (1.75)	8.87*(1.18)	10.83 (1.34)	17.43 (2.26)
	SHD	1.60 (0.66)	**10.93** (2.66)	4.70*(1.64)	13.33 (2.88)	24.80 (3.03)
	Rel.	**8.00** (0.00)	19.67*(0.54)	**12.00** (0.0)	14.27*(0.81)	28.07*(0.63)
	Err.	0.03*(0.18)	6.03*(1.52)	1.57 (0.56)	9.90*(2.71)	14.17*(2.44)
IMK2GA	C	6.57*(0.56)	14.20*(2.04)	9.23*(0.50)	12.13*(0.76)	**19.60** (2.14)
	SHD	1.47 (0.67)	11.30*(2.76)	3.80*(0.60)	11.57 (1.96)	**20.90** (3.23)
	Rel.	**8.00** (0.00)	19.83*(0.37)	**12.00** (0.00)	14.23*(0.76)	28.03*(0.48)
	Err.	0.03*(0.18)	**5.67** (1.07)	**1.03** (0.18)	9.47*(2.08)	**12.47** (2.16)
COMMA	C	6.70*(0.46)	13.43*(1.87)	9.13*(1.23)	**12.47** (0.96)	18.70*(2.08)
	SHD	1.37 (0.66)	12.27*(2.64)	4.27*(1.71)	**10.93** (2.14)	22.87*(2.59)
	Rel.	**8.00** (0.00)	**19.87** (0.34)	**12.00** (0.00)	**14.33** (0.54)	**28.17** (0.64)
	Err.	0.07*(0.36)	5.83*(1.13)	1.40 (0.55)	**9.07** (1.67)	13.40*(1.65)
$IM-COMMA$	C	**6.90** (0.30)	14.17*(2.03)	**9.63**(1.20)	11.63*(1.17)	18.13*(2.32)
	SHD	**1.10** (0.30)	11.83*(2.78)	**3.60** (1.74)	12.80 (2.41)	23.53 (3.33)
	Rel.	**8.00** (0.00)	19.70*(0.46)	**12.00** (0.00)	13.87*(0.81)	28.07*(0.63)
	Err.	**0.00** (0.00)	6.30*(1.39)	1.23*(0.62)	10.57*(2.38)	13.60*(2.24)
$COMMA_d$	C	6.87*(0.34)	14.53*(2.06)	9.37*(1.28)	11.40 (1.11)	17.10 (1.78)
	SHD	1.13*(0.34)	11.23*(3.24)	4.10*(1.81)	12.80 (2.18)	24.93 (2.39)
	Rel.	**8.00** (0.00)	19.83*(0.37)	**12.00** (0.00)	14.20*(0.54)	27.83*(0.90)
	Err.	**0.00** (0.00)	5.93*(1.59)	1.47 (0.62)	10.00*(1.73)	14.20*(2.27)
$IM-COMMA_d$	C	6.87*(0.34)	14.43*(2.46)	9.53*(1.33)	11.63 (1.08)	17.87*(2.31)
	SHD	1.13*(0.34)	11.20*(3.69)	3.70*(1.93)	11.93*(2.62)	24.77 (2.92)
	Rel.	**8.00** (0.00)	19.70*(0.46)	**12.00** (0.00)	14.17*(0.73)	28.03*(0.80)
	Err.	**0.00** (0.00)	5.93*(1.67)	1.23*(0.67)	9.40*(2.22)	14.60 (1.98)



(a) Best Fitness **(b)** SHD

Fig. 2. Evolution of the best fitness and SHD on *car*, averaged over 30 runs

of BN quality. Figure 2 illustrates how best fitness and SHD vary over time on *car*. The best fitness in $IMK2GA$ appears to be better than those of other methods along the search, while $K2GA$ converges very early. The variation between COMMA and GA-based methods shows different patterns. For both SHD and fitness, $IMK2GA$ follows cycles of improvement and convergence interrupted by migrations of solutions. COMMA approaches show a more gentle improvement over time and the effect of migration is less obvious. Here it seems that migration has a different effect on the algorithms. While migrating solutions helps bringing together local optima within a GA, it does not affect as much COMMA because convergence is never reached. A migration may just be considered as a successful mutation from the ρ worst agents in the population.

To investigate how the different methods behave with respect to diversity within their populations, KTD is plotted in Figure 3 for *car*. The effect of the three migrations on $IMK2GA$ is very clear, but a consequent loss of diversity is still observed that leads to its convergence. Figure 3 also points out that diversity is maintained with the different COMMA algorithms throughout the run. Here, it can be argued that $COMMA$ would perform better on longer runs because it had not converged when experiments were stopped. Although it is difficult to draw a clear picture of the impact of allowing degradation of solutions in $COMMA$, it seems that in the chosen configuration, it does not bring obvious improvements. Figure 4 illustrates how positions of agents vary along the search on *tank* in $COMMA$ and $COMMA_d$ respectively. Each column represents the ordered population of agents in a typical run from the initial state on the left hand side to its final state on the right hand side. Each agent is represented by a shade of grey. For both $COMMA$ and $COMMA_d$, the constant exploration and exploitation is clear as agents change positions many times. During the run of $COMMA$, a total of 8 distinct agents have reached the two first positions in

[1] Additional results and coloured figures are available at `http://www.comp.rgu.ac.uk/staff/orc/ppsn2012-results`

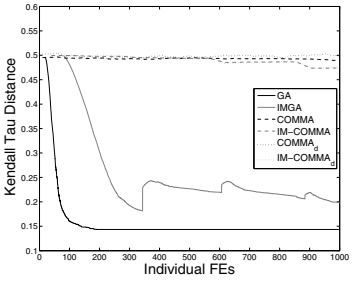


Fig. 3. Evolution of the KTD on *car*, averaged over 30 runs

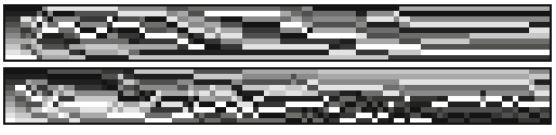


Fig. 4. Evolution of agent positions along one typical run of $COMMA$ (top) and $COMMA_d$ (bottom) on *tank*

the population and the final best agent has moved and occupied every position for at least one generation. High pixelation in the lower ranks of the position representation of $COMMA_d$ highlights that new solutions are being explored even when variation in top ranks becomes rare due to the high solution quality.

6 Conclusions

In this paper, a novel approach to learning BN structures was presented. The algorithm focuses on avoiding early convergence, a limitation foreseen in other methods such as K2GA by means of agents mutating their solutions to different extents. Since the effect of such mutation largely depends on the distance that is defined, each agent was assigned a mutation distance relative to its position in the population. Results show that COMMA is generally competitive with IMK2GA and is able to outperform it on some of the problems from the selected test suite when considering the quality of BNs obtained at the end of the runs. In addition, diversity remains high in COMMA all along the search while GA-based methods converge fast, suggesting that better BNs could be produced with more FEs. Future work will focus on assessing the effect of different mutation operators on the performance of COMMA, but also to study how such an approach performs on other permutation-based optimization problems. Finally the idea of monitoring the changes in agents positions should be extended in order to perform migrations in a dynamic manner within an IM.

References

1. Kabli, R., Herrmann, F., McCall, J.: A chain-model genetic algorithm for bayesian network structure learning. In: Proc. of the 9th Conference on Genetic and Evolutionary Computation, pp. 1271–1278 (2007)
2. Regnier-Coudert, O., McCall, J.: An island model genetic algorithm for bayesian network structure learning. In: Proce. of the IEEE CEC 2012 (2012)
3. Wu, Y., McCall, J., Corne, D.: Two novel ant colony optimization approaches for bayesian network structure learning. In: Proce. of the IEEE CEC 2010, pp. 4473–4479 (2010)
4. Cowie, J., Oteniya, L., Coles, R.: Particle swarm optimization for learning bayesian networks. In: Proc. of World Congress on Engineering, pp. 2–4 (2007)
5. Larranaga, P., Kuijpers, C., Murga, R., Yurramendi, Y.: Learning bayesian network structures by searching for the best ordering with genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics 26(4), 487–493 (1996)
6. Cooper, G., Herskovits, E.: A bayesian method for the induction of probabilistic networks from data. Mach. Learn. 9(4), 309–347 (1992)
7. Heckerman, D., Geiger, D., Chickering, D.: Learning bayesian networks: the combination of knowledge and statistical data. Mach. Learn. 20(3), 197–243 (1995)
8. dos Santos, E., Hruschka, E., Ebecken, N.: A distance-based mutation operator for learning bayesian network structures using evolutionary algorithms. In: Proc. of the IEEE Congress on Evolutionary Computation, pp. 1–8 (2010)
9. Thierens, D.: Adaptive mutation rate control schemes in genetic algorithms. In: Proc. of the IEEE CEC, pp. 980–985 (2002)
10. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
11. Lauritzen, S., Spiegelhalter, D.: Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society, 157–224 (1988)
12. Genie and smile, http://genie.sis.pitt.edu/ (accessed: March 30, 2012)
13. Norsys, http://www.norsys.com (accessed: March 30, 2012)
14. Boerlage, B.: Link strength in bayesian networks. Master's thesis, University of British Columbia (1992)
15. de Jongh, M., Druzdzel, M.: A comparison of structural distance measures for causal bayesian network models. In: Recent Advances in Intelligent Information Systems, pp. 443–456 (2009)

A Meta-learning Prediction Model of Algorithm Performance for Continuous Optimization Problems*

Mario A. Muñoz[1], Michael Kirley[2], and Saman K. Halgamuge[1]

[1] Department of Mechanical Engineering
[2] Department of Computing and Information Systems
The University of Melbourne, Parkville, Victoria, Australia
mariom@student.unimelb.edu.au

Abstract. Algorithm selection and configuration is a challenging problem in the continuous optimization domain. An approach to tackle this problem is to develop a model that links landscape analysis measures and algorithm parameters to performance. This model can be then used to predict algorithm performance when a new optimization problem is presented. In this paper, we investigate the use of a machine learning framework to build such a model. We demonstrate the effectiveness of our technique using CMA-ES as a representative algorithm and a feed-forward backpropagation neural network as the learning strategy. Our experimental results show that we can build sufficiently accurate predictions of an algorithm's expected performance. This information is used to rank the algorithm parameter settings based on the current problem instance, hence increasing the probability of selecting the best configuration for a new problem.

Keywords: Automatic analysis of algorithms, algorithm configuration, heuristic methods, randomized algorithms, meta-learning models.

1 Introduction

One of the most interesting questions in search and optimization is: "Before we perform a run, can we estimate the likelihood that the algorithm a will be successful on a given continuous optimization problem f?" In most circumstances, it is very difficult to answer this question. It is a well-known fact that each search algorithm exploits particular characteristics of the landscape [1]. As such, it is very optimistic to expect that an algorithm would work reasonably well across a wide range of continuous optimization problems unless some restrictions are in place [2]. However, it is possible to use machine learning techniques to elucidate information related to the effects of algorithm selection and/or parameter settings on similar problems; an approach known as *meta-learning*.

Leyton-Brown and co-workers [3] describe an automated algorithm selection method for boolean satisfiability problems. Their approach, based on methods proposed by Rice [4], employs supervised learning to build a model that can predict algorithm runtime. By comparing the estimated performance, it was possible to select the algorithm most likely

* This work has been partially funded through a 2012-2013 DAAD/Go8 Grant.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 226–235, 2012.

to be suited to the task at hand. Related work was also reported in [1]. Smith-Miles [5] describe a similar meta-learning framework that can be used to develop automated algorithm selection and ranking models. More generally, Hoos [6] describes automated techniques applicable for algorithm selection and configuration for NP-hard problems.

In this paper, we use the meta-learning framework outlined above to build a prediction model of algorithm performance for continuous optimization problems. The model inputs include information about f – such as the dimension, target value and landscape features – and the parameter settings of a. The model output is the algorithm performance measured by the number of required function evaluations to find a solution. Information about a new problem f_n can then be used by the model to estimate the performance of a on f_n. To illustrate the efficacy of this approach, we model instances of the well-known Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) using a feed-forward backpropagation neural network (NN). The knowledge base used for training the model is obtained by processing data from the Comparing Continuous Optimization (COCO) set of benchmarks [7]. To validate the model, we use data from the CEC2005 set of benchmarks [8].

The remainder of the paper is organized as follows. In Section 2, we discuss background material related to the meta-learning algorithm selection framework used in this study. Section 3 describes the meta-learning model for continuous optimization problems in detail. Here, we describe the landscape features and the performance metric used as inputs and outputs. Section 4 describes the experimental procedure based on alternative CMA-ES parameter settings. Section 5 presents the results obtained from our experiments. Finally, Section 6 discusses the results and states the conclusions. Avenues for further work are also identified in this section.

2 Background

A continuous optimization problem is such that, given a cost function f that maps the search set $\mathcal{X} \subset \mathbb{R}^n$ to the objective set $\mathcal{Y} \subset \mathbb{R}$, we want to find one or more candidate solutions $\mathbf{x}_o \in \mathcal{X}, y_o = f(\mathbf{x}_o)$, such that $|y_o - y^\star| \ll \delta$, where $\delta \to 0$ and $\mathbf{x}^\star \in \mathbb{R}^n, y^\star = f(\mathbf{x}^\star)$ are the location of the global optimum and its value.

This type of problem is usually described through the search landscape metaphor [9]. A landscape $\mathcal{L}$ for a function f is defined as the tuple $\mathcal{L} = \{\mathcal{X}, f, d\}$, where d denotes a distance measure. The distance relates solutions among each other, hence it allows the systematic search for $\mathbf{x}_o$ in $\mathcal{X}$. Ideally, we expect that our search produces an acceptable solution after a bounded number of function evaluations. The opposite case is known as *premature convergence*, when the search is unable to generate solutions outside a small area under examination and the solution obtained is unacceptable.

As a direct consequence of the large number of existing optimization algorithms, it is difficult to determine which algorithm is able to efficiently exploit the search landscape structure for a given problem [10]. Deciding which algorithm to use is referred to as the "algorithm selection problem" by Rice [4]. In his seminal work, Rice defined four different sets: The *problem set*, $\mathcal{F}$, which contains functions that map $\mathcal{X}$ to $\mathcal{Y}$; the *algorithm set*, $\mathcal{A}$, which contains algorithms capable of searching for $\mathbf{x}_o$ in $\mathcal{X}$; the *performance set*, $\mathcal{P} \subset \mathbb{R}$, which contains the feasible values of $\rho(f, a)$, a measure for the

cost of applying an algorithm $a \in \mathcal{A}$ in a problem f; and the *set of landscape features*, $\mathcal{C} \subset \mathbb{R}^m$, which is a set of attributes of the functions in $\mathcal{F}$. These features are selected in a way such that varying complexities are exposed, known structural properties are captured, and any known advantages and limitations of the different algorithms can be related to the features.

The algorithm selection problem investigated by Rice has been extended and evaluated in a variety of computational problem domains using a meta-learning framework [1,3,5]. In this framework, algorithm selection is performed by exploring insights gained from previous experiments. Here, a function $g : \mathcal{C} \mapsto \mathcal{P}$ can be used to predict the algorithm performance based on specific input features. If we know beforehand the values of the features of a subset of functions from $\mathcal{F}$ and the values of ρ for an algorithm in $\mathcal{A}$; then, it is possible to use a learning strategy (such as linear regression) to identify the function g. When a new problem is encountered, g (or the performance model) can be used to predict the performance of the algorithm.

When working in the continuous optimization domain, careful attention must be given to the design of model inputs and outputs. Inputs to the model g may include the set of landscape features $\mathcal{C}$, and parameters of the algorithm set, $\mathcal{A}$. However, calculating the set of landscape features may well be a stumbling block, as this is a non-trivial computation [11,12]. Previous work in the continuous optimization domain recognizes the necessity to link landscape features and algorithm parameters to algorithm performance [13–18]. Subsequently, an appropriate learning strategy must be employed to generate model output $\rho\,(f,a)$ based on a range of algorithm parameter values.

3 Prediction Model

Our model is an implementation of the meta-learning framework described above tailored for continuous optimization problems. Here, we build a regression model. By considering the landscape features and algorithm parameters as independent variables and algorithm performance as the dependent variable, the model can be used to predict algorithm behavior for a given problem. A high-level overview of the model is presented in Figure 1.

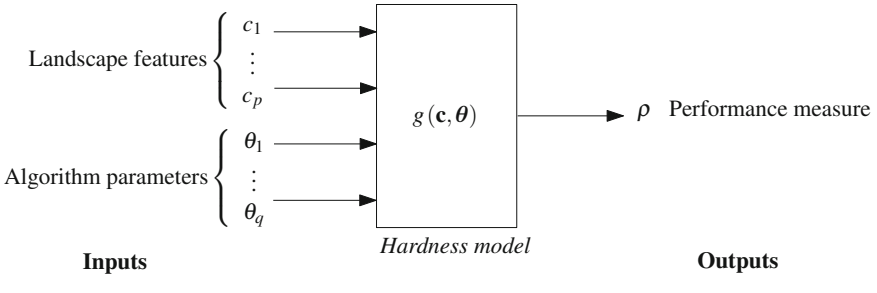


Fig. 1. Structure of a meta-learning model for a continuous optimization algorithm

3.1 Model Inputs

Landscape Features. Over the past few decades, when compared with the number of novel algorithms, relatively limited attention has been given to the question of what attributes a certain problem has, how to quantify them, and how the performance can be related to such attributes [18]. This is because, among other reasons, it is not a single attribute that defines the difficulty, but the interplay between different attributes [19]. Landscape analysis methods provide descriptive statistics related to algorithm performance. However, designing suitable analysis methods for a given domain is not straightforward. In some cases, the effort in calculating exact values of these statistics is greater than running a simple search algorithm [11,12]. Approximations can be efficiently calculated, but the question remains if the loss of precision is too large to work with [20]. Besides, analyzing a whole landscape by using a single statistic is overly optimistic [18,21,22]. These are limitations that must be acknowledged.

For this work, we selected the following features:

- Dispersion (DISP) [23] identifies features of the global structure. It is defined as the pairwise distance between the q best points — usually $q = 100$ — from a sample of size p, as shown in (1).

$$DISP = \frac{1}{q\,(q-1)} \sum_{i=1}^{q} \sum_{j=1, j\neq i}^{q} d\,(\mathbf{x}_i, \mathbf{x}_j) \tag{1}$$

- Fitness distance correlation (FDC) [24] identifies the relationship between the position and the cost value, and has demonstrated capability to identify deceptiveness in the landscape. To calculate FDC, assume that from a sample of size p, $\hat{\mathbf{x}}_o = \arg\min f\,(\mathbf{x}_i)\,, i = 1, \ldots, p$ and $\hat{y}_o = f\,(\hat{\mathbf{x}}_o)$. Then, FDC is calculated using (2), where $d = \|\hat{\mathbf{x}}_o - \mathbf{x}_i|$, $\bar{y}$ and $\bar{d}$ are the averages of the cost and the distance, and $\hat{\sigma}_y$ and $\hat{\sigma}_d$ are the standard deviation of the cost and the distance.

$$FDC = \frac{1}{p-1} \sum_{i=1}^{p} \left(\frac{y_i - \bar{y}}{\hat{\sigma}_y} \right) \left(\frac{d_i - \bar{d}}{\hat{\sigma}_d} \right) \tag{2}$$

- Multiple correlation coefficient (R^2) identifies the relationship between n variables using a linear model approximation. Let R^2 be calculated using (3), where $\mathbf{r}_{\mathbf{x}y}$ is the vector of cross-correlations between the predictor variables on $\mathcal{X}$ and the criterion variable on $\mathcal{Y}$ and $\mathbf{R}_{\mathbf{xx}}$ is the matrix of inter-correlations between predictor variables.

$$R^2 = \mathbf{r}_{\mathbf{x}y}^{\top} \mathbf{R}_{\mathbf{xx}}^{-1} \mathbf{r}_{\mathbf{x}y} \tag{3}$$

- Variable significance [25] estimates the amount of information that a subset of predictor variables provides for the criterion variable. Let $\mathcal{V} = \{1, \ldots, n\}$ be a set of variables indexes, $v \in \mathcal{V}$ the index of one of such variables, and $V \subset \mathcal{V}$ be any combination of such indexes. The significance of V is calculated by (4), where $\hat{I}\,(\mathcal{X}_V; \mathcal{Y})$ is the estimated mutual information and $\hat{H}\,(\mathcal{Y})$ the estimated entropy of $\mathcal{Y}$. Let $\zeta^{(k)}$ and $\sigma_{\varepsilon}^{(k)}$ be the average significance of order k and its standard deviation respectively, where $k = |V|$.

$$\zeta(V) = \frac{\hat{I}(\mathcal{X}_V;\mathcal{Y})}{\hat{H}(\mathcal{Y})} \tag{4}$$

$$\zeta^{(k)} = \frac{1}{\binom{n}{k}} \sum_{V\subset\mathcal{V},|V|=k} \zeta(V) \tag{5}$$

$$\sigma_\zeta^{(k)} = \sqrt{\frac{1}{\binom{n}{k}} \sum_{V\subset\mathcal{V},|V|=k} \left(\zeta(V) - \zeta^{(k)}\right)^2} \tag{6}$$

– Entropic epistasis [25] evaluates the contribution that a single variable has to the fitness given the state of other variables. For a variable subset V the entropic epistasis is calculated by (7). Let $\varepsilon^{(k)}$ and $\sigma_\varepsilon^{(k)}$ be the average entropic epistasis of order k and its standard deviation.

$$\varepsilon(V) = \frac{\hat{I}(\mathcal{X}_V;\mathcal{Y}) - \sum_{v\in V} \hat{I}(\mathcal{X}_v;\mathcal{Y})}{\hat{I}(\mathcal{X}_V;\mathcal{Y})} \tag{7}$$

$$\varepsilon^{(k)} = \frac{1}{\binom{n}{k}} \sum_{V\subset\mathcal{V},|V|=k} \varepsilon(V) \tag{8}$$

$$\sigma_\varepsilon^{(k)} = \sqrt{\frac{1}{\binom{n}{k}} \sum_{V\subset\mathcal{V},|V|=k} \left(\varepsilon(V) - \varepsilon^{(k)}\right)^2} \tag{9}$$

Algorithm Parameters. The "algorithm selection" problem in many cases is equivalent to the "algorithm configuration" problem as it is possible to consider two instances of the same algorithm as two completely different ones if they differ only in one parameter [26]. An experimentally driven meta-learning approach has been suggested for the latter problem [12]. Thus, we adopt this approach in our model.

We use CMA-ES as the base algorithm for our investigation, and the following parameters as inputs for the model.

– Target precision (e_{target}) is the error between the best solution and the target solution. As many practical problems do not have a target solution, a value can be developed if we consider an improvement by certain user defined percentage over the best known solution before the experimental run.
– Size of the population (λ).
– Depending on the algorithm, it might be possible to have rules that define how the individuals are generated or evaluated. Mirrored ($M \in \{0,1\}$) indicates whether the offspring are generated in pairs, where one is 180° from the other. Serialized ($S \in \{0,1\}$) indicates if the offspring is evaluated sequentially. Hence, if an improving offspring is found the others are not evaluated.

3.2 Model Output

The performance ρ of the model is set to be the expected running time $\hat{t}$ of the algorithm. Here, $\hat{t}$ provides an estimation of the average number of function evaluations required by the algorithm a to reach y_{target} for the first time [7]. The expected running time is calculated as follows:

$$\hat{t}(f, a, y_{\text{target}}) = \frac{\#\text{FEs}\left(y_{\text{best}} \geq y_{\text{target}}\right)}{\#\text{succ}} \tag{10}$$

where $\#\text{FEs}\left(y_{\text{best}} \geq y_{\text{target}}\right)$ is the number of function evaluations over all trials where the best function value, y_{best}, was not smaller than the target function value, and #succ is the number of runs where target precision was achieved. When not all trials are successful, $\hat{t}$ depends strongly on the termination criteria of the algorithm.

3.3 Regression Model

In this study, we use an NN to build the model. It is important to note that the meta-learning framework is flexible and any appropriate learning strategy could be used. While other regression methods might provide different — even superior — accuracy, as a proof of concept the NN will suffice.[1]

To train this model, inputs include landscape features (from a collections of problem instances of various complexities) and algorithm parameter values (corresponding to alternative instantiations of the CMA-ES algorithm) as described in Section 3.1. The model output value is $\log_{10}(\hat{t})$. Since the target precision e_{target} can take different values for the same problem, a pattern for each e_{target} is created where the other input values are kept constant.

The accuracy of the resulting model depends on several factors: the diversity in the knowledge base used to train the model, the relevance of the features and their precision, and the training method used in the model. For our purpose, the accuracy of the model would be evaluated on the capability to provide a realistic ranking of the different configurations of the algorithm. While an accurate estimation of the expected running time would be desirable, at this exploratory stage it is unlikely to be obtained.

4 Experiments

A comprehensive set of simulation experiments were performed to evaluate the efficacy of the proposed meta-learning prediction model of algorithm performance for continuous optimization problems.

A multi-layered feed-forward neural network (2 hidden layers; 10 neurons in each layer) was used for the regression model. The training method employed was the Levenberg-Marquardt back-propagation algorithm. The inputs and outputs of the model were normalized in the $[-1, 1]$ range. MATLAB version 2009b was used for implementation.

[1] We leave the evaluation of alternative learning strategies to future work.

Eight configurations of the CMA-ES algorithm were used for this experiment: standard, mirrored, serialized and mirrored-serialized with a single parent and either two or four offspring.

To train the model, we use the functions from the COCO noiseless set [7] in $\{2,3,5,10,20\}$ dimensions. The features are calculated using fifteen runs of $10^3 \cdot n$ function evaluations using uniform random sampling ($p = 15 \cdot 10^3 \cdot n$). While it seems that a large amount of data must be collected before the model can be used, we presume that it is possible to use data from an algorithm run to calculate the features, hence avoiding the need for an additional data extraction experiment. However, this hypothesis is not tested in this work. The performance metric is calculated over fifteen runs for each of the eight CMA-ES configurations with a target value of 10^{-8} and a maximum number of function evaluations equal to $10^4 \cdot n$.

To evaluate the effectiveness of our model in predicting the performance of a given algorithm on new problems, we use a subset of the problems from the CEC2005 benchmarks [8]. We limit the evaluation of our model to the noiseless functions and those functions whose optimum is inside the initialization region. The functions were evaluated on $\{2,3,5,10,20\}$ dimensions. The collection of metric values for the CEC2005 problems followed the same procedure as for COCO benchmark functions.

5 Results

We examine the predictions made by our model using the CEC benchmarks. The model is fed with landscape features that represent a benchmark problem ($\mathbf{c}$), algorithm parameters that define the configuration ($\boldsymbol{\theta}$), and a desired target precision (e_{target}). Then, the predicted $\hat{t}$ for each configuration is ranked from the lowest to the highest at a fixed e_{target}. Figures 2(a) and 2(b) show the resulting rankings for the 5-dimensional versions of the Sphere function and the Hybrid composition function 1, respectively. The top plots represent the actual ranking while the bottom plots represent the predicted ranking. The abscissa are the $\log_{10}\left(e_{\text{target}}\right)$ organized from lowest precision on the left to the highest precision on the right. The ordinates are the ranking of a given configuration, where the lowest is the worst performing and the highest is the best performing. Each line on the plot represents a configuration.

We quantify the similarity between rankings using the following procedure. For a fixed e_{target}, let r_a be the ranking based on the actual performance of each configuration and r_p be the ranking produced by sorting the predictions generated by the model. Let $\delta_p = \left|r_a - r_p\right|$ be the difference between the two rankings. When this measure is calculated for CEC benchmark data, the average δ_p over all scenarios — for each target precision, dimension and benchmark function — is 12.69.

A baseline is required to asses the impact of the differences in rankings. Let r_r be a random ranking — the positions in the ranking of each configuration are randomly generated — which is kept fixed for all the scenarios. Let δ_r be the difference between the random to the actual ranking. We produce 100 different random rankings, each with its own value of δ_r. The average value of the δ_r is 21.01, which is 39.59% higher than the average δ_p. This indicates that by using the predicted ranking we improve our chances to select the best algorithms early on the experiment. Table 1 lists the difference between δ_r

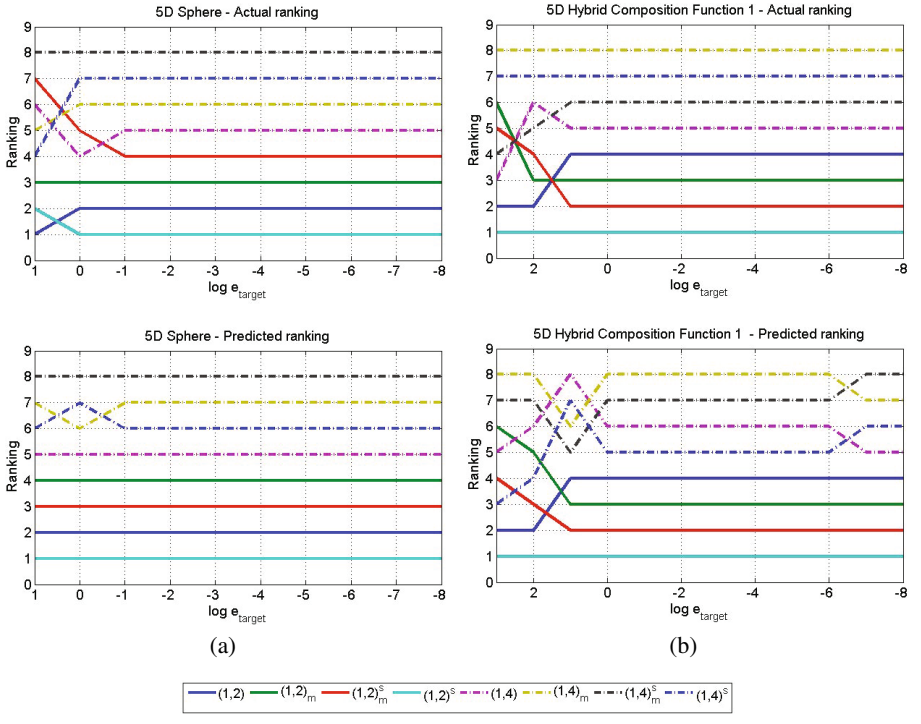


Fig. 2. Actual and predicted rankings for the different CMA-ES configurations on the 5D Sphere function (a) and on the 5D first hybrid composition function (b) of the CEC2005 benchmark

Table 1. Average difference between the random and predicted rankings. A negative value indicates a scenario where the random ranking is more accurate than the predicted ranking.

f	2D	3D	5D	10D	20D	f	2D	3D	5D	10D	20D
1	7.42	8.85	14.28	7.81	4.59	12	4.17	6.19	9.54	6.03	6.76
2	8.11	8.17	13.14	8.37	4.41	13	15.26	8.72	-2.54	-10.74	21.48
3	10.33	5.89	11.37	8.58	10.73	14	5.07	7.88	13.90	6.48	2.94
4	12.28	5.55	10.74	14.06	-0.16	15	12.96	9.09	6.23	10.40	10.55
5	8.36	9.42	12.65	8.40	8.45	16	4.42	3.67	4.21	2.52	1.12
6	6.68	11.23	6.94	3.78	3.09	17	3.21	6.30	8.10	4.90	4.88
7	4.31	11.49	16.18	8.63	20.14	18	11.64	18.11	18.34	10.19	12.08
8	9.88	11.19	15.10	12.64	7.20	19	2.41	-5.44	3.20	4.19	1.34
9	9.43	5.59	16.68	8.60	12.62	20	-2.73	10.02	14.90	1.06	1.36
10	7.94	8.53	10.97	11.61	4.58	21	11.37	15.79	5.27	2.79	-5.13
11	5.39	3.35	10.41	6.63	-3.22						

and δ_p for each benchmark function when averaged over e_{target}. With some exceptions on f_4, f_{11}, f_{13}, f_{19}, f_{20} and f_{21}, in most cases the dissimilarity between δ_p is lower than δ_r. When this is not the case, the differences can be due to the precision of the model, which can be improved by changing the learning strategy or preprocessing the input data.

6 Conclusion

In this paper, we used a meta-learning framework to build a model that captures the relationship between problem cost function structure, algorithm parameters and a given performance metric. The model is used to predict algorithm performance measured in terms of the number of function evaluation required.

A NN was used as the underlying learning strategy. The network inputs included a number of landscape characteristics (calculated using well-known statistical estimators) and selected parameter settings. The network was trained, and subsequently tested, using a suite of benchmark continuous optimization problems with varying characteristics. The simulation results clearly demonstrate that the model was able to predict the relative ranking values for given algorithm-parameter combinations effectively.

Model performance was measured by comparing predicted and actual rankings of algorithm parameter settings on new problem instances. This implies that the ranking can be verified if all the configurations are tested. In a practical situation this is often not the case, as the experiments will be censored when an acceptable solution is reached. Examining the effects of censoring on the rankings is one avenue of future work.

All landscape analysis was performed off-line — an initial experiment was carried out to calculate landscape statistics that form part of the input of our model. In future work, we will transfer such calculations to an on-line mode. The benefits of such an approach are highlighted by reviewing Figures 2(a) and 2(b). Note that the best configuration is not the same at all target values. An on-line prediction mechanism coupled with the ability to switch between configurations offers the potential for significant improvement in optimization performance.

References

1. Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K.: Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 213–228. Springer, Heidelberg (2006)
2. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1(1), 67–82 (1997)
3. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Empirical hardness models: Methodology and a case study on combinatorial auctions. J. ACM 56, 22:1–22:52 (2009)
4. Rice, J.: The algorithm selection problem. In: Advances in Computers, vol.15, pp. 65–118. Elsevier (1976)
5. Smith-Miles, K.A., James, R.J.W., Giffin, J.W., Tu, Y.: A Knowledge Discovery Approach to Understanding Relationships between Scheduling Problem Structure and Heuristic Performance. In: Stützle, T. (ed.) LION 3. LNCS, vol. 5851, pp. 89–103. Springer, Heidelberg (2009)
6. Hoos, H.H.: Programming by optimization. Commun. ACM 55(2), 70–80 (2012)
7. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking BBOB-2010: Experimental setup. Technical Report RR-7215, INRIA (September 2010)

8. Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y., Auger, A., Tiwari, S.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, NTU, Singapore and IIT, Kanpur (2005)
9. Reeves, C.: Fitness landscapes. In: Search Methodologies, pp. 587–610. Springer (2005)
10. Hough, P., Williams, P.: Modern machine learning for automatic optimization algorithm selection. In: Proceedings of the INFORMS Artificial Intelligence and Data Mining Workshop (2006)
11. He, J., Reeves, C., Witt, C., Yao, X.: A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability. Evol. Comput. 15(4), 435–443 (2007)
12. Smith-Miles, K.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. 41(1), 6:1–6:25 (2009)
13. Francois, O., Lavergne, C.: Design of evolutionary algorithms-a statistical perspective. IEEE Trans. Evol. Comput. 5(2), 129–148 (2001)
14. Steer, K.C.B., Wirth, A., Halgamuge, S.K.: Information Theoretic Classification of Problems for Metaheuristics. In: Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H.A., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K.C., Branke, J., Shi, Y. (eds.) SEAL 2008. LNCS, vol. 5361, pp. 319–328. Springer, Heidelberg (2008)
15. Malan, K., Engelbrecht, A.: Quantifying ruggedness of continuous landscapes using entropy. In: Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC 2009), pp. 1440–1447 (May 2009)
16. Caamaño, P., Prieto, A., Becerra, J.A., Bellas, F., Duro, R.J.: Real-Valued Multimodal Fitness Landscape Characterization for Evolution. In: Wong, K.W., Mendis, B.S.U., Bouzerdoum, A. (eds.) ICONIP 2010, Part I. LNCS, vol. 6443, pp. 567–574. Springer, Heidelberg (2010)
17. Mersmann, O., Preuss, M., Trautmann, H.: Benchmarking Evolutionary Algorithms: Towards Exploratory Landscape Analysis. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 73–82. Springer, Heidelberg (2010)
18. Müller, C.L., Sbalzarini, I.F.: Global Characterization of the CEC 2005 Fitness Landscapes Using Fitness-Distance Analysis. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcázar, A.I., Merelo, J.J., Neri, F., Preuss, M., Richter, H., Togelius, J., Yannakakis, G.N. (eds.) EvoApplications 2011, Part I. LNCS, vol. 6624, pp. 294–303. Springer, Heidelberg (2011)
19. Richter, H.: Coupled map lattices as spatio-temporal fitness functions: Landscape measures and evolutionary optimization. Phys. Nonlinear Phenom. 237(2), 167–186 (2008)
20. Watson, J., Howe, A.: Focusing on the individual: Why we need new empirical methods for characterizing problem difficulty. In: Working Notes of ECAI 2000 Workshop on Empirical Methods in Artificial Intelligence (August 2000)
21. Beck, J., Watson, J.: Adaptive search algorithms and fitness-distance correlation. In: Proceedings of the Fifth Metaheuristics International Conference (2003)
22. Smith, T., Husbands, P., Layzell, P., O'Shea, M.: Fitness landscapes and evolvability. Evol. Comput. 10(1), 1–34 (2002)
23. Lunacek, M., Whitley, D.: The dispersion metric and the CMA evolution strategy. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 477–484. ACM, New York (2006)
24. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 184–192. Morgan Kaufmann Publishers Inc. (1995)
25. Seo, D., Moon, B.: An information-theoretic analysis on the interactions of variables in combinatorial optimization problems. Evol. Comput. 15(2), 169–198 (2007)
26. Rice, J.: Methodology for the algorithm selection problem. In: Proceedings of the IFIP TC 2.5 Working Conference on Performance Evaluation of Numerical Software (1979)

Pruning GP-Based Classifier Ensembles by Bayesian Networks

C. De Stefano[1], G. Folino[2], F. Fontanella[1], and A. Scotto di Freca[1]

[1] Università di Cassino e del Lazio Meridionale, Italy
{destefano,fontanella,a.scotto}@unicas.it
[2] ICAR-CNR Istituto di Calcolo e Reti ad Alte Prestazioni, Italy
folino@icar.cnr.it

Abstract. Classifier ensemble techniques are effectively used to combine the responses provided by a set of classifiers. Classifier ensembles improve the performance of single classifier systems, even if a large number of classifiers is often required. This implies large memory requirements and slow speeds of classification, making their use critical in some applications. This problem can be reduced by selecting a fraction of the classifiers from the original ensemble. In this work, it is presented an ensemble-based framework that copes with large datasets, however selecting a small number of classifiers composing the ensemble. The framework is based on two modules: an ensemble-based Genetic Programming (GP) system, which produces a high performing ensemble of decision tree classifiers, and a Bayesian Network (BN) approach to perform classifier selection. The proposed system exploits the advantages provided by both techniques and allows to strongly reduce the number of classifiers in the ensemble. Experimental results compare the system with well-known techniques both in the field of GP and BN and show the effectiveness of the devised approach. In addition, a comparison with a pareto optimal strategy of pruning has been performed.

1 Introduction

In the last two decades, classifier ensemble techniques have shown to be a viable alternative to using a single classifier [10]. Such techniques try to effectively combine the responses provided by a set of classifiers, that have been properly trained in such a way that they are "diverse", i.e. they make uncorrelated errors. The responses are usually combined by means of a voting mechanism, which labels an unknown sample by assigning it the class label which has the highest occurrence among those provided by the whole set of classifiers. Ensemble techniques have been also used for improving GP–based classification systems [2,6,9]. In [2] and [6], ensembles of decision trees are evolved, and the diversity among the ensemble members is obtained using techniques like bagging and boosting. Both such approaches are meta-algorithms that aggregate multiple classifiers, or hypotheses, generated by the same learning algorithm trained on different distributions of training data. As concerns the combining rules, bagging uses

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 236–245, 2012.

the majority vote, while boosting adopts the weighted majority vote, where the weight associated to a classifier is computed on the basis of its overall accuracy on the training data. In [6], a novel GP–based classification system, called *BoostCGPC*, based on the AdaBoost.M2 boosting algorithm [8], has been presented. It is based on a model of the population, in which individuals interact according to a cellular automata inspired model, whose goal is to enable a fine-grained parallel implementation of GP. In this model, each individual has a spatial location on a low-dimensional grid and interacts only with other individuals within a small neighborhood. The experimental results presented in [6] showed that boostCGPC represents an effective classification algorithm able to deal with large data sets.

As mentioned above, classifier ensembles may improve the performance of single classifier systems, but often a large number of classifiers is required. This implies large memory requirements and slow speeds of classification, making their use critical in some applications. This problem can be solved by selecting a fraction of the classifiers from the original ensemble. Such reduction, often denoted as "ensemble pruning" in the literature, can perform even better than the whole ensemble if a subset of complementary classifiers is selected [12,1]. When the cardinality N of the whole ensemble is high, the problem of finding the optimal sub-ensemble becomes computationally intractable because of the resulting exponential growth of the search space. Several heuristic algorithms have been proposed in the literature for finding near optimal solutions [12].

In a previous work [4] the above problem has been faced by reformulating the classifier combination problem as a pattern recognition one, in which the pattern is represented by the set of class labels provided by the classifiers when classifying a sample. According to this approach, for each training sample, the combiner estimates the conditional probability of each class, given the set of labels provided by the ensemble classifiers. In this way, it is possible to automatically derive the combining rule through the estimation of the conditional probability of each class. Moreover, it is also possible to identify redundant classifiers, i.e. classifiers whose outputs do not influence the output of the combiner. In fact, if the behavior of such classifiers is very similar to that of other classifiers in the ensemble, then they may be discarded without affecting the overall performance of the combiner. In such a way the main drawback of the combining methods discussed above can be overcame. In [5] a Bayesian Network (BN) [11] has been used to automatically infer the joint probability distributions between the outputs of the classifiers and the class label. The BN learning has been performed by means of an evolutionary algorithm using a direct encoding scheme of the BN structure.

In this paper we present a new classification system that exploits the advantages of the two aforementioned approaches. The goal is to build a high performance classification system that uses a small number of classifiers and is able to deal with large data sets. For this purpose, we built a two–module system that combines the BoostCGPC algorithm [6] with the BN based approach to classifier combination [5]. The proposed system allows us to strongly reduce the

number of classifiers in the ensemble. More specifically, such result is achieved by following two different approaches: the boostCGPC evolves diverse classifiers (decision trees) by means of a boosting technique; the BN module evaluates classifiers diversity by estimating the statistical dependencies of the responses they provide. Such estimate is used to select, among the classifiers provided by the BoostCGPC module, a small number of them. Moreover, the responses provided by the selected classifiers are effectively combined by means of a rule learned by the BN module.

The effectiveness of the proposed system, has been tested by performing several experiments. The obtained results have been compared with those obtained by the BoostCGPC approach [6] and with those achieved by using the K2 algorithm [4]. Moreover, the effectiveness of the devised approach as pruning strategy has been tested by comparing its results with those obtained by the Pareto optimal pruning strategy [10].

2 System Architecture

The proposed system consists of two main modules: the first one builds an ensemble of decision tree classifiers (experts) by means of the BoostCGPC algorithm. The second one uses a BN to implement the combining rule that produces the final output of the whole system. More specifically, unknown samples are recognized using a two–step procedure: (i) the feature values describing the unknown sample are provided to each of the ensemble classifiers built by the BoostCGPC module; (ii) the set of responses produced is given in input to the BN module. Such module labels the sample with the most likely class, among those of the problem at hand, given the responses collected by the first module[1]. Also the learning phase requires two steps. In the first step, the BoostCGPC module is trained using a data set containing labeled samples described by their feature values. This learning is carried out, by means of a boosting–based technique (described in Subsection 2.1). In the second step, the responses provided by the set of decision trees built in the first step are used to learn the BN of the second module (Subsection 2.2).

2.1 BoostCGPC Algorithm

The *Boost Cellular Genetic Programming Classifier* [6] algorithm builds GP ensembles using a hybrid variation of the classical distributed island model of GP. GP ensembles offer several advantages over a monolithic GP, i.e. the possibility of coping with very large data sets, more simple and understandable models, robustness and obviously the advantages correlated with a distributed implementation.

[1] Note that the second step does not require any further computation with respect to the Majority Voting rule. In fact, it only needs to read tables storing class probabilities.

Each GP classifier forming the ensemble is built using a cellular GP algorithm (cGP), enhanced with the boosting technique, which runs on each node. cGP runs for T rounds; for every round it generates a classifier per node, exchanges it with the other nodes, and updates the weights of the samples for the next round, according to the boosting algorithm. The selection rule, the replacement rule and the asynchronous migration strategy are specified in the cGP algorithm. Each node generates the GP classifier by running for a fixed number of generations. During the boosting rounds, each classifier maintains the local vector of the weights that directly reflect the prediction accuracy. At each boosting round the hypotheses generated by each classifier are exchanged among all the processors in order to produce the ensemble of predictors. In this way each node maintains the entire ensemble and it can use it to recalculate the new vector of weights. After the execution of the fixed number of boosting rounds, the classifiers are updated.

BoostCGPC adopts the AdaBoost.M2 version of the well-known boosting algorithm introduced by Schapire and Freund for "boosting" the performance of any weak learner, i.e. an algorithm that "generates classifiers which need only be a little bit better than random guessing".

In practice, the original boosting algorithm adaptively changes the distribution of the training set depending on how difficult each example is to classify. Given the number T of trials (rounds) to execute, T weighted training sets $S_1, S_2, \ldots, S_T$ are sequentially generated and T classifiers $C^1, \ldots, C^T$ are built to compute T weak hypotheses h_t. Let w_i^t denote the weight of the example x_i at trial t. At the beginning $w_i^1 = 1/n$ for each x_i. At each round $t = 1, \ldots, T$, a weak learner C^t, whose error ϵ^t is bounded to a value strictly less than $1/2$, is built and the weights of the next trial are obtained by multiplying the weight of the correctly classified examples by $\beta^t = \epsilon^t/(1-\epsilon^t)$ and renormalizing the weights so that $\Sigma_i w_i^{t+1} = 1$. In this way, it focuses on examples that are hardest to classify, as "easy" examples get a lower weight, while "hard" examples, that tend to be misclassified, get higher weights. The boosted classifier gives the class label y that maximizes the sum of the weights of the weak hypotheses predicting that label, where the weight is defined as $log(1/\beta^t)$. The final classifier h_f is defined as follows:

$$h_f = arg\ max\ \left(\sum_t^T log(\frac{1}{\beta^t}) h_t(x, y) \right) \tag{1}$$

Given the training set $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and the number P of processors to use to run the algorithm, we partition the population of classifiers in P subpopulations, one for each processor and draw P sets of samples of size $n < N$, by uniformly sampling instances from S with replacement. Each subpopulation is evolved for k generations and trained on its local sample by running cGP.

After k generations, the individual with the best fitness is selected for participating to vote. In fact the P individuals of each subpopulation having the best fitness are exchanged among the P subpopulations and constitute the ensemble of predictors that will determine the weights of the examples for the next round.

After the execution of the fixed number T of boosting rounds, the overall classifiers composing the ensemble, collected during the different rounds, are used to evaluate the accuracy of the classification algorithm.

2.2 The BN Module

As mentioned in the Introduction, the problem of combining the responses provided a set of classifiers can be handled by estimating the conditional probability of each class given the set of labels provided by the classifiers. Such problem may be effectively solved by using a Bayesian Network (BN). In particular, in [4], a BN has been used for combining the responses of more classifiers in a multi expert system.

A BN is a probabilistic graphical model that allows the representation of a joint probability distribution of a set of random variables through a Direct Acyclic Graph (DAG) [11]. The nodes of the graph correspond to variables, while the arcs characterize the statistical dependencies among them. An arrow from a node i to a node j has the meaning that j is conditionally dependent on i, and we can refer to i as a *parent* of j. In a BN, the i–th node e_i is associated with a conditional probability function $p(e_i|pa_{e_i})$, where pa_{e_i} indicates the set of nodes which are parents of e_i. Such function quantifies the effect that the parents have on that node.
Once the statistical dependencies among variables have been estimated and encoded in the DAG structure, the joint probability of the represented variables $\{e_1, \ldots, e_L\}$ can be described as:

$$p\left(e_1, \ldots, e_L\right) = \prod_{e_i \in E} p(e_i|pa_{e_i}) \tag{2}$$

In the classifier ensemble framework, this property can be used to infer the true class c of an unknown sample when the responses of the ensemble classifiers are known, if we consider c as a variable in the joint probability of Eq. (2). In fact,

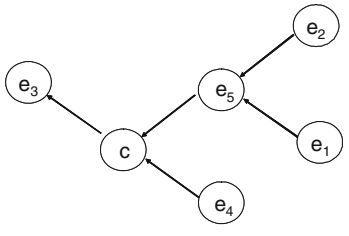


Fig. 1. An example of a BN. The sets $\mathbf{pa_{e_5}} = \{\mathbf{e_1}, \mathbf{e_2}\}$, $\mathbf{pa_c} = \{\mathbf{e_4}, \mathbf{e_5}\}$ and $\mathbf{pa_{e_3}} = \{\mathbf{c}\}$ respectively represents the parent sets of the nodes e_5, c and e_3. The DAG structure induces the factorization of the joint probability $\mathbf{p(c, e_1, e_2, e_3, e_4, e_5) = p(e_3|c)p(c|e_4, e_5)p(e_5|e_1, e_2)p(e_1)\ p(e_2)p(e_4)}$. In this case $\mathbf{E_c = \{e_3\}}$, $\mathbf{E_{\overline{c}} = \{e_4, e_5\}}$.

suppose the ensemble consists of L classifiers, then the true class c and the L classifier responses can be modeled as a set of $(L+1)$ variables $\{c, e_1, \ldots, e_L\}$, and the Eq. (2) allows the description of their joint probability as:

$$p\,(c, e_1, \ldots, e_L) = p\,(\,c\,|\,pa_c\,) \prod_{e_i \in E} p\,(\,e_i\,|\,pa_{e_i}\,) \tag{3}$$

The node c may be parent of one or more nodes of the DAG. Therefore, it may be useful to divide the set of DAG nodes that are not parent of c in two groups: the first one, denoted as E_c, contains the nodes having the node c among their parents, and the second one, denoted as $E_{\overline{c}}$, the remaining ones. With this assumption, Eq. (3) can be rewritten as:

$$p(c, e_1, \ldots, e_L) = p(c|pa_c) \prod_{e_i \in E_c} p(e_i|pa_{e_i}) \prod_{e_i \in E_{\overline{c}}} p(e_i|pa_{e_i}) \tag{4}$$

As will be shown in the following section, this property allows a BN to recognize a given sample only considering the responses provided by classifiers represented by the nodes that are directly linked to the class node. For instance, the BN shown in Fig. 1 considers only the responses of the experts e_3, e_4 and e_5, while the experts e_1 and e_2 are not taken into account. Thus, this approach allows to detect a reduced set of relevant experts, namely the ones connected to node c, whose responses are actually used by the combiner to provide the final output, while the set $E_{\overline{c}}$ of experts, which do not add information to the choice of $\widehat{c}$, are discarded.

Using a BN for combining the responses of a set of classifiers requires that both the network structure, which determines the statistical dependencies among variables, and the parameters of the probability distributions be learned from a training set of examples. The structural learning is aimed at capturing the relation between the variables, and hence the structure of the DAG. It can be seen as an optimization problem which requires the definition of a search strategy in the space of graph structures, and a scoring function for evaluating the effectiveness of candidate solutions. A typical scoring functions is the posterior probability of the structure given the training data [3]. Once the DAG structure has been determined, the parameters of the conditional probability distributions are computed from training data.

The exhaustive search of the BN structure which maximizes the scoring function is a NP-hard problem. For this reason, greedy algorithms are used to search for suboptimal solutions by maximizing at each step a local scoring function which takes into account only the local topology of the DAG. To overcome this problem, we use an alternative approach in which the structure of the BN is learned by means of an Evolutionary algorithm. The algorithm is based on a specifically devised data structure for encoding DAG, called *multilist* (ML), which allows an effective and easy implementation of the genetic operators. Further details about ML data structure and the genetic operators can be found in [5].

3 Experimental Results

The proposed approach has been tested on five data sets: *Census*, *Segment*, *Adult*, *Phoneme* and *Covtype*. The size and class distribution of these data sets are described in Table 1. They present different characteristics in the number and type (continuous and nominal) of attributes, two classes versus multiple classes and number of samples. Each dataset has been divided, as usual, in a training set (2/3 of the original data) and in a test set (1/3 of the original data).

All the experiments have been performed on a Linux cluster with 16 Itanium2 1.4GHz nodes each having 2 GBytes of main memory and connected by a Myrinet high performance network. The BoostGCPC module used standard GP parameters (prob. of crossover=0.8, prob. of mutation=0.1, maximum depth=17, no parsimony factor) and a population of 100 individuals for node. The original training set has been partitioned among 5 nodes and 10 rounds of boosting, with 100 generations for round, have been performed in order to produce 50 classifiers. It is worth to remember the algorithm produce a different classifier for each round on each node.

All results were obtained by averaging over 30 runs. For each run of the BoostCGPC module, a run of the BN module has been carried out. Each BN run has been performed by using the responses, on the whole training set, provided by the classifiers learned in the corresponding BoostCGPC run. The results on the test set has been obtained by first submitting each sample to the learned decision trees ensemble. Then the ensemble responses have been provided to the learned BN. Finally, the BN output label has been compared with the true one of that sample.

The results achieved by our approach (hereafter BN-Boost-CGPC) have been compared with those obtained by the BoostCGPC approach, which uses the wighted majority rule (Eq. 1) for combining the ensemble responses. Moreover, in order to test the effectiveness of the evolutionary learning performed by the second module of the proposed system, we also compared our results with those obtained by a standard algorithm for learning Bayesian Networks, namely the K2 algorithm [4]. Such a algorithm uses a hill climbing technique to learn Bayesian Networks from data. With the aim of performing a fair comparison, we adopted for the K2 algorithm the same scoring function used by our system for evaluating the quality of the network structure. For each dataset, these BNs have been learned on the responses provided by the set of classifiers supplied by the first

Table 1. The data sets used in the experiments

datasets	attr.	samples	classes
Adult	14	48842	2
Census	4	299285	2
Phoneme	5	5404	2
Segment	36	2310	6
Covtype	54	581012	7

module of our system on the training set. The trained BNs have been tested on the responses given by the just mentioned classifiers, obtained on the test set.

Comparison results are shown in Tab. 2. The second column shows the cardinality of the ensembles taken into account (10, 20 and 50 classifiers). The ensembles made of 10 and 20 classifiers have been obtained by considering respectively the first 10 and 20 classifiers generated by the BoostCGPC algorithm. For each considered method, the table reports the training and test error. Column 5 contains the number of classifiers actually used by our approach, i.e. only the classifiers that are directly connected to the class label node in the DAG. Note that for the other methods such number has not been reported since it coincides with the number of classifier making up the ensemble (10, 20 or 50). In order to statistically validate the comparison results, we performed the two–tailed t–test($\alpha = 0.05$) over the 30 carried out runs. The values in bold in the test error columns highlight, for each dataset, the results which, according to the performed test, are significantly better with respect to the second best result (such results are starred). The proposed approach, for all considered datasets, achieves better performance than those obtained by the two methods used for the comparison. It is worth to remark that such results are always achieved by using only a small number of the available classifiers.

In order to test the effectiveness of the ensemble pruning performed by our system, we compared its results with those obtained by the Pareto optimal pruning strategy [10]. Such approach considers, for each couple of classifiers in the original ensemble, two quality measures: the average train error and a pairwise diversity measure. These couples of values, can be plotted in a two-dimensional

Table 2. Comparison results. Bold values represent the best statistically significant results, while starred values represent the second best results. The Columns with the headers tr and ts respectively contain the train and test errors.

Dataset	ens.	BN-BoostCGPC			BoostCGPC		K2-BN	
		tr	ts	# sel.	tr	ts	tr	ts
Adult	10	15.03	**15.05**	3.05	17.24	17.38	17.16	17.34⋆
	20	15.70	**15.65**	3.05	16.99	17.11⋆	17.55	17.83
	50	13.19	**13.53**	3.90	14.43	14.33⋆	17.66	18.29
Census	10	4.72	**4.85**	3.50	5.27	5.27⋆	5.45	5.42
	20	4.73	**4.87**	4.25	5.24	5.24⋆	5.00	5.40
	50	4.09	**4.20**	3.65	4.97	5.08⋆	4.50	5.20
Covtype	10	33.83	**34.05**	3.15	35.97	35.83⋆	37.23	38.37
	20	33.06	**33.29**	3.75	34.73	34.72⋆	36.55	37.00
	50	30.80	**31.00**	3.50	32.52	32.51⋆	33.04	33.94
Phoneme	10	17.97	**18.92**	3.05	19.14	19.84	19.52	19.72⋆
	20	17.10	**17.82**	3.86	17.92	18.37⋆	19.03	19.35
	50	15.81	**16.12**	3.21	16.85	17.36⋆	18.73	19.33
Segment	10	12.08	**12.48**	2.25	17.33	18.28	13.52	14.43⋆
	20	10.80	**11.50**	2.55	15.24	16.14	12.33	13.30⋆
	50	11.08	**11.46**	2.85	14.15	14.90	11.69	12.87⋆

Table 3. Comparison results for the selection strategies. Bold values represent the best statistically significant results, while starred values represent the second best results.

Dataset	ens.	BN-Boost		Pareto optimal			
				geno		pheno	
		error	#sel.	error	#sel.	error	#sel.
Adult	10	**15,05**	3,05	17,25	4,95	17, 24*	5,20
	20	**13,53**	3,90	17,15	9,75	16, 99*	13,05
	50	**13,53**	3,90	17,15	9,75	16, 99*	13,05
Cens	10	**4,85**	3,50	5, 42*	4,90	5,42	5,75
	20	**4,87**	4,25	5, 40*	7,05	5,40	10,40
	50	**4,20**	3,65	5, 38*	9,65	5,39	16,60
Covtype	10	**34,05**	3,15	36, 01*	5,10	36,01	6,55
	20	**33,29**	3,75	35, 15*	7,65	35,38	9,65
	50	**32,00**	3,50	34, 33*	9,35	34,71	14,70
Phoneme	10	**18,92**	3,05	20,29	5,50	20, 09*	5,85
	20	**17,82**	3,86	20,04	6,70	19, 59*	9,10
	50	**16,12**	3,21	19,79	8,90	19, 51*	10,75
Segment	10	**12,48**	2,25	30, 14*	5,90	30,74	5,40
	20	**11,50**	2,55	29,11	7,85	28, 38*	10,10
	50	**11,46**	2,85	28,52	9,20	27, 59*	14,45

space, where every pair of classifiers is represented by a dot. At this point we can imagine that the most desirable pairs of classifiers are those represented by the dots making up the Pareto front of the whole set of classifier pairs. Note that the Pareto front contains all non-dominated points of the plot. A point i is non-dominated if and only if there is no other point j, so that j is better than i on both quality measures. As concerns the diversity measure, we taken into account two different measures, better described in [7]: a genotypic measure that evaluates the structural diversity between the two trees representing the couple of classifiers to be assessed; a phenotypic measure based on Kappa statistics, which gives a score of how much homogeneity there is in the responses provided by two classifiers. The comparison results are shown in Table 3. Also in this case the results have been statistically validate by means of the two–tailed t–test($\alpha = 0.05$) over the 30 carried out runs and the best statistically significant results are marked in bold. From the table it can be seen that for all the datasets, our method outperforms the Pareto optimal approach although it selects a significant minor number of classifiers.

4 Conclusions

We presented a new framework for improving the performance of classifier ensemble, by means of an effective pruning algorithm based on Bayesian networks. The framework consists of two modules: an ensemble-based Genetic Programming system, which produces a high performing ensemble of decision tree classifiers, and a Bayesian Network approach to perform classifier selection.

The effectiveness of the proposed system has been tested by comparing the accuracy of the framework with those obtained by the BoostCGPC approach and with those achieved by using a Bayesian Network learned by using the K2 algorithm. In addition, in order to validate the effectiveness of the approach as pruning strategy, it has been compared with the Pareto optimal pruning strategy. For all the datasets, our method obtains better results than the other methods both in terms of accuracy and of the number of classifiers selected, confirming the goodness of its usage as a pruning strategy. Future works will include the comparison with other state-of-the-art methods and the exploration of the overhead in terms of execution time our method requires.

Acknowledgments. This research work has been partially funded by the MIUR project FRAME, PON01-02477.

References

1. Banfield, R., Hall, L., Bowyer, K., Kegelmeyer, W.: Ensembles diversity measures and their application to thinning. Information Fusion 6, 49–62 (2005)
2. Cantú-Paz, E., Kamath, C.: Inducing oblique decision trees with evolutionary algorithms. IEEE Trans. on Evolutionary Computation 7(1), 54–68 (2003)
3. Cooper, G.F., Herskovits, E.: A bayesian method for the induction of probabilistic networks from data. Machine Learning 9(4), 309–347 (1992)
4. De Stefano, C., D'Elia, C., Scotto di Freca, A., Marcelli, A.: Classifier combination by bayesian networks for handwriting recognition. Int. Journal of Pattern Rec. and Artif. Intell. 23(5), 887–905 (2009)
5. De Stefano, C., Fontanella, F., Marrocco, C., Scotto di Freca, A.: A Hybrid Evolutionary Algorithm for Bayesian Networks Learning: An Application to Classifier Combination. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., Neri, F., Preuß, M., Togelius, J., Yannakakis, G.N. (eds.) EvoApplicatons 2010, Part I. LNCS, vol. 6024, pp. 221–230. Springer, Heidelberg (2010)
6. Folino, G., Pizzuti, C., Spezzano, G.: Gp ensembles for large-scale data classification. IEEE Trans. on Evolutionary Computation 10(5), 604–616 (2006)
7. Folino, G., Pizzuti, C., Spezzano, G.: Training distributed gp ensemble with a selective algorithm based on clustering and pruning for pattern classification. IEEE Trans. Evolutionary Computation 12(4), 458–468 (2008)
8. Freund, Y., Shapire, R.: Proceedings of the 13th Int. Conference on Machine Learning
9. Gagné, C., Sebag, M., Schoenauer, M., Tomassini, M.: Ensemble learning for free with evolutionary algorithms? In: GECCO, pp. 1782–1789 (2007)
10. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley-Interscience (2004)
11. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
12. Zhou, Z.H., Tang, W.: Selective Ensemble of Decision Trees. In: Wang, G., Liu, Q., Yao, Y., Skowron, A. (eds.) RSFDGrC 2003. LNCS (LNAI), vol. 2639, pp. 476–483. Springer, Heidelberg (2003)

A Multi-parent Search Operator for Bayesian Network Building

David Iclănzan

Department of Computer Science, Babe s-Bolyai University,
Kogălniceanu no. 1, 400084, Cluj-Napoca, Romania
david.iclanzan@gmail.com

Abstract. Learning a Bayesian network structure from data is a well-motivated but computationally hard task, especially for problems exhibiting synergic multivariate interactions. In this paper, a novel search method for structure learning of a Bayesian networks from binary data is proposed. The proposed method applies an entropy distillation operation over bounded groups of variables. A bias from the expected increase in randomness signals an underlaying statistical dependence between the inputs. The detected higher-order dependencies are used to connect linked attributes in the Bayesian network in a single step.

1 Introduction

A Bayesian networks is a probabilistic graphical model that depicts a set of random variables and their conditional independence via a directed acyclic graph. It represents a factorization of a multivariate probability distribution that results from an application of the product theorem of probability theory and a simplification of the factors achieved by exploiting conditional independence statements of the form $P(A|B, X) = P(A|X)$, where A and B are attributes and X is a set of attributes.

The represented joint distribution is given by:

$$P(A_1, \ldots, A_n) = \prod_{i=1}^{n} P(A_i|par(A_i)) \tag{1}$$

where $par(A_i)$ denotes the set of parents of attribute A_i in the directed acyclic graph that is used to represents the factorization.

Bayesian networks provide excellent means to structure complex domains and to draw inferences. They can be acquired from data or be constructed manually by domain experts (a tedious and time-consuming task).

One of the most challenging task in dealing with Bayesian networks is learning their structures, which is an NP-hard problem [1,2]. Most algorithms for the task of automated network building from data, consist of two ingredients: a search method that generates alternative structures and an evaluation measure or scoring function to assess the quality of a given network by calculating the goodness-of-fit of a structure to the data.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 246–255, 2012.

Due to the computational cost implications [2], most of the algorithms that learn Bayesian network structures from data use a heuristic local search to find a good model, trading accuracy for tractability and efficiency. Exact Bayesian network learning has a $O(n2^n)$ complexity, thus it is feasible up to 30 variables [3].

Heuristic methods, at each step apply some search operators like perturbation or solution mixing, to some current network structure(s), exploring their neighborhoods. After evaluating the new solutions, they promote changes that result in the improvement of some discriminative metric.

Because these search methods alter only a few arcs at the time, they can hardly find and express multivariate interactions that only manifest at a synergic level like the parity function. Here, adding edges between less than k nodes, where k is the size of the block containing the multivariate interaction, will not result in any improvement, thus are hard to discover by methods closely following the discriminative metric gradient.

In this paper we propose a linkage-detection method that is able to select all relevant parents for an attribute in one step, by finding and expressing even relationships not manifesting at pairwise level. Our method exploits the property of the exclusive or (XOR) operator to produce randomness from non-deterministic sources. We search for groups of variables where entropy distillation does not occur, signaling a non-determinism in the source - statistical dependence between the variables.

Albeit a costly search for the groups of variables must be performed, this approach enables the correct detection of Bayesian network, unattainable by simple heuristic search methods.

2 Detecting Higher-Order Dependencies

Binary problems of real interest may have many variables with complicated multivariate interactions among them. The dependency of a binary variable X_e on a (noisy) feature expressed by several other variables of the problem can be formalized as follows:

$$\begin{aligned} \text{if } \ & f_b(X_{v1}, X_{v2}, \ldots, X_{vl}) \ \ [\text{and } noise(X)] \\ & \text{then} \qquad\qquad X_e = b \\ & [\text{else} \qquad\qquad X_e = \overline{b}] \end{aligned}$$

where f_b is an arbitrary deterministic boolean function of l binary variables, which analyzes if the input variables satisfy a certain feature or not. $\overline{b}$ is the bitwise negation of b. As the relation must not be fully specified, the else branch is optional. The optional boolean $noise(X)$ function can be used to introduce stochasticity to the relation, to model external influences or factors which are not directly considered when evaluating the feature. This boolean function may prevent the expression of the feature even if the conditions are present, thus adding noise to the relation.

For these kind of problems, pairwise dependencies might be very small or lacking altogether. Therefore, finding the correct dependency structure is a very hard task. Prospective methods must combine an extensive higher order model search guided by a criteria that evaluates the quality of the model in rapport with the evidence, like the Minimum Description Length (MDL) principle [4] or Bayesian-Dirichlet metric [5].

The complexity of the model determination is a product of the complexity of the search and candidate model evaluation.

For problems where statistical dependence can only be detected by considering at least k variables, the search must enumerate at least all combinations of variables taken k at the time.

An ordered tuple of binary random variables $X = (X_1, X_2, \ldots, X_n)$ is independent iff the joint distribution $Pr(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$ and the product of the independent ones $\prod_{k=1}^{n} Pr(X_k = x_k)$ is equal for all $x = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$.

Relative entropy or Kullback-Leibler divergence [6] can be used to measure the "distance" between these two distributions:

$$D_{KL}(p||q) = \sum_{x \in X} p(x) \log_2 (\frac{p(x)}{q(x)}) \tag{2}$$

Measuring the D_{KL} between the observed joint distribution of some variables and the product of independent joint distribution one can measure the information gain by considering a group of variables linked. The complexity of calculating D_{KL} is exponential in k with a base equal to the cardinality of the random variables. Thus, for the binary case the complexity is $O(2^k)$.

While the burden of the combinatorial search can not be obviated, in the following we consider ways in which the complexity of the model discriminatory function can be heavily reduced from the exponential complexity.

3 Entropy Distillation Based Multivariate Dependency Detection

Most practical sources of randomness, be it hardware or pseudo-random number generators, exhibit a certain level of imperfection or bias. A perfectly random bit has an entropy of one bit and bias of 0. To obtain a highly random bit, there are algorithms that combine multiple, streams of imperfect random bits, each with entropy less than one, to create a single bit with entropy one and bias 0. This process is called entropy distillation or entropy extraction.

Exclusive OR (XOR, also denoted by $\otimes$) is commonly used to reduce the bias from imperfectly random bits, provided that the random bits are statistically independent.

The reduction in bias by repeatedly applying the XOR on non-deterministic inputs can be computed using the Piling-Up Lemma [7].

Lemma 3.1. *Let X_i for $i \in \overline{1,n}$ be statistically independent random binary variables, where p_i is the probability that $X_i = 0$ and $\epsilon_i = p_i - 1/2$ are the biases. Then the probability that $X_1 \otimes X_2 \otimes \ldots \otimes X_n = 0$ is*

$$\frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} \epsilon_i \tag{3}$$

Note that as biases $\epsilon_i \in [0, 1]$ their product is a monotonically decreasing function. If any of the ϵ's is zero, that is, one of the binary variables is unbiased, the resulting probability function will be unbiased. Also, performing an XOR with a constant variable having $p_i = 0$ or $p_i = 1$ i.e a maximum bias of $\epsilon_i = \pm 1/2$ will not reduce the bias.

In our algorithm we will apply the XOR operation in a sequential manner, performing in each step the operation between a variable X_i and the result $Y = X_1 \otimes X_2 \otimes \ldots \otimes X_{i-1}$ of the repeated XOR up to that variable i. Therefore, we take a closer look on the expected result of XOR for two variables.

Lemma 3.2. *If X and Y are independent random binary variables with expectations $E(X) = \mu$ and $E(Y) = \nu$ then*

$$E(X \otimes Y) = \mu + \nu - 2\mu\nu \tag{4}$$

$$= \frac{1}{2} - 2(\mu - \frac{1}{2})(\nu - \frac{1}{2}) \tag{5}$$

Proof. Following from the logical table of the XOR, for two bits a and b, $a \otimes b$ equals 1 if $a = 0$ and $b = 1$ or if $a = 1$ and $b = 0$.

Thus, $E(X \otimes Y)$ can be written as

$$\begin{aligned} E(X \otimes Y) &= (1 - \mu)\nu + \mu(1 - \nu) \\ &= \mu + \nu - 2\mu\nu \\ &= \mu + \nu - 2\mu + \frac{1}{2} - \frac{1}{2} \\ &= \frac{1}{2} - 2\left[\mu\nu - \frac{\mu}{2} - \frac{\nu}{2} + \frac{1}{4}\right] \\ &= \frac{1}{2} - 2(\mu - \frac{1}{2})(\nu - \frac{1}{2}) \end{aligned}$$

3.1 XOR Based Multivariate Dependency Detection

The Piling-Up Lemma is successfully used in linear cryptanalysis to construct linear approximation to the action of non-linear block ciphers. In this application, the X_i-s are approximations to the substitution-boxes of block ciphers for which the biases are trivial to measure. The attack relies on performing a costly search for finding combinations of input and output values that have very high biases i.e probabilities very close of zero or one.

Similarly, we perform a search to find groups of variables for which the actual probability mass of the result obtained by performing the XOR greatly differs

from the value predicted by the Piling-Up Lemma. For these cases the high bias must come from the fact that the assumption of non-determinism is not satisfied. Thus, there is an underlaying (higher-order) statistical dependence between the inputs.

The proposed metric has a great complexity advantage, as performing k consecutive XOR operations in linear. While the approach is efficient in detecting multivariate dependences, we still have to perform an ample search to find the higher-order groups of variables that are dependent. In the next section we apply this multivariate dependence detection technique to determine all relevant parents for the Bayesian network building task.

4 Bayesian Network Building

In Bayesian network building, the goal is to decide the set $par(A_i)$ for each attribute, with the restriction, that adding the edges between and attribute and its parents must not result in a cycle.

To detect all dependencies, up to a predefined bounded size k in one step, for each attribute we compute the repeated $\otimes$ between the attribute and all possible combinations of other variables up to the threshold k. For each combination, we compute the difference between the percentage of zeros in the result as predicted by the Pilling-Up lemma and the actual outcome percentage. For each attribute, we retain the combinations that yield the biggest discrepancies.

In the network building phase, we process the attributes in a random order. For each attribute A_i, we sequentially assign the potential parent set $par_j^*(A_i)$ to be the j^{th} combination of variables with the highest bias, as quantified with the help of the Pilling-Up lemma in the previous step. In this way we process a prefixed top S_{nr} interacting subsets for each attribute. For every subset, we process each potential parent $p^*(A_i)$, $p^*(A_i) \in par_j^*(A_i)$, and if adding an edge between the attribute and its potential parent does not result in a cycle, $p^*(A_i)$ becomes a parent of A_i: $par_j(A_i) = par_j(A_i) \cup p^*(A_i)$. From all the obtained and tested parent subsets for each attribute, we choose attribute and its parents that maximizes a given discriminative scoring function, in our case the Bayesian Dirichlet metric [8].

The search stops when we determined the parents of each attribute, or when considering the extension of the network does not result in improvements. The outline of this parent search procedure is outlined in Algorithm 1.

4.1 Test Suite

To assess the performance of the proposed search method, we built some artificially generated test samples that contain various types of multivariate interactions. We consider 10 variables $X_1, \ldots, X_{10}$, sampled 5000 times.

Algorithm 1. Constructing a Bayesian network that is able to capture higher-order interactions up to a prefixed order k

```
1  BN ← EmptyNetwork();
2  foreach attribute A, iterating by i do
3      foreach e possible combinations of variables that do not contain A_i, up to
       size k, iterating by j do
4          Measure the entropy distillation bias between A_i and e_j and retain the
           S_nr combinations with highest biases in M(i,:);
5  repeat
6      for i=randompermutation(1:n) do
7          if HasParents(i) then
8              continue;
9          for j=1:S_nr do
10             par*(A_i) ← M(i,j);
11             par(A_i) ← EliminateCycles(par*(A_i));
12             BN* ← ExtendNetwork(BN, A_i, par(A_i));
13             if Score(BN*) > Score(BN) then
14                 BN ← BN*;
15 until No improvement was found;
16 return BN;
```

The *first* data set contains two highly noisy features:

- A highly noisy conditioning, where whenever three out of the four first variables are one, X_5 is also set to 1 with a probability of 0.5:

$$\text{if } (sum([X_1, X_2, X_3, X_4]) == 3) \text{ and } (rand \leq 0.5) \text{ then } X_5 = 1$$

- A noisy feature based on a parity function conditioning where if variables X_6, X_7, X_9, X_{10} have an even number of ones, X_8 is set to 0 with a 0.8 probability:

$$\text{if } (parity([X_6, X_7, X_9, X_{10}]) == true) \text{ and } (rand \leq 0.8) \text{ then } X_8 = 0$$

In the *second* dataset we reduce the amount of explicit noise but introduce an overlap between the two features, which are:

- We have the noisy conditioning, where whenever exactly half of a group of six variables are 1, X_5 is also set to 1 with a probability of 0.95:

$$\text{if } (sum([X_1, X_2, X_3, X_4, X_9, X_{10}]) == 3) \text{ and } (rand \leq 0.95) \text{ then } X_5 = 1$$

– Again a noisy feature based on a parity function conditioning:

$$\text{if } (parity([X_6, X_7, X_9, X_{10}]) == true) \text{ and } (rand \leq 0.9) \text{ then } X_8 = 0$$

In the *third* dataset we introduce an interplay between the features, where the realization of the first feature may inhibit the realization of the second one:

– We use again the first conditioning, from dataset one.
– A feature which may be short circuited by the realization of the first feature: if $X_5 == 1$, the feature regarding X_8 is not expressed.

$$\text{if } (sum([X_6, X_7, X_9, X_{10}]) == 3) \text{ and } (X_5 == 0) \text{ then } X_8 = 0$$

4.2 Results

For each test case, we generated 50 instances and tested the proposed method against the classical Bayes network model building K2 algorithm [9], which extends a current model by performing one arc operation at the time. The allowed in degree in the classic search and the k parameter in the proposed method was set to 6, thus both methods could consider up to 6 parents. The number of analyzed possible parent sets S_{nr} was set to 5.

For each batch of 50 runs, we recorded the best network found, its score, the worst and the average score. Because the data is stochastically generated and incorporates noise, the exact quantity of this values is of a little importance. The same network structure will score differently when evaluated on different noisy samples. Nevertheless, these values may be used to make qualitative assessments, in the cases where the worst result of one method surpasses the best network score or the average score found by the other method.

More important aspect regards the methods ability to extract the same structure from different samples of noisy data. We measure this robustness by comparing the best and worst scoring network out of each batch of 50 runs. If the adjacency matrix of the two networks is not similar (one can not be transformed into the other one by using only row and column swapping), implies that the search method may find different network topologies on different runs.

The numerical scores are presented in Table 1. The plot of the best networks found for each of the three cases are presented in Figures 1, 2, 3.

In the first case, where there is a high amount of noise, the classical approach can not detect the real structure, the network is filled with spurious connections where often an attribute is accounted as the parent of all other attributes following after. Observe for example in Figure 1, that Node 1 is attributed as parent for all other nodes. On the other hand, even with such a high amount of noise, the extended multi-parent search is able to detect the correct topology of the network.

For the second dataset it is expected that the classical approach is not able to detect the parity, multivariate interaction as it would need to add at least six arcs at once to reveal this interaction. Furthermore, as this feature overlaps with

Table 1. The performance of the proposed and classical methods on the three test suites. The multi-parent extended search worse results are better than the best scores obtained by the classical search method in all cases.

	Best	Worst	Average	Std.	Robust
Test suite 1					
Classic	-34128.06	-34269.11	-34204.73	32.89	No
Extended	-33662.24	-33826.25431	-33748.88	37.67	Yes
Test suite 2					
Classic	-33876.23	-34040.17	-33950.13	36.63	Yes
Extended	-33063.69	-33308.58	-33192.06	52.87	Yes
Test suite 3					
Classic	-34172.68	-34298.18	-34228.73	27.03	No
Extended	-33915.10	-34065.95	-33982.97	29.87	Yes

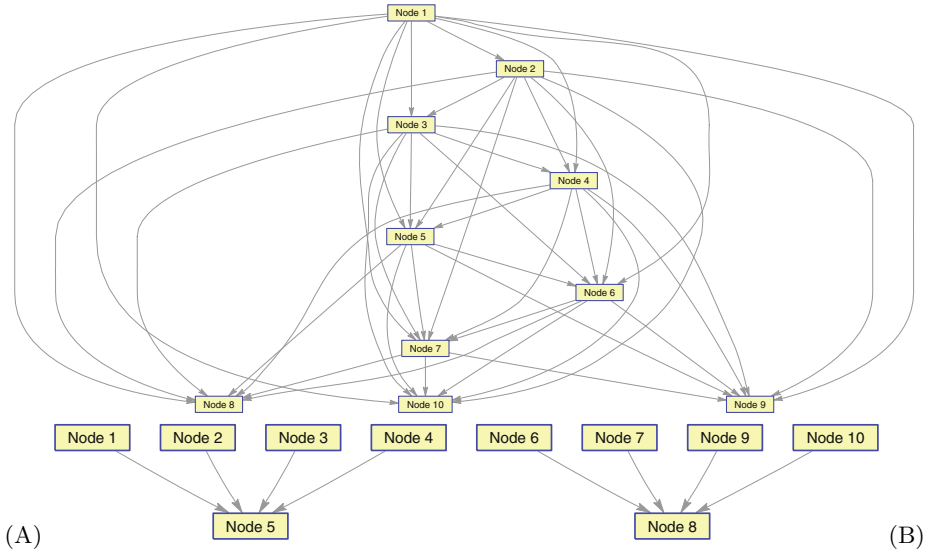


Fig. 1. Best networks found by the classical method (A) and the multi-parent extended search (B) on the first test suite

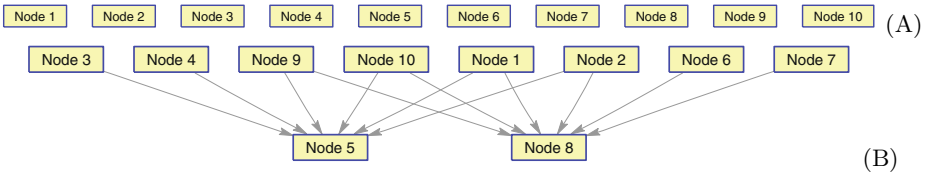


Fig. 2. Best networks found by the classical method (A) and the multi-parent extended search (B) on the second test suite

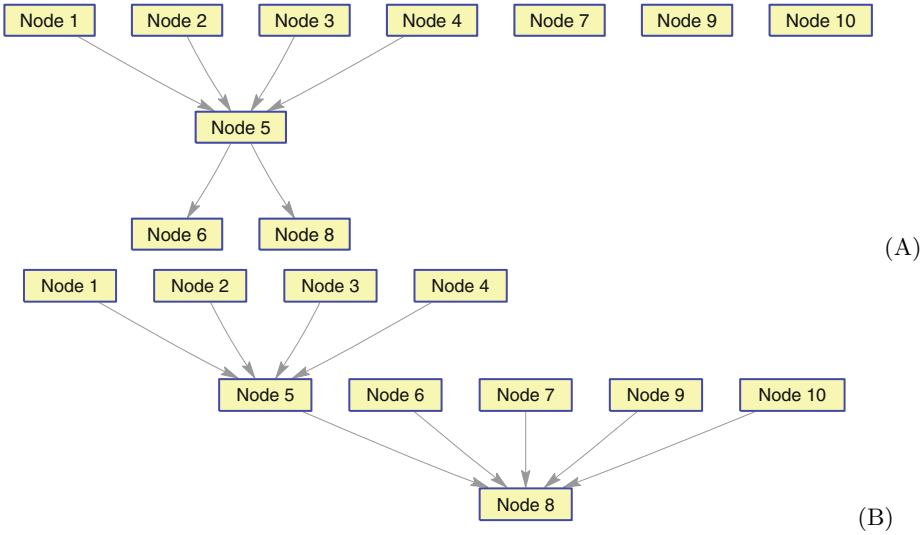


Fig. 3. Best networks found by the classical method (A) and the multi-parent extended search (B) on the third test suite

the other feature which also spans across six variables, the method is unable to account for useful relations and returns the empty network, without edges, in all cases. Please note by looking at the best and worst score in Table 1 for test suite 2, how the same empty network may score differently when presented with different test data. The proposed method is again able to find the correct structure, as we allowed the feature space exploration up to six combined variables, which is also the length of the highest multivariate relation.

On the third case, the classical method is able detect the interactions influencing attribute 5 and its relation to attribute 8, while failing to model the synergic interaction of the other variables. Sometimes, as depicted in Figure 3 A, it reports attribute 5 as linked to other variable different from attribute 8, but this result is rarely achieved. By modeling all interactions up to size six in the feature space, the multi-parent search is able to correctly decipher the interplay between the two features.

For all cases, as shown above, the extended search found qualitatively better networks; the worst scoring results of the proposed method were always better than the best results returned by the classical method. As it does not contain stochastic components, the proposed showed robustness, finding the same topology on different runs.

5 Conclusions

Usual Bayesian network building starts by exploiting pairwise dependencies. When no such relations are available a successful approach must do k-wise multivariate interaction search.

In this paper a search algorithm for constructing Bayesian networks from binary data was developed, where all dependencies of each attribute is detected in one step. The proposed method has demonstrated a great ability to identify simpler and synergic multivariate interactions even in the case of noisy feature interplay, where considering one edge addition at the time is fruitless.

While it uses a small number of model evaluations and it is much more effective than doing greedy search using a k-wise stochastic edge search operator, the extended multi-parent search is still very costly in terms of building and evaluating all combinations of variables, having an $O(n^k)$ complexity. Fortunately, backtracking algorithms are very easy to parallelize as processing different paths in the search tree is an embarrassingly parallel task, with no communication overhead [10,11]. Parallel backtracking scales very well with the number of available processors. Therefore, future effort will focus on parallelizing the higher-order dependency detection search.

Acknowledgments. This research is supported by the Sectoral Operational Program for Human Resources Development 2007-2013, co-financed by the European Social Fund, within the project POSDRU 89/1.5/S/60189 with the title "Postdoctoral Programs for Sustainable Development in a Knowledge Based Society". We also acknowledge the financial support of the Sapientia Institute for Research Programs (KPI).

References

1. Bouckaert, R.: Properties of Bayesian belief network learning algorithms. In: Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference, July 29-31, p. 102. Morgan Kaufmann (1994)
2. Chickering, D., Geiger, D., Heckerman, D.: Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research (November 1994)
3. Koivisto, M., Sood, K.: Exact bayesian structure discovery in bayesian networks. J. Mach. Learn. Res. 5, 549–573 (2004)
4. Rissanen, J.: Modelling by the shortest data description. Automatica 14, 465–471 (1978)
5. Pelikan, M.: Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms. Springer (2005)
6. Cover, T.M., Thomas, J.A.: Elements of information theory. Wiley-Interscience, New York (1991)
7. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
8. Heckerman, D., Geiger, D., Chickering, D.: Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning 20(3), 197–243 (1995)
9. Cooper, G.F., Herskovits, E.: A bayesian method for the induction of probabilistic networks from data. Mach. Learn. 9(4), 309–347 (1992)
10. Kouril, M., Paul, J.L.: A parallel backtracking framework (bkfr) for single and multiple clusters. In: Proceedings of the 1st Conference on Computing Frontiers, CF 2004, pp. 302–312. ACM, New York (2004)
11. Herley, K.T., Pietracaprina, A., Pucci, G.: Deterministic parallel backtrack search. Theor. Comput. Sci. 270, 309–324 (2002)

Enhancing Learning Capabilities by XCS with Best Action Mapping

Masaya Nakata[1], Pier Luca Lanzi[2], and Keiki Takadama[1]

[1] Department of Informatics,
The university of Electo-Communications, Tokyo, Japan
m.nakata@cas.hc.uec.ac.jp, keiki@inf.uec.ac.jp
[2] Dipartimento di Elettronica e Informazione,
Politecnico di Milano, Milano, Italy
lanzi@elet.polimi.it

Abstract. This paper proposes a novel approach of XCS called XCS with Best Action Mapping (XCSB) to enhance the learning capabilities of XCS. The feature of XCSB is to learn only *best actions* having the *highest* predicted payoff with the high accuracy unlike XCS which learns actions having the *highest* and *lowest* predicted payoff with the high accuracy. To investigate the effectiveness of XCSB, we applied XCSB to two benchmark problems: multiplexer problem as a single step problem and maze problem as a multi step problem. The experimental results show that (1) XCSB can solve quickly the problem which has a large state space and (2) XCSB can achieve a high performance with a small max population size.

1 Introduction

Learning Classifier Systems (LCSs)[6] are rule-based learning systems that combine Reinforcement Learning[8] with Genetic Algorithm[5]. The aim of LCSs is to acuire *classifiers* as condition-action rules that perform well in any given environmental state by reinforcing and evolving them to match several environmental states through a *generalization* process. Among LCSs, the XCS classifier system[10] is a traditional LCS and can learn the accurate generalized classifiers by employing an *accuracy* of the predicted payoff of classifiers. In the learning process of XCS, it learns all state-action pairs, which is called a *complete map*[10]. The complete map contributes to stably perform the high generalization capabilities of XCS but requires a huge number of the max population size. This is because that XCS needs to generate a lot of variety of classifiers at the first learning stage to find accurate classifiers. For this reason, the condition of classifiers should consist of an enough number of *specific bits* in which XCS does not trigger a *cover-delete* cycle[3]. Here, the cover-delete cycle occurs when the classifiers do not cover all state-action pairs due to the condition has too much specific bits, and reduces the learning capabilities. To learn with the small max population size, XCS requires the strong generalization pressure to prevent the cover-delete cycle. Because the condition has more generalized bits covers more

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 256–265, 2012.

state by itself. However, in this case, the learning capabilities are reduced due to that many overgeneralized classifiers are generated.

In this paper, we propose XCS with Best Action Mapping, which is called XCSB. XCSB leans only a best action in each state to perform the high learning capabilities even with the small max population size, this is called a *best action map*[7]. Here, the best action has the highest predicted payoff in each state. XCSB does not need the strong generalized pressure even with the small max population size since it learns only classifiers has the best action. The features of XCSB are that XCSB acquires the best action map with the accurate generalization in comparison with *strength*-based LCSs as ZCS[9], and that XCSB can adapt not only the single step problems but also the multi-step problems without a supervised learning in comparison with UCS[1]. To investigate the learning capabilities of XCSB, we apply it to Multiplexer problem as a single step problem and Maze problem as a multi step problem.

This paper organized as follows. The next section gives the description of XCS in brief, and Section 3 describes XCSB. Section 4 shows the experiment results of XCSB. Finally, conclusions are given in Section 5.

2 XCS in Brief

A classifier is represented by the *condition-action* rule. In general, a condition C is coded by $C \in \{0, 1, \#\}^L$, where L is the length of condition, and the symbol '#' means the *don't care symbol* which matches all conditions (*i.e.*, 0 or 1). In XCS, the classifier has the following main parameters: 1) the prediction p which represents the average of the reward; 2) the prediction error ϵ which represents the average of the absolute error of the prediction p; 3) the fitness F which represents the accuracy of classifiers.

When XCS recognizes a state as input from the environment, XCS generates a *match set* $[M]$ composed of classifiers and each of the conditions in $[M]$ matches the current state from population $[P]$. If a number of the type of actions in $[M]$ are less than θ_{nma}, the *covering* mechanism takes place to generate the classifiers that have the condition matched to the current state and the action with the different type of actions in $[M]$. In this process, the some parts of conditions in the generated classifiers are replaced to '#' with the probability $P_{\#}$. When two or more classifiers are included in $[M]$, XCS calculates the *prediction array* $P(a_j)$ for each possible actions a_j in $[M]$ by Eq.(1), where $[M] \mid a_j$ represents the classifiers which have the action a_j in $[M]$.

$$P(a_j) = \frac{\sum_{cl_k \in [M]|a_j} p_k \times F_k}{\sum_{cl_k \in [M]|a_j} F_k} \tag{1}$$

After calculating $P(a_j)$, the *action selection* mechanism in XCS selects the one action from $[M]$ according to a proportion to $P(a_j)$. Next, XCS generates an *action set* $[A]$ composed of the classifiers which action is the selected action from $[M]$, and executes the selected action in the environment. After that, XCS sometimes gets the reward r. We count one *step* when this cycle is repeated.

After one step is executed, the several parameters of classifiers in the *previous action set* $[A]_{-1}$ stored one step before is updated. Especially, the prediction p_i are updated by Eq.(2), r is the reward is obtained from the environment; γ is the *discount factor* $(0 \leq \gamma \leq 1)$ which determines the consideration degree of a future reward; β is the *learning rate* $(0 \leq \beta \leq 1)$ controls the speed of learning.

$$p_i \leftarrow p_i + \beta(r + \gamma \, max \, P(a) - p_i) \tag{2}$$

In the case of XCSG, the gradient of the fitness is added to Eq.(2) as Eq.(3) [2] to promptly adapt an environment in the multi-step problem.

$$p_i \leftarrow p_i + \beta(r + \gamma \, max \, P(a) - p_i) \times \frac{F_i}{\sum_{cl_k \in [A]} F_k} \tag{3}$$

After updating the parameters, the *absolute accuracy* κ_i and *relative accuracy* κ'_i of the each classifier in $[A]$ are calculated depend on an error threshold ϵ_0. Finally, the fitness of classifier F_i is updated as: $F_i \leftarrow F_i + \beta(\kappa'_i - F_i)$. Next, GA is executed to evolve the previous action set $[A]_{-1}$ when the average of the time step since GA is lastly executed exceeds the threshold θ_{GA}. In detail, two classifiers are selected as *parents* with the probability proportioned (called *roulette wheel selection*) to their fitness of classifiers in $[A]_{-1}$ and two classifiers are generated as *offspring*. Here, XCSTS is added the *tornament selection* to XCS, and can derive the high learning capabilities and is more robust than XCS[4]. Then, the crossover and mutation are executed with the probabilities χ (cross over rate) and μ (mutation rate). Finally, the two generated offsprings are added to the population.

3 XCS with Best Action Mapping

In general, the learning and generalization capabilities of XCS are higher than that of several *strength*-based LCS[7]. However, the learning capabilities of XCS are reduced when XCS is not given the suitable generalization and specificity pressures depend on the max population size since it acquires the complete map which covers all state-action pairs(Fig.1-a). Because if it is given the stronger specificity pressure than suitable one, XCS cannot enough estimate several parameters of classifiers since alternates the covering and the deletion continually, this cycle is called *cover-delete* cycle[3]. In contrast, if XCS is given the stronger generalization pressure than suitable one, XCS does not trigger the cover-delete cycle but generates the overgeneralized classifiers whose are generalized more than necessary. Especially, to derive the high learning capabilities and speed, XCS needs to get the suitable specificity pressure, since XCS finds the accurate classifier from more variety of classifiers in the first learning stage. For this reason, XCS requires a large max population size to get the suitable specificity pressure in which the cover-delete cycle do not trigger, or XCS has to set the strong generalization pressure with the small max population size at the cost of the learning capabilities.

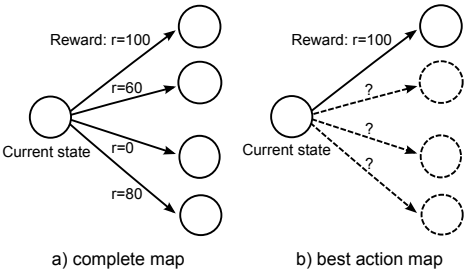


Fig. 1. Representations of complete map and best action map at the current state

To tackle these problems, we propose the XCS with Best Action Mapping, is called XCSB. XCSB acquires the *best action map*[7] which has only best actions lead to the highest prediction in each state(Fig.1-b). Due to this, XCSB can derive the high learning capabilities since XCSB is possible to set the suitable specificity pressure without occurring cover-delete cycle even with the small max population size. XCSB is improved the mechanism of XCS and has the two different mechanisms from XCS as the following; 1) Distinguishing and learning the best action and classifiers whose have the best action; and 2) Tuning the number of called covering depend on the learning states which mean the degree of fixed best action in each states. The following sub-sections explain the these mechanisms of XCSB.

3.1 Description of XCSB

Distinguishing and Learning the Best Action: To distinguishing the best actions, it is possible to identify the actions which has the highest prediction array value represents as $maxP(a)$. However, when the best action is not fixed, *i.e.*, the prediction array for each actions do not enough converge, it is not always true that the action has $maxP(a)$ indicates the best action. To identify correctly the best action, we identify it by comparing with $maxP(a)$ at the nest state. As shown Eqs.(1) and (2), the previous $maxP(a)$ as $maxP(a)_{-1}$ converges to $\gamma maxP(a)$ at the next state, *i.e.*, the $maxP(a)$ converges to the $maxP(a)_{-1}/\gamma$, since each prediction of each classifier in $[A]_{-1}$ also converges to $\gamma maxP(a)$. Here, each best action has to lead the next state which has the $maxP(a)$ is equal to $maxP(a)_{-1}/\gamma$. For this reason, after the selected action is executed, we can identify its action as the best action if $maxP(a)$ is equal or larger than $\zeta \times maxP(a)_{-1}/\gamma$, where the ζ represents the rate that permits the error of convergence of $maxP(a)$, and should be set to larger than γ. Note that we identify it if the given reward is equal or larger than $\zeta \times maxP(a)$ when the terminal criteria are met. Since $maxP(a)$ converges to reward value at that time.

To leaning the best action, XCSB generates not only action set but also a *not action set* $[\bar{A}]$ which is composed of classifiers in $[M]$ whose do not have the selected action. When the executed action is identified as the best action,

the parents in GA are selected from $[A]$ and deleted classifiers are selected from $[\bar{A}]$ to acquire only best action map. Here if $[\bar{A}]$ is empty they are selected from population. On the other hand, when the executed action is not identified as the best action, the parents are selected from $[\bar{A}]$ to find the best action and deleted classifiers are selected from the population.

Tuning the Number of Called Covering: In XCS, the covering operator is called if the number of different actions in $[M]$ is smaller than θ_{nma}. In general, θ_{nma} in XCS is set to the number of actions in the environment. However, to acquire the best action map in XCS calls the covering continually since it does not cover all state-action pairs. For this reason, XCSB tunes the number of called covering depend on the degree of the fixed best action, *i.e.*, θ_{nma} is not fixed value in XCSB. In detail, θ_{nma} is set to 1 (means only best action) to prevent calling covering if the best action is fixed, while θ_{nma} is set to the number of actions in the environment to find a best action if the best action is not fixed. To estimate the degree of fixed best action, we add a new parameter to classifiers, is called *number of action* represent as *nma*. Each nma_i of classifiers cl_i is updated by Eq.(4), where η controles the updated speed, and *nma in env* represents the number of actions in environment, for instance in a binary classification problem *nma in env* is 2. As shown Eq.(4), if the executed action is identified as best action by above distinguishing mechanism, nma_j of each classifiers in $[A]_{-1}$ is converged to 1. Otherwise, it is converge to the number of actions in environment. Note that *nma* of generated classifiers by covering is set to *nma in env* as an initial value.

$$nma_i \leftarrow \begin{cases} nma_i + \eta(1 - nma_i) & \text{if } \ maxP(a) \geq \zeta \times maxP(a)_{-1}/\gamma \\ nma_i + \eta(nma\ in\ env\ - nma_i) & \text{otherwise.} \end{cases} \tag{4}$$

Finally, θ_{nma} in XCSB is calculated from the average of *nma* of each classifier which has the best action in the current state. In detail, in generating $[M]$, XCSB calculates the prediction array to identify the best action before the covering is called. Then, it calculates the average of *nma* of each classifier which has the best action, and θ_{nma} is set to the average value which is rounded off the average of *nma*. Note that θ_{nma} is set to the number of actions in environment if $[M]$ is empty. After that, the covering is called when the number of different action in $[M]$ is smaller than θ_{nma}. XCSB again calculates the prediction array if some generated classifiers by covering is added to $[M]$.

Other Settings: In a single step problem, to acquire only the best action map causes to select the best action incorrectly. Since the generated classifiers by GA, which have a wrong condition-action pair, have the high prediction. For example, there are only two type classifies in $[M]$, one is the correct classifiers which has the best action, the high prediction as 1000 and the high fitness as 1.0, other one is wrong classifiers which also has the high prediction as 1000 and low fitness as 0.1. Then, the both prediction array value is equal to 1000 independent of both fitness values. Due to this, the best action is not always selected. For this reason,

we employ the selection array $S(a)$ for the selection probability to prevent the incorrect selection, and is calculated by Eq.(5). Note that the denominator of $S(a)$ is different from the that of prediction array. In $S(a)$, the selection probabilities are calculated based on the summation of fitness in $[M]$. For example, above cases, the selection probability of the correct classifier is 909 and that of wrong classifier is 90.9. The best action is correctly selected. We employ the $S(a)$ only for a single-step problem and use it for selecting action. Note that XCSB calculates also the prediction array for the update several parameters and the distinguishing the best action.

$$S(a_j) = \frac{\sum_{cl_k \in [M]|a_j} p_k \times F_k}{\sum_{cl_k \in [M]} F_k} \tag{5}$$

4 Experiment

4.1 Single Step Problem

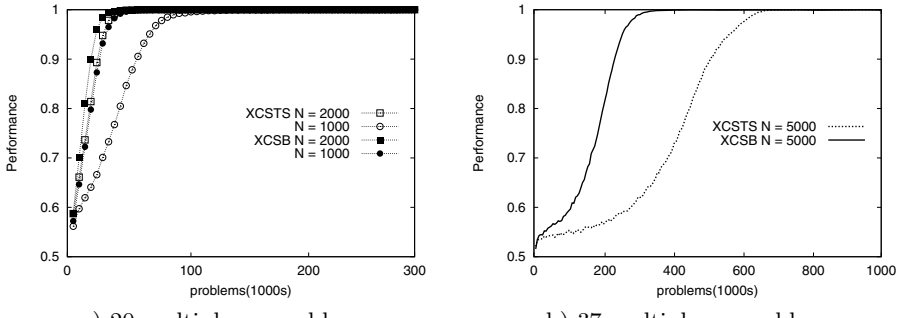
To investigate the effectiveness of XCSB on the single step problem, we apply it to Multiplexer problem[10] as a benchmark problem in single step problem, and compare the performance of XCSB with XCSTS[4]. Multiplexer problem is either a *learning* problem or a *test* problem. In test problem, each system randomly selects actions from each possible action, and takes place the parameter updating (reinforce component) and the genetic algorithm. In test problem, each system always selects the action which has the highest prediction array value, and turns off the parameter updating and the genetic algorithm. We employ the three evaluation criteria as follows: 1) *performance* represents the rate of correct answers is executed during the test problems; 2) *number of called covering* represents the number of called covering operators during learning problem; 3) *average of nma in action set* represents the average of the *nma* of each classifiers in $[A]$ during test problems, *i.e.*, the the average of nma of the classifiers has best action in each step. Each evaluation criterion is computed as the moving average during 5000 problems.

Multiplexer Problem: The multiplexer function is defined for binary strings of length $k + 2^k$. The output of the multiplexer function is determined by one of the 2^k value bits. The *reference bits* is determined by the k *address bits*. For example, in the 6-multiplexer function($k = 2$), $f(110001) = 1$, $f(010111) = 1$, $f(110110) = 0$. The reward of correct action is 1000 and the incorrect action is 0. Here, we employ the 20 and 37 multiplexer problems ($k = 4$ and 5).

Results on the 20 and 37 Multiplexer Problem: In the 20 multiplexer problem, the max population size N is set to different values as 1000 and 2000. XCSTS parameter setting is the same in [3] as follows: $\epsilon_0 = 10$, $\mu = 0.04$, $P_\# = 0.5$, $P_{explr} = 1.0$, $\chi = 0.8$, $\beta = 0.2$, $\alpha = 0.1$, $\delta = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, $\theta_{del} = 20$, $\theta_{sub} = 20$, $\tau = 0.4$, Tornament selection = 1, GAsubsumption = 1, ASsubsumption = 1. On the 37 multiplexer problem, N is set to 5000. The

parameter setting is the same in [3] as follows: $P_{\#} = 0.65$, others is the same in the 20 multiplexer problem. XCSB parameters is set as follows: ζ =0.99, η = 0.2, and other parameters is the same of each setting on XCSTS. Experimental results on the 20 multiplexer problem are averaged over 20 experiments, and they on the 37 multiplexer problem are averaged over 10 experiments.

Fig.2 shows the performance of XCSB and XCSTS on the 20 and 37 multiplexer problems. On the 20 multiplexer problem, XCSTS performs the high learning capabilities with $N = 2000$ but is reduced the learning speed when the small max population size $N = 1000$. While XCSB performs the higher leaning speed than XCSTS even with the small max population size. On the 37 multiplexer problem, XCSTS reaches the high performance after about 650 000 problems while XCSB reaches the high performance after about 350 000 problems. XCSB solves quickly the 37 multiplexer problems which has the large state space than 20 multiplexer problem. Fig.3 shows the number of called covering rate and the average of nma on the 37 multiplexer problem. XCSTS and XCSB do not call the covering with the progress of the learning. Additionally, the average of nma is converged to near 1, which means learns correctly the best action map to prevent calling covering. For these results, XCSB can derive the high learning capabilities even with the small max population size and with the large state space. Since XCSB correctly identify the best action and tune the calling covering to prevent the cover-delete cycle.



a) 20 multiplexer problem b) 37 multiplexer problem

Fig. 2. Performance and population size of XCSB and XCSTS with $N = 2000$ and 1000 on the 20 and 37 multiplexer problem

4.2 Multi Step Problem

To investigate the effectiveness of XCSB on the multi step problem, we apply it to Maze problem[10] as a benchmark problem in the multi step problem, and compare the performance of XCSB with XCSG[2]. Maze problem also is either a *learning* problem or a *test* problem. In the multi step problem, during the test problem, each system takes place the parameter updating but turns off the genetic algorithm. We employ the same evaluation criteria of experiment in the single step problem but *performance* represents the *step to goal* which is

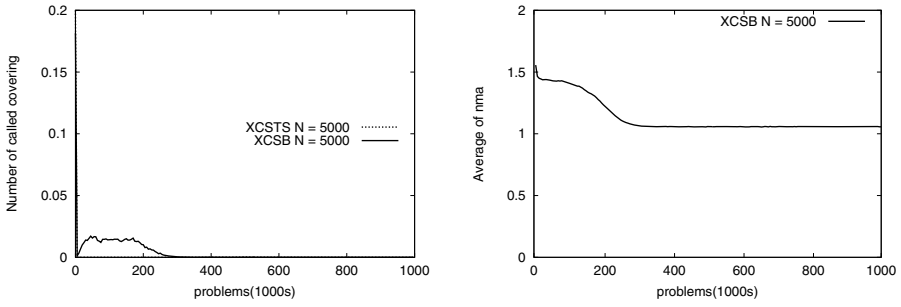


Fig. 3. Number of called covering rate of XCSB and XCSTS and average of nma of XCSB on the 37 multiplexer problem

T	T	T	T	T	T	T	T	T
T						T	F	T
T			T		T	T		T
T		T						T
T				T	T			T
T		T		T			T	T
T		T						T
T						T		T
T	T	T	T	T	T	T	T	T

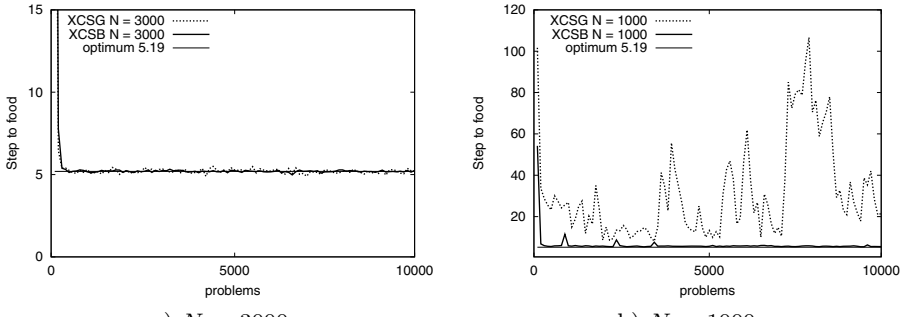
Fig. 4. Maze6

needed to reach goal position during the test problems. Each evaluation criterion is computed as the moving average during 50 problems. All experimental result are averaged over 20 experiments.

Maze Problem: This problem is composed of the empty cell as " ", the obstacle cell as "T", and the food cell as "F". The agent can recognize the eight surrounding neighbor cells, and moves cells to search the food(goal). At the beginning, the agent is randomly placed at any empty cell in the maze. When the agent reaches at the food cell, he acquires the reward 1000. To speed up the experiments, the max step number is set as 1500, which makes the agent reset and restart from the randomly selected cells when the steps exceed 1500 [2]. Here, we employ the maze6 as shown Fig. 4, the optimum step of maze 6 is 5.19.

Result on the Maze Problem: The max population size N is set to different values as 1000 and 3000. The parameters setting of XCSG is the same in [2] as follows: $\epsilon_0 = 1$, $\mu = 0.01$, $P_{\#} = 0.3$, P_{explr} =1.0, $\chi = 0.8$, $\beta = 0.2$, $\alpha = 0.1$, $\delta = 0.1$, $\gamma = 0.7$, $\nu = 5$, $\theta_{GA} = 100$, $\theta_{del} = 20$, $\theta_{sub} = 20$, Tornament selection =0, GAsubsumption = 0, ASsubsumption = 0. The parameters setting of XCSB is set as follows: ζ =0.99, $\eta = 0.2$, and other setting is the same of XCSG.

Fig.5 shows the step to goal of XCSB and XCSG. XCSG reaches to the optimum step with N =3000 while does not reach to it with N =1000. In contrast, XCSB derives the learning speed as well as XCSG with N =3000 and keep to perform the learning capabilities and speed even with the small max population size. Fig.6 shows the number of called covering of XCSB and XCSG, and the average of nma of XCSB with N = 1000 and 3000 on the maze6. XCSG prevents the called covering with the progress of learning with N = 3000 but occurs cover-delete cycles with N = 1000 since XCSG does not prevent calling covering with N = 1000. In contrast, XCSB reduces calling covering than XCSG even with N = 1000. Additionally, the number of nma is converged to near 1.5 with N =3000. Here, there are 16 cells which has two best actions, 1 cells which has three ones and 19 cells which has only one best action in all 36 empty cells of maze6. Therefore the average of number of best action in each cells is equal to 1.5 $(=(2 \times 16 + 3 + 1 \times 19)/36)$. From these results, in the multi step problem, XCSB also can derive stably the high learning capabilities even with the small max population size. Since XCSB can turn the number of called covering to identify correctly the best action even in the problem which has two or more best actions in each state.



a) N = 3000 b) N = 1000

Fig. 5. Step to goal of XCSB and XCSGon the maze6

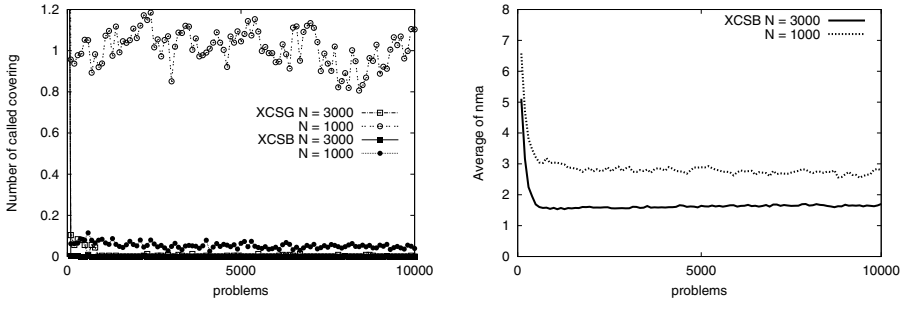


Fig. 6. Number of called covering rate of XCSB and XCSG and average of nma of XCSB on the maze6

5 Conclusion

This paper proposed a novel approach of XCS as XCS with Best Action Mapping(XCSB) to perform stably the learning capabilities even with the small max population size and even in the large state space problem. XCSB acquires only best action map and tunes the number of called covering to prevent the cover-delete cycle. To investigate the effectiveness of XCSB, we apply it to Multiplexer problem and Maze problem. Experimental results show that XCSB can derive the high performance not only with the small max population size but also in the large state space problem.

References

1. Bernadó-mansilla, E., Garrell-Guij, J.M.: Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. Evolutionary Computation 11, 209–238 (2003)
2. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Gradient Descent Methods in Learning Classifier Systems: Improving XCS Performance in Multistep Problems. Evolutionary Computation 9(5), 452–473 (2005)
3. Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.W.: Toward a Theory of Generalization and Learning in XCS. IEEE Transactions on Evolutionary Computation 8(1), 28–46 (2004)
4. Butz, M.V., Sastry, K., Goldberg, D.E.: Tournament Selection in XCS. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), pp. 1857–1869 (2003)
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley (1989)
6. Holland, J.H.: Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-based system. Machine Learning 2, 593–623 (1986)
7. Kovacs, T.: Strength or Accuracy? Fitness Calculation in Learning Classifier Systems. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 1999. LNCS (LNAI), vol. 1813, pp. 143–160. Springer, Heidelberg (2000)
8. Sutton, R.S.: Learning to Predict by the Methods of Temporal Differences. Machine Learning 3(1), 9–44 (1988)
9. Wilson, S.W.: ZCS: A Zeroth Level Classifier System. Evolutionary Computation 2(1), 1–18 (1994)
10. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)

Using Expert Knowledge to Guide Covering and Mutation in a Michigan Style Learning Classifier System to Detect Epistasis and Heterogeneity

Ryan J. Urbanowicz, Delaney Granizo-Mackenzie, and Jason H. Moore

Computational Genetics Laboratory, Department of Genetics
Dartmouth Medical School, Lebanon, NH, USA
{ryan.j.urbanowicz,jason.h.moore}@dartmouth.edu
http://www.epistasis.org/

Abstract. Learning Classifier Systems (LCSs) are a unique brand of multifaceted evolutionary algorithms well suited to complex or heterogeneous problem domains. One such domain involves data mining within genetic association studies which investigate human disease. Previously we have demonstrated the ability of Michigan-style LCSs to detect genetic associations in the presence of two complicating phenomena: epistasis and genetic heterogeneity. However, LCSs are computationally demanding and problem scaling is a common concern. The goal of this paper was to apply and evaluate expert knowledge-guided covering and mutation operators within an LCS algorithm. Expert knowledge, in the form of Spatially Uniform ReliefF (SURF) scores, was incorporated to guide learning towards regions of the problem domain most likely to be of interest. This study demonstrates that expert knowledge can improve learning efficiency in the context of a Michigan-style LCS.

Keywords: Expert Knowledge, Learning Classifier System, Genetics, Epistasis, Heterogeneity, Evolutionary Algorithm, Mutation, Covering.

1 Introduction

Learning Classifier Systems (LCSs) [1] are a rule-based class of algorithms which combine machine learning with evolutionary computing and other heuristics to produce an adaptive system. We focus on Michigan-style LCSs (M-LCSs) which uniquely make decisions using the entire rule population giving them the ability to perform on-line learning, form niches, and adapt. They have been applied to many problems including behavior modeling, function approximation, classification, and data mining [1].

Scalability and learning speed have been synonymous targets for improvement in the LCS literature [2,3,4,5,6] largely in the context of Pittsburgh-style LCSs (P-LCSs) due to their inherent limitations in this area. However these considerations are just as important for M-LCS algorithms, especially in the context of large-scale, high dimensional problems.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 266–275, 2012.

1.1 The Problem Domain: Human Genetics

One domain where these shortcomings clearly impact the utility of LCS is within human genetics. Single nucleotide polymorphisms (SNPs) are single loci in the DNA sequence where alternate nucleotides (i.e. alleles) are observed between members of a species or between paired chromosomes in an individual. In a typical genetic association study, researchers look for differences in SNP allele frequencies between a group of individuals with the disease of interest, and a matched group healthy controls.

Despite the rising quality and abundance of genetic data, epidemiologists continue to struggle with the connection of disease phenotypes to reliable genetic and environmental markers. While strategies seeking single locus associations (i.e. main effects) are often sufficient to address diseases which follow Mendelian patterns of inheritance, their application to diseases characterized as complex has yielded limited success [7,8]. Epistasis and heterogeneity have been recognized as phenomena which complicate the epidemiological mapping of genotype to phenotype [9]. In the present context, epistasis simply refers to attribute interaction. *Heterogeneity*, referring to either *genetic* heterogeneity (locus and allelic) or *environmental* heterogeneity, occurs when individual (or sets of) attributes are independently predictive of the same phenotype (i.e. class).

As proof of principle, M-LCSs were applied to the detection and modeling of simulated epistatic and heterogeneous genetic disease associations [10]. These evaluations identified the strengths and weaknesses of M-LCS on these types of complex, noisy problems. To address the shortcoming of knowledge discovery in M-LCSs we previously introduced an analysis pipeline with statistical and visualization-guided strategies for rule population interpretation [11]. In order to explicitly identify heterogeneity we introduced a strategy to link instances in the dataset to respective heterogeneous subgroups using attribute tracking and feedback [12]. To date, we have a functional LCS algorithm able to concurrently detect patterns of association with epistasis and heterogeneity. In this work, we turn our focus to improving the efficiency and scalability of this strategy.

The present study explores the adaptation of expert knowledge, previously utilized in the context of other evolutionary algorithms, to improve algorithm efficiency [13,14,15,16]. Here, we uniquely introduce expert knowledge to an M-LCS algorithm and develop the strategies to utilize it. To achieve this goal we: (1) derive expert knowledge from Spatially Uniform ReliefF (SURF) [17], (2) adopt a logistic function to reliably transform any set of expert knowledge scores into a set of attribute respective probabilities, (3) utilize these probabilities to intelligently guide covering and mutation operators within M-LCS towards regions of the problem domain most likely to be of interest.

2 Methods

In this section we describe (1) the M-LCS algorithm and run parameters used in this investigation, (2) SURF, the selected source of our EK, (3) logistic transformation of the EK values into probabilities, (4) the incorporation of EK into

the covering and mutation mechanisms, and (5) our experimental evaluation of the proposed mechanisms.

2.1 Learning Classifier System: UCS

M-LCSs, often varying widely from version to version, generally possess four basic components; (1) a population of rules or classifiers, (2) a performance component that assesses how well the population of rules collectively explain the data, (3) a reinforcement component that distributes the rewards for correct prediction to each of the rules in the population, and (4) a discovery component that uses different operators to discover new rules and improve existing ones. Learning progresses iteratively, relying on the performance and reinforcement components to drive the discovery of better rules. For a complete LCS introduction and review, see [1].

The sUpervised Classifier System (UCS) [18], is a M-LCS based largely on the very successful XCS algorithm [19], replacing reinforcement learning with supervised learning. UCS was designed specifically to address single-step problem domains such as classification and data mining, displaying particular promise when applied to attribute interaction and heterogeneity in [10].

For evaluation purposes we implement expert knowledge into a Python encoding of the UCS algorithm [10]. We utilize mostly default run parameters with the exception of 200,000 learning iterations, a population size of 1600, tournament selection, uniform crossover, subsumption, attribute mutation probability = 0.04, crossover probability = 0.8, and $\nu = 1$. ν has been described as a "constant set by the user that determines the strength [of] pressure toward accurate classifiers" [20], and is typically set to 10 by default. A low ν was used to place less emphasis on high accuracy in this type of noisy problem domain, where 100% accuracy is only indicative of over-fitting. While we run each algorithm for a maximum of 200,000 iterations, we also stop and evaluate the systems after 10,000, 50,000, and 100,000 iterations. Also, as in [10], we employ a quaternary rule representation, where for each SNP attribute, a rule specifies genotype as (0, 1, or 2), or instead generalizes with "#", a character which implies that the rule doesn't care about the state of that particular attribute. The implementation described above is available on request (*ryanurbanowicz@gmail.com*) and will be posted on the LCS and GBML Central webpage.

2.2 Expert Knowledge from Spatially Uniform ReliefF (SURF)

Expert knowledge (EK) is an external source of information providing, in this context, a measure of attribute quality. In this study, the external measure was statistical, but it could just as easily be biological. The usefulness of EK is entirely dependent on it's quality. There are many statistical and computational methods for determining the quality of attributes. We selected a method that is capable of identifying attributes that predict class primarily through dependencies or interactions with other attributes. To this end we selected Spatially

Uniform ReliefF (SURF) [17] as the source of EK. SURF estimates the quality of attributes through a nearest neighbor algorithm that selects neighbors (case or control instances) within an automatically determined distance threshold. Weights, or quality estimates, for each attribute are estimated based on whether the nearest neighbor (nearest hit) of a randomly selected instance from the same class and the nearest neighbor from the other class (nearest miss) have the same or different values. In addition to SURF, [17] also evaluated Tuned ReliefF (TuRF) [21] which was designed for human genetics application. TuRF systematically removes attributes that have low quality estimates so that the ReliefF values of the remaining attributes can be re-estimated. SURF and TuRF could be combined in future work to derive EK scores, but in the present study we exclusively utilize SURF to generate EK scores for every simulated training dataset described in section 2.5. Note that the SURF run time was negligible compared to that of UCS: requiring a matter of seconds to run.

2.3 Transformation of Expert Knowledge into Probabilities

This section describes our application of the logistic function in order to transform raw EK scores into normalized probability values. We start with a set of raw EK values $K \subset \mathbb{R}$. We know neither the range nor distribution of the EK values. The only requirement is that greater importance should translate to a larger EK score. Let $n = |K|$ and $k_i \in K | 1 \leq i \leq n$ be the ith score.

We use the logistic function to transform the raw values into selection probabilities. Namely,

$$\ell_{\alpha,\beta}(x) = \frac{1}{1 + e^{-(\alpha+\beta x)}}$$

.

Before applying this function we must determine values for constants α and β. First, the user specifies a range constant d. We have chosen $d = 0.4$ which yields an output range of $(0.5 - d = d_l = 0.1, 0.5 + d = d_u = 0.9)$. This range ensures that the attribute with the lowest EK score retains at least a 10% chance to be specified, while the attribute with the highest EK score has no greater than a 90% chance to be specified. Next, the user must specify c, which is the resulting sum of probabilities in the transformed set. This is useful when one wants to guarantee that a selection algorithm, given the set of probabilities, returns a certain number of individuals on average. In this study we want selection to choose 50% of the attributes during covering, therefore we set $c = 10$ since datasets each have 20 attributes. Lastly, the user must specify how many digits of precision (set to 5 here) we use when calculating α in step 5. EK transformation progresses in five steps as follows:

Step 1: Compute range $r = max(K) - min(K)$.

Step 2: Shift scores such that the lowest score is at minimum zero. If $min(K) <$ 0 then add $abs(min(K))$ to every EK score.

Step 3: Compute β such that the logistic function's slope 'nicely' occupies the data range, i.e. $\ell_{\alpha,\beta}(-r/2) = d_l$ and $\ell_{\alpha,\beta}(r/2) = d_u$. Since α does not affect

slope we set $\alpha = 0$ while calculating β. We solve for β after some simple algebra with the following:

$$\beta = 2\ln(\frac{1-d_l}{d_l})/r$$

Step 4: Compute an initial guess for α such that $\ell_{\alpha,\beta}(min(K)) = d_l$. Since α simply shifts the function left or right we only need to move the curve such that the minimum EK score is transformed to d_l. We solve for this initial α_0 using the following equation derived again with some simple algebra:

$$\alpha_0 = -\beta(min(K) + r/2)$$

Step 5: In order to find α such that the transformed probabilities sum to c we iteratively search for an appropriate value of α using the Newton-Raphson method. For our purposes β is a constant. We applied differential calculus to obtain:

$$\alpha_j = \alpha_{j-1} - \frac{\sum_{i=0}^{n} \ell_{\alpha,\beta}(k_i) - c}{\sum_{i=0}^{n} \ell'_{\alpha,\beta}(k_i)}$$

We iterate this equation until $\alpha_j = \alpha_{j-1}$ with respect to the digits of precision. At this point applying the logistic function to the input scores (K) using the computed parameters, α and β, will produce a set of probabilities that sum to the desired c.

2.4 Expert Knowledge Applied to Covering and Mutation

Once EK-based probabilities has been generated for all attributes, we incorporated these as weights to guide LCS learning. To achieve this, we apply these probabilities to both covering and mutation operators within M-LCS. The covering operator is responsible for population initialization, as well as ensuring that a matching rule exists for a given data instance within each learning iteration. Previously, EK has been successfully applied to population initialization in genetic programming [16]. In the context of LCS, EK probabilities drive the specialization (state is important) or generalization (state is not important) of attributes within rules. Having chosen a c of 10, rule generated via covering will tend to have half of the 20 attributes specified, and half generalized. The standard covering mechanism gives each attribute a 50% chance of being specialized. With the incorporation of EK, attributes with higher EK scores (likely to be useful) will have a higher probability of being specified, and vice-versa.

The mutation mechanism is a discovery component of the M-LCS. When activated, mutation traditionally randomly permutes an element of the rule such that if it had been specified it becomes generalized and vise-versa. Previously, EK has been successfully applied to mutation in genetic programming [15]. Here, we apply EK probabilities to mutation, such that if an attribute is selected for

'possible' mutation, the probability of mutation is equal to the EK probability for that attribute. Specified attributes with high EK-scores will be less likely to be generalized while generalized attribute with high EK-scores will more likely to be flipped to specified. The opposite is true for attribute with low EK-scores.

In this study we evaluate the utilization of EK into UCS over four trials. The trials include the following scenarios: (1) UCS algorithm without EK (UCS), (2) UCS with EK applied to covering only (UCS-EK-Cov), (3) UCS with EK applied to mutation only (UCS-EK-Mut), and (4) UCS with EK applied to both covering and mutation (UCS-EK-Both).

2.5 Data Simulation and Analysis

We evaluate EK using simulated datasets which concurrently model heterogeneity and epistasis as they might appear in a SNP gene association study of common complex disease [10,22]. All data sets were generated using a pair of distinct, two-locus epistatic interaction models, both utilized to generate instances (i.e. case and control individuals) within a respective subset of each final data set. Each two-locus epistatic model was simulated without Mendelian/main effects, as a penetrance table as in [10]. Due to the computational demands of LCSs, this study limited its evaluation to 3 heterogeneity/epistasis model combinations. For simplicity the minor allele frequency of each predictive attribute was set to 0.2, a reasonable assumption for a common complex disease SNP. The three model combinations included a pair of models with a heritability of either (0.1, 0.2, or 0.4). We considered model architectural "difficulties" of both "easy" and "hard" [23]. Balanced datasets simulated from these models were generated as having four different sample sizes (200, 400, 800, or 1600) and a heterogeneous mix ratio of either (50:50 or 75:25) (e.g. 75% of instances were generated from one epistatic model, and 25% were generated from a different one). Twenty replicates of each dataset were analyzed and 10-fold cross validation (CV) was employed to measure average testing accuracy and account for over-fitting. Together, a total of 48 data set configurations (*3 Model Combos* x *4 Sample Sizes* x *2 Ratios* x *2 Difficulties*), and a total of 960 data sets (20 random seeds each) were simulated. With 10-fold CV, 9600 runs of each of the four UCS trials were completed.

For each run we track the following statistics; training accuracy, testing accuracy, generality, macro population size, the power to find both underlying models, the power to find at least one underlying model, the power to correctly rank attribute co-occurrence [11], and run time. Power is a reflection of our ability to reliably mine knowledge from the evolved rule population. Co-occurrence power is a reflection of our ability to distinguish heterogeneous models. Each of these values represent an average over the 10 CV runs.

Statistical comparisons were made using the Wilcoxon signed-rank tests due to a lack of normality in the value distributions. All statistical evaluations were completed using R. Comparisons were considered to be significant at $p \leq 0.05$.

3 Experimental Results and Discussion

This analysis of EK spans over a spectrum of complex datasets, with evaluations taken at four different iteration intervals, and examines a variety of M-LCS statistics in order to compare the performance of UCS, UCS-EK-Cov, UCS-EK-Mut, and UCS-EK-Both. Table 1 summarizes averages and identifies significant differences for all run statistics over all simulated datasets between our four experimental investigations after either 10,000 or 200,000 iterations. Averages after 10,000 iterations reveal the impact of EK very early on in the learning process, while averages after 200,000 give a better sense of it's impact after the learning curve has started to level off.

Table 1. Comparing UCS with EK implementations after either 10,000 or 200,000 learning iterations each averaged over all simulated datasets. Arrows indicate a significant increase or decrease when compared to UCS.

10,000 Iterations								
Statistics	UCS	UCS-EK-Cov	p	UCS-EK-Mut	p	UCS-EK-Both	p
Train Accuracy	0.8268	0.8439	↑ **	0.7918	↓ **	0.8301	↑ **
Test Accuracy	0.5909	0.6112	↑ **	0.6283	↑ **	0.6278	↑ **
Both Power	0.15	0.2677	↑ **	0.2458	↑ **	0.2969	↑ **
Single Power	0.5427	0.7281	↑ **	0.6885	↑ **	0.7302	↑ **
Co-Occur. Power	0.1531	0.1625	-	0.1469	-	0.0396	↓ **
Generality	0.7019	0.6543	↓ **	0.7351	↑ **	0.6580	↓ **
Macro Population	1371.82	1400.71	↑ **	1244.44	↓ **	1276.68	↓ **
Run Time (min)	1.61	1.79	↑ **	1.20	↓ **	1.43	↓ **
200,000 Iterations							
Statistics	UCS	UCS-EK-Cov	p	UCS-EK-Mut	p	UCS-EK-Both	p
Train Accuracy	0.8544	0.8546	-	0.8468	↓ **	0.8467	↓ **
Test Accuracy	0.6134	0.6141	-	0.6283	↑ **	0.6278	↑ **
Both Power	0.3115	0.3083	-	0.3927	↑ **	0.3990	↑ **
Single Power	0.7125	0.7156	-	0.7677	↑ **	0.7625	↑ **
Co-Occur. Power	0.2438	0.25	-	0.2302	-	0.2260	↓ *
Generality	0.7136	0.7138	↑ *	0.7535	↑ **	0.7533	↑ **
Macro Population	1317.09	1316.84	-	1173.03	↓ **	1172.21	↓ **
Run Time (min)	37.48	35.85	↓ **	29.66	↓ **	29.06	↓ **

– Not Sig.
* $p < 0.05$
** $p << 0.001$

Most notable after only 10,000 iterations, all three EK implementations show significant improvement in testing accuracy, and the power to find one or both underlying models supporting the hypothesis that EK can direct LCS towards important regions of the problem space to improve learning efficiency. After 200,000 iterations the advantages of using UCS-EK-Cov become hidden, as most observed statistics are comparable to those seen using UCS. This indicates that

UCS-EK-Cov speeds up learning, but given enough time, UCS is able to achieve similar performance. However, for UCS-EK-Mut, and UCS-EK-Both, we again observe significant improvements in testing accuracy, both, and single power even after 200,000 iterations. Additionally for these two implementations we observe increased rule generality, a smaller macro population size, and a decrease run time, all considered to be indicators of improved learning efficiency in this context. Figure 1A illustrates changing rule generality at different learning intervals for each implementation, while Figure 1B illustrates the same for macro population size. Increasing generality while maintaining or improving accuracy indicates that UCS is doing a better job focusing on attributes that will be valuable for making predictions on subjects it has not yet seen. Decreasing macro population size suggests that a smaller, more reliable, and applicable set of rules have been found by UCS. The only statistic which was not improved via EK incorporation was co-occurrence power. This is a logical finding given that EK operates globally during the learning process. Since co-occurrence power reflects the ability of the system to separate heterogeneous models, it makes sense that a global EK mutation pressure (applied uniformly to all rules) may reduce the systems overall ability to differentiate heterogeneity.

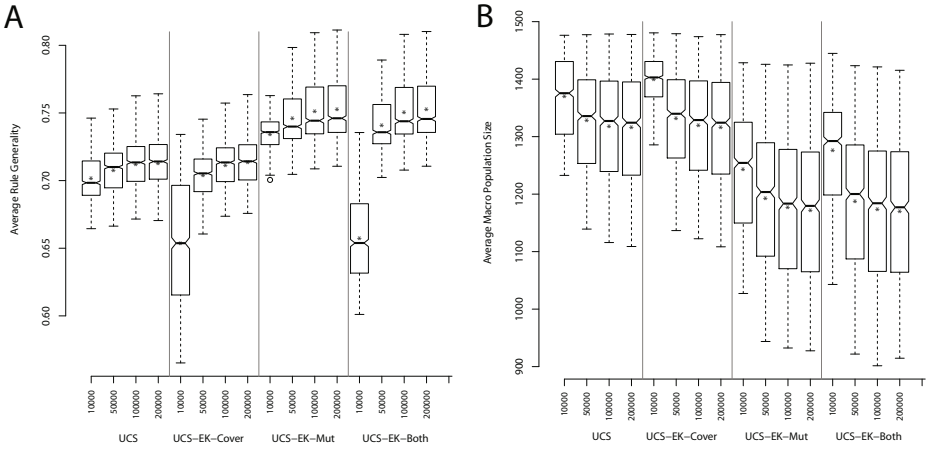


Fig. 1. (A) Comparing average rule population generality and (B) comparing average macro population size between UCS and UCS implementations utilizing EK. In both plots, the values on the x-axis indicate the number of completed learning iterations. Each box includes 960 observations. The star within each box plot indicates the average of those values.

4 Conclusions

The primary conclusion of this work is that EK may be successfully applied to an M-LCS algorithm to improve learning efficiency. We observe better algorithm performance when using EK after as few as 10,000 iterations. We have developed

and evaluated a strategy for implementing EK using SURF scores as a source of EK, transforming any EK score source into usable probabilities, and incorporating these probabilities into covering and mutation mechanisms for the M-LCS. Overall, our findings support the inclusion of EK in M-LCS as a strategy for improving learning efficiency. However, in the context of potentially heterogeneous problems it may be better to (1) limit the use of EK to covering alone and (2) guide GA mechanisms such as mutation and crossover with local information as explored in [12] rather than with global information (i.e. EK). In future work we will extend this effort to consider the integration of EK and attribute feedback [12]. We will also extend these analysis to datasets with larger numbers of attributes in order to more directly evaluate improvements in scalability. This work supports the overall conclusion of related studies examining EK in the context of evolutionary algorithms [14,15,16]: that EK can be effective at pointing the algorithm towards attributes of greatest interest therefore facilitating the algorithm's ability to find "the genetic needle in the the genomic haystack".

Acknowledgments. This work was supported by the William H. Neukom 1964 Institute for Computational Science and by Award Number R25CA134286 from the National Cancer Institute. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Cancer Institute or the National Institutes of Health.

References

1. Urbanowicz, R., Moore, J.: LCSs: A Complete Introduction, Review, and Roadmap. Journal of Artificial Evolution and Applications 2009 (2009)
2. Bacardit, J., Goldberg, D.E., Butz, M.V., Llorà, X., Garrell, J.M.: Speeding-Up Pittsburgh Learning Classifier Systems: Modeling Time and Accuracy. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN VIII. LNCS, vol. 3242, pp. 1021–1031. Springer, Heidelberg (2004)
3. Bacardit, J., Stout, M., Hirst, J., Sastry, K., Llorà, X., Krasnogor, N.: Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 346–353. ACM (2007)
4. Llorà, X., Sastry, K.: Fast rule matching for lcss via vector instructions. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 1513–1520. ACM (2006)
5. Bacardit, J., Burke, E., Krasnogor, N.: Improving the scalability of rule-based evolutionary learning. Memetic Computing 1(1), 55–67 (2009)
6. Franco, M., Krasnogor, N., Bacardit, J.: Speeding up the evaluation of evolutionary learning systems using gpgpus. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 1039–1046. ACM (2010)
7. Shriner, D., Vaughan, L., Padilla, M., et al.: Problems with genome-wide association studies. Science 316(5833) (2007) 1840c
8. Eichler, E., Flint, J., Gibson, G., Kong, A., Leal, S., Moore, J., Nadeau, J.: Missing heritability and strategies for finding the underlying causes of complex disease. Nature Reviews Genetics 11(6), 446–450 (2010)

9. Thornton-Wells, T., Moore, J., Haines, J.: Genetics, statistics and human disease: analytical retooling for complexity. TRENDS in Genetics 20(12), 640–647 (2004)
10. Urbanowicz, R., Moore, J.: The application of michigan-style lcss to address genetic heterogeneity and epistasis in association studies. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 195–202. ACM (2010)
11. Urbanowicz, R., Granizo-Mackenzie, A., Moore, J.: An Analysis Pipeline with Visualization-Guided Knowledge Discovery for Michigan-Style LCSs. IEEE CIM Special Issue on Computational Intelligence in Bioinformatics (2012)
12. Urbanowicz, R., Granizo-Mackenzie, A., Moore, J.: Instance-Linked Attribute Tracking and Feedback for Michigan-Style Supervised LCSs. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (2012)
13. Jamshidi, M., et al.: Incorporating a-priori expert knowledge in genetic algorithms. In: Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA 1997, pp. 300–305. IEEE (1997)
14. Moore, J.H., White, B.C.: Exploiting Expert Knowledge in Genetic Programming for Genome-Wide Genetic Analysis. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN IX. LNCS, vol. 4193, pp. 969–977. Springer, Heidelberg (2006)
15. Greene, C.S., White, B.C., Moore, J.H.: An Expert Knowledge-Guided Mutation Operator for Genome-Wide Genetic Analysis Using Genetic Programming. In: Rajapakse, J.C., Schmidt, B., Volkert, L.G. (eds.) PRIB 2007. LNCS (LNBI), vol. 4774, pp. 30–40. Springer, Heidelberg (2007)
16. Greene, C.S., White, B.C., Moore, J.H.: Sensible initialization using expert knowledge for genome-wide analysis of epistasis using genetic programming. In: IEEE Congress on Evolutionary Computation, CEC 2009, pp. 1289–1296. IEEE (2009)
17. Greene, C.S., Penrod, N., Kiralis, J., Moore, J.: Spatially uniform relieff (surf) for computationally-efficient filtering of gene-gene interactions. BioData Mining 2(1), 1–9 (2009)
18. Bernadó-Mansilla, E., Garrell-Guiu, J.: Accuracy-based LCSs: models, analysis and applications to classification tasks. Evolutionary Computation 11(3), 209–238 (2003)
19. Wilson, S.: Classifier fitness based on accuracy. Evolutionary Computation 3(2), 149–175 (1995)
20. Orriols-Puig, A., Bernadó-Mansilla, E.: Revisiting ucs: Description, fitness sharing, and comparison with xcs. Learning Classifier Systems, 96–116 (2008)
21. Moore, J.H., White, B.C.: Tuning ReliefF for Genome-Wide Genetic Analysis. In: Marchiori, E., Moore, J.H., Rajapakse, J.C. (eds.) EvoBIO 2007. LNCS, vol. 4447, pp. 166–175. Springer, Heidelberg (2007)
22. Urbanowicz, R.J., Moore, J.H.: The Application of Pittsburgh-Style Learning Classifier Systems to Address Genetic Heterogeneity and Epistasis in Association Studies. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 404–413. Springer, Heidelberg (2010)
23. Urbanowicz, R., Kiralis, J., Fisher, J., Moore, J.: Predicting Difficulty in Simulated Genetic Models: Metrics for Model Architecture Selection. BMC Bioinformatics (submitted)

On Measures to Build Linkage Trees in LTGA

Peter A.N. Bosman[1] and Dirk Thierens[2]

[1] Centre for Mathematics and Computer Science, P.O. Box 94079,
1090 GB Amsterdam, The Netherlands
Peter.Bosman@cwi.nl
[2] Department of Information and Computing Sciences, Utrecht University,
Utrecht, The Netherlands
D.Thierens@uu.nl

Abstract. For an evolutionary algorithm (EA) to be efficiently scalable, variation must be linkage friendly. For this reason many EAs have been introduced that build and exploit linkage models, amongst which are estimation-of-distribution algorithms (EDAs). Although various models have been empirically evaluated, it remains of key importance to better understand the conditions under which model building is successful. In this paper, we consider the linkage tree genetic algorithm (LTGA). LTGA is a recent powerful linkage-learning EA that builds a hierarchical linkage model known as the linkage tree (LT). LTGA exploits this model using an intensive mixing procedure aimed at optimally exchanging building blocks. Empirical evaluation studies of LTGA have appeared in literature using different entropy-based measures for building the LT, but with comparable results. We study the differences in these measures to better understand the requirements for detecting important linkage information and point out why some measures are more successful than others.

1 Introduction

Having a tunable model that drives variation in an evolutionary algorithm (EA) potentially allows efficiently tackling a large class of optimization problems. Key to successfully solving a particular problem is the ability to configure this model for that problem, i.e. such that combining solutions leads to (significant) improvements. Key questions are whether such a proper configuration can be learned efficiently and what type of model and learning algorithm are required. In this paper we consider this question in the light of one of the latest and most promising model-building EAs for discrete optimization problems: the linkage tree genetic algorithm (LTGA) [1,5,10].

The type of EA that is perhaps best known for building and using models is the estimation-of-distribution algorithm (EDA). Models in EDAs represent probability distributions over the space of solutions. In EDAs, linkage information, i.e. which variables should be considered jointly when generating new solutions, is processed via probabilistic dependency relations. Although probability theory provides very powerful tools, estimating complete distributions might be more than what is required. Instead therefore, LTGA learns linkage relations directly,

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 276–285, 2012.

although the statistical techniques used to do so have much in common with building probabilistic models as is done in EDAs.

LTGA exhibits excellent performance on several benchmark problems [1,5,10]. The measures used to learn the linkage model in these studies are however different, although all are entropy-based. In this paper we take a closer look at why the results using different measures are so similar in order to better understand the requirements for detecting important linkage information. To do so, we consider what it is we require of a linkage measure from a viewpoint of EA dynamics rather than from probability theory and notions of probabilistic independence as in EDAs.

2 The Linkage Tree Genetic Algorithm (LTGA)

Here we only briefly describe the most recent version of LTGA [1]. For more details we refer the interested reader to the related literature [1,5,10].

To model linkage, a linkage tree (LT) is used in which variables can be linked on one level but not linked on another, lower, level. At the lowest level, all variables are unlinked and form singleton sets. An LT then can be formed by merging pairs of sets until all sets are merged. An example of an LT is given in Figure 1.

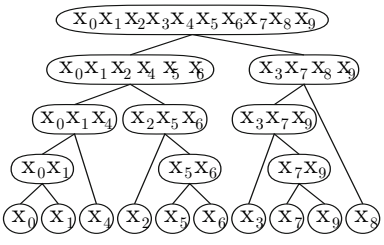


Fig. 1. Example of a LT for 10 variables

In each generation of LTGA a set of n solutions is selected from a population of size n using tournament selection with a tournament size of 2. The LT is learned from this set. New solutions are however generated from the population directly. Instead of fully creating new solutions first and only then evaluating fitness, LTGA uses a procedure called Genepool Optimal Mixing (GOM) [10]. For each solution in the population, exactly one offspring solution is generated. To do so, the solution is first cloned. The LT is then traversed in reverse-merging order, i.e. ending with all variables in separate groups, and skipping the top level. For each group, a donor solution is chosen randomly from the population. The values in the donor pertaining to the variables in the linkage group are then copied. If the solution is thereby improved, it is kept, otherwise the changes are reverted. The use of GOM increases selection pressure in a building-block-wise manner instead of on an entire solution basis. It is mostly because of OM that LTGA requires only very small population sizes compared to most EDAs.

To ensure efficient convergence to a single solution, in the latest version of LTGA [1] GOM is extended with forced improvements (FI). In FI, if a solution could not be improved by GOM, an additional GOM operation is performed on that solution but now with the currently known best solution as the donor, this time stopping as soon as an improvement is detected. FI introduces a special directed convergence pressure, through linkage space, toward the best solution found so far. Note that FI doesn't continuously reduce population diversity, but only if a solution couldn't be improved anymore anyway.

3 Measures to Build Linkage Trees

To build a LT, a similarity measure to be maximized or distance measure to be minimized is required in order to decide which two groups of variables to merge next. Building a LT is also known as hierarchical clustering [4].

3.1 Commonly Used Measures for Hierarchical Clustering

The most commonly adopted measures are based on mutual information (MI) and variation of information (VI). Both measures themselves are based on entropy. For a set X of random variables, the entropy $H(X)$ is given by:

$$H(X) = \sum_{x \in \Omega_X} -P(X = x)\log(P(X = x)) \quad (1)$$

where Ω_X is the sample space of X, i.e. all 2^k bit combinations for k binary variables. For two sets of random variables X and Y, MI and VI are given by:

$$MI(X,Y) = H(X) + H(Y) - H(X \cup Y) \quad (2)$$

$$VI(X,Y) = H(X \cup Y) - MI(X,Y) = 2H(X \cup Y) - H(X) - H(Y) \quad (3)$$

MI is a similarity measure whereas VI is a distance measure. MI is closely related to probabilistic model building in EDAs. Specifically, MI times the size of the data set is identical to the negative log-likelihood difference of two distributions that differ only in modelling X and Y independently or jointly. This difference is part of extended-likelihood measures, which are commonly used, e.g. in the well-known EDA ECGA [3].

The actual measures commonly encountered in hierarchical clustering literature are normalized versions of MI and VI. The reason for this is that the range of both measures is dependent on the number of variables in X and Y. This results in a bias to favoring large sets when deciding which two sets to merge [4]. Different normalizations are possible. We denote the normalized version used in LTGA by NVI. We similarly normalize MI and denote that by MNI, giving:

$$MNI(X,Y) = MI(X,Y)/H(X \cup Y) = (H(X) + H(Y))/H(X \cup Y) - 1 \quad (4)$$

$$NVI(X,Y) = VI(X,Y)/H(X \cup Y) = 2 - (H(X) + H(Y))/H(X \cup Y) \quad (5)$$

Although the direct use of NVI results in well-balanced LTs in LTGA and excellent optimization performance of LTGA [9], joint entropies need to be computed for every candidate set. High up in the LT these sets contain many variables. This poses a computational burden because Equation 1 requires summing over all variable configurations (encountered in the population). For this reason, a measure adaptation known as UPGMA (unweighted pair group method with arithmetic mean) was used in recent versions of LTGA [1,5,10]. With UPGMA all possible pairs of variables are considered. This is computationally beneficial if the computational effort to compute a measure grows faster than quadratic

in the number of variables, which is the case for VI (and MI). The UPGMA adaptation of a measure M is given by:

$$M^{UPGMA}(X,Y) = \frac{1}{|X||Y|} \sum_{X_i \in X} \sum_{Y_j \in Y} M(\{X_i\},\{Y_j\}) \tag{6}$$

LTGA with an UPGMA adaptation of NVI performs at least as good as LTGA with NVI, in terms of the required number of function evaluations [5]. However, with UPGMA, every actual VI computation is performed for just 2 random variables. Arguably therefore normalization is no longer required. For this reason LTGA was also recently tested with UPGMA and MI, obtaining apparently comparable results [10]. This raises the natural question of how these measures really guide hierarchical clustering and thereby LTGA.

3.2 Measures from a Viewpoint of EA Dynamics

What we ideally desire from a EA is efficient mixing of building blocks, i.e. instances of sets of variables that have an above-average contribution to a solutions' quality. Selection gives these building blocks more copies, thereby reducing the diversity of instances for the involved variables. The latter is exactly what is measured by H (Equation 1). This therefore suggests that we could minimize $H(X,Y) = H(X \cup Y)$ when deciding which sets of variables X and Y to merge.

However, reducing the dispersion of instances by itself isn't sufficient because this also happens simply because the EA converges. Thus, from a viewpoint of EA dynamics H has an undesirable bias toward more converged variables. This can also be seen in Figure 2. Combinations with the converged variable X_1 lead to a lower H than when variables X_0 and X_2 are combined, merely because the entropy of X_1 itself is 0. Instead therefore, what is really of interest is the *change* in dispersion of instances when going from possible combinations of available instances of X and Y to actually available instances of $X \cup Y$. Such an effect is exactly what MI (Equation 2) measures. Indeed, in Figure 2 we see that when considering what variable to best join X_0 with, X_2 is best when using MI whereas using H directly the converged and uninteresting variable X_1 appears best.

Data	X_0	X_1	X_2
0	1	1	0
1	1	1	1
2	0	1	0
3	1	1	1
4	1	1	0
5	1	1	1
6	1	1	1
7	0	1	0
8	0	1	0
9	1	1	1

H	X_0	X_1	X_2
	0.88	0.00	1.00

MI	X_0	X_1	X_2
X_0	0.88	0.00	0.40
X_1	0.00	0.00	0.00
X_2	0.40	0.00	1.00

NMI	X_0	X_1	X_2
X_0	1.00	0.00	0.27
X_1	0.00	1.00	0.00
X_2	0.27	0.00	1.00

H	X_0	X_1	X_2
X_0	0.88	0.88	1.48
X_1	0.88	0.00	1.00
X_2	1.48	1.00	1.00

VI	X_0	X_1	X_2
X_0	0.00	0.88	1.09
X_1	0.88	0.00	1.00
X_2	1.09	1.00	0.00

NVI	X_0	X_1	X_2
X_0	0.00	1.00	0.73
X_1	1.00	0.00	1.00
X_2	0.73	1.00	0.00

Fig. 2. Example of all measures for a specific set of data and 3 random variables

Since VI is a negation of MI, it appears equally useful. However, the negation involves $H(X,Y)$, resulting in $H(X,Y)$ weighing twice as heavy in VI as it does in MI (Equations 2 and 3). As a result, the bias in H toward more converged variables rings through more in VI. Considering EA dynamics, VI is therefore further away from the desirable properties of a linkage detection measure than MI. Accordingly, using VI in Figure 2 results in a different preference relation than using MI. Now X_1 is best to combine with X_0, just like when using H.

As mentioned earlier, normalization removes the bias of VI to large clusters. The above however suggests VI additionally has a bias toward more converged variables. However, recent literature suggests that, in combination with UPGMA, LTGA using NVI [5] performs equally good as LTGA using MI [10]. Normalization thus appears to change *more* things. This can indeed already be seen in the example in Figure 2 because, similar to MI, when using NVI we find that the best variable to combine with X_0 is X_2. When we look closer, we can indeed see that normalization brings VI very close to MI. Clearly, from Equations 4 and 5 we have $NVI(X,Y) = 1 - MNI(X,Y)$. This also holds using UPGMA, since $NVI^{UPGMA}(X,Y) = (|X||Y|)^{-1} \sum_{X_i \in X} \sum_{Y_j \in Y} 1 - MNI(\{X_i\},\{Y_j\}) = (|X||Y|)^{-1}(|X||Y| - \sum_{X_i \in X} \sum_{Y_j \in Y} MNI(\{X_i\},\{Y_j\})) = 1 - MNI^{UPGMA}(X,Y)$. Using MNI or NVI to build an LT is therefore an identical approach. Contrary to the use of H and VI, the use of NVI is however quite similar to the use of MI, as can be seen by equivalently comparing MNI and MI. Normalization of MI shifts the importance of the *absolute* difference between joint entropy and the individual entropies to their *relative* difference. For EA dynamics this means that normalization brings the advantage that if variables are nearly converged, the reduction in instance dispersion that we want to detect can still lead to large "signals" for the learning algorithm to pick up. Compared to MNI, MI therefore has a larger preference for less converged variables. Note that this relative bias only plays a role in cases where a desirable reduction in instance dispersion is already present. Thus, this bias in MI toward less converged variables is likely not harmful like the bias in VI or in H toward more converged variables. Furthermore, it is beforehand unclear whether MI or MNI is to be preferred in LTGA. Preferring entropy reductions in less converged variables (i.e. MI) allows to reduce noise in the overall optimization process faster. The opposite however allows first considering variables whose diversity is running out the fastest.

For all combinations of measures we empirically determined how often they disagree. For a fine-grained sampling of all possible probabilities for two sets of two binary variables $\{X_0, X_1\}$ and $\{X_2, X_3\}$ we determined, for two different measures M_0 and M_1, whether $M_i(\{X_0\},\{X_1\}) \succ M_i(\{X_2\},\{X_3\})$, $i \in \{0,1\}$. We also created correlation graphs for measure differences $M_i(\{X_2\},\{X_3\}) - M_i(\{X_0\},\{X_1\})$. The results are shown in Figure 3. Because MI and MNI need to be maximized and H, VI and NVI need to be minimized, we replaced MI and MNI by -MI and -MNI. In the correlation graphs, two measures agree if the observed differences are both positive or both negative (i.e. all-positive and all-negative quadrants). Moreover, observations for H and VI were divided by 2 to ensure that all measures have a difference in the range $[-1;1]$.

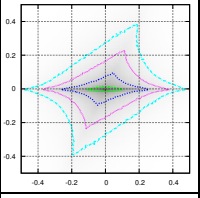
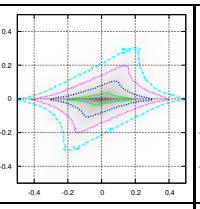
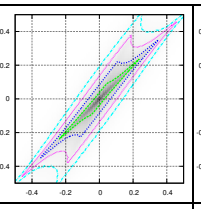
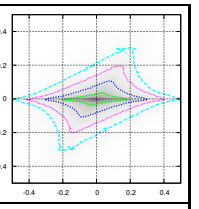
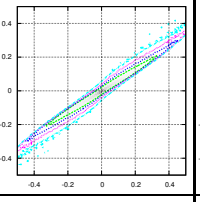
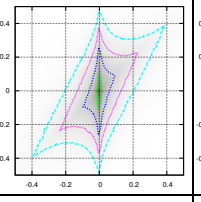
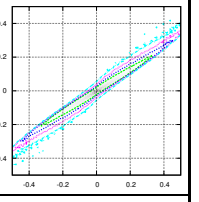
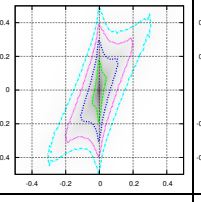
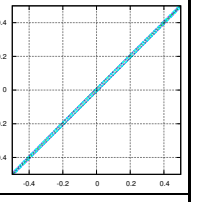
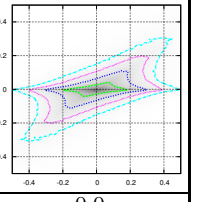
	H	MI	MNI	VI	NVI
H	0.0				
MI	40.4	0.0			
MNI	37.0	3.4	0.0		
VI	10.1	30.3	26.9	0.0	
NVI	37.0	3.4	0.0	26.9	0.0

Fig. 3. Percentage of cases in which measures disagree about the preference ordering of two sets of random variables (lower-left triangle) and gray-value coded maps of the density of observed combinations of measure differences for two sets of random variables, overlaid with density contours

Figure 3 shows a strong agreement of VI with H. Given the strong bias of H toward more converged variables, VI is, like H, expected to lead to less efficient behavior of LTGA. MI is decorrelated the most with H, followed by MNI. This is in accordance with our earlier finding that MI is relatively more biased to less converged variables. NVI and MNI are, as expected, in perfect agreement. The difference between MI and MNI is small. When they do differ, the measures themselves exhibit only small differences (i.e. decisions are a "close" call).

Ultimately, it is the optimization performance of the EA that is of importance. From the analysis above it is to be expected that the use of MI and MNI (and, equivalently, NVI) in LTGA leads to better performance on non-randomly structured problems than the use of H and VI. However, the difference between MI and MNI, i.e. normalization, is subtle. Therefore, in the next Section we empirically determine whether normalization does something desirable in terms of EA dynamics by running LTGA on various optimization problems.

4 Experiments

4.1 Optimization Problems

We consider three well-known benchmark problems from linkage-learning literature and one well-known NP-hard problem, all of which need to be maximized. The first problem is onemax in which every variable is independent of the others:

$$f_{\text{Onemax}}(\boldsymbol{x}) = \sum_{i=0}^{l-1} x_i$$

The second problem is the mutually-exclusive, additively decomposable sum of the well-known order-k deceptive trap functions [2] with $k = 5$:

$$f_{\text{Trap5}}(\boldsymbol{x}) = \sum_{i=0}^{(l/k)-1} f^{\text{sub}}_{\text{Trap-k}}\left(\sum_{j=ki}^{ki+k-1} x_j\right) \quad \text{, with } f^{\text{sub}}_{\text{Trap-k}}(u) = \begin{cases} 1 & \text{if } u = k \\ \frac{k-1-u}{k} & \textit{otherwise} \end{cases}$$

It is commonly known that the linkage groups pertaining to the subfunctions need to be detected and processed in order for optimization to proceed efficiently.

The third problem is the nearest-neighbour overlapping, additively decomposable sum of a-priori randomly generated subfunctions of length k, which constitutes a NK-landscape [6]. We use $k = 5$ and the maximum overlap of 4, but without wraparound:

$$f_{\text{NK-S1}}(\boldsymbol{x}) = \sum_{i=0}^{l-k} f^{\text{sub}}_{\text{NK}}\left(\boldsymbol{x}_{(i,i+1,\ldots,i+k-1)}\right)$$

where $f^{\text{sub}}_{\text{NK}}\left(\boldsymbol{x}_{(i,i+1,\ldots,i+k-1)}\right)$ is an a-priori randomly chosen value in $[0;1]$.

The NP-hard problem we consider is weighted MAXCUT. It is defined given a weighted undirected graph with a set of l vertices $V = \{v_0, v_1, \ldots, v_{l-1}\}$, a set of edges E between the vertices, and a weight w_{ij} for each edge $(v_i, v_j) \in E$. The goal is to split V into two sets such that the combined weight of edges that are thereby cut, i.e. running between vertices in different sets, is maximized. By introducing a binary variable x_i for every vertex that indicates if vertex v_i is either in set 0 or set 1, the function to be optimized is therefore:

$$f_{\text{weighted MAXCUT}}(\boldsymbol{x}) = \sum_{(v_i,v_j)\in E} \begin{cases} w_{ij} \text{ if } x_i \neq x_j \\ 0 \quad \textit{otherwise} \end{cases}$$

We encode this problem straightforwardly using the x_i directly. Benchmarks problem instances of various types and sizes exist in literature, but because we want to perform a controlled scalability analysis, we generated our own instances. For problem sizes $l \in \{6, 12, 25, 50, 100\}$ we generated fully connected graphs with $\frac{1}{2}l(l-1)$ edges. To set the weights, we follow the approach by Rubinstein [8] to obtain interesting instances and choose them randomly following a β distribution with parameters $\alpha = 100$, $\beta = 1$ and scaled to the range of $[1;5]$. For each problem size, we generate 10 instances. The maximum problem

size was chosen such that the exact optimizer BIQMAC [7] could provide optimal solutions within reasonable time. Because MAXCUT is NP-hard, we will consider both obtaining the optimum as well as obtaining 95% of the optimum where we accounted for the average random value ARV of an instance by setting the actual target value for an instance to $ARV + 0.95(OPT - ARV)$. The ARV is determined empirically by averaging over many randomly sampled solutions.

4.2 Experimental Setup

We say that a problem is solved if at least 99 out of 100 independent runs converged to either the global optimum or to a predefined sufficiently close approximation. Moreover, instead of stopping when the target is reached, we run until convergence (all solutions are the same) because we feel this is more realistic in practice where the optimum is not known beforehand and outcomes are typically collected upon termination.

For NK-S1, we have generated 100 instances per problem size randomly. The 100 independent runs are performed on 100 different, but always the same 100, instances per problem size. For weighted MAXCUT however, every problem instance is considered separately as is more typical of combinatorial optimization literature. The reason for this is that specific instances may be easy or hard and their properties may be interesting to study separately. Therefore, for weighted MAXCUT 100 independent runs are performed per problem instance. Note that this alters the interpretation of the range of outcomes where we aggregate per problem size. For weighted MAXCUT these ranges will be much larger.

A key performance indicator is scalability, i.e. how do the required resources (population size n and number of required evaluations) increase as the problem gets larger. To study this, we determine, for various problem sizes, the minimally required n to solve the problem. To do so, we perform a bisection search in which, starting from $n = 1$, n is doubled until the problem is solved. Subsequently, a binary search is performed in the range between the last two tested values for n. Because this process is still subject to noise, rooted in the stochastic nature of EAs, we perform 10 independent bisection searches for every problem size (and in case of weighted MAXCUT, for every problem instance).

4.3 Results

In Figure 4, the minimally required population size and the associated number of evaluations (upon convergence) are shown. The outcomes of the 10 independent bisection searches are shown for each problem size (for each instance in case of weighted MAXCUT). A least-squares polynomial fit of the form $\alpha \cdot l^{\beta}$ is also shown except for solving MAXCUT to optimality as it doesn't fit the data well. Instead, the lines we show connect the average values for each problem size.

All variants are virtually indistinguishable on onemax because the LT always contains every variable in a singleton set. However, close examination shows that the variant that uses H scales worst. These results are clearer on Trap 5 where only MI and NVI are virtually indistinguishable. Most clear are the results on NK-S1 where MI and NVI are again virtually indistinguishable but H

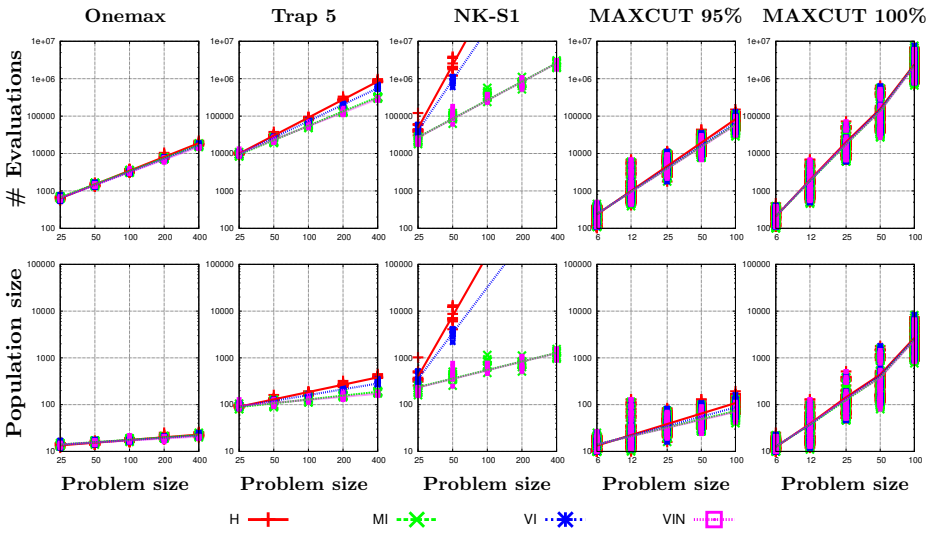


Fig. 4. Scale-up of LTGA on all problems, using different measures to build the LT

and VI clearly guide LTGA less efficiently. Combined with our analysis in Section 3, these results suggest that using LTGA on NK-S1 some variables converge much faster than others. Once converged, VI and, even more so, H favor these variables much stronger. Not only is this inefficient with respect to finding and mixing building blocks amongst variables where diversity still remains, it also creates strongly unbalanced trees with many large clusters, further reducing the efficiency of optimal mixing. MI and NVI are not affected by this cascading effect of converging variables. Finally, it appears that for solving our randomly generated weighted MAXCUT instances proper linkage learning is not as crucial because just like on onemax, all variants scale quite similarly. Differences do seem to increase with problem size, although far less severly. Proper linkage learning may still be required when solving instances with specific structure, especially when targeting the global optimum. A more in-depth study on weighted MAXCUT and the impact of linkage learning will be topic of future research.

As expected from Section 3, LTGA performs best when the measure used is MI or NVI (or, equivalently, MNI). For these two alternatives, using a Mann-Whitney U test at a significance level of 1%, results differ only for onemax with $l = 400$ (in favor of NVI), trap 5 with $l = 400$ (in favor of NVI), never for NK-S1, never for weighted MAXCUT 95% and only for one instance with $l = 100$ for weighted MAXCUT 100% (in favor of MI). Arguably therefore, it is virtually impossible to prefer one measure over another (based on the selected problems).

5 Conclusions

The linkage tree genetic algorithm (LTGA) was previously combined with different measures for building its linkage model (the linkage tree). In this paper

we took a closer look at these measures, related them to the convergence of an EA and we identified potential biases in the measures. The closest correspondence to the notion of a building block was found for the mutual information (MI) measure and a normalized (MNI) variant that is obtained by dividing by joint entropy. MNI is equivalent to the normalized variation of information (NVI) measure, even when using the less-computationally demanding pairwise measure adaptation known as UPGMA. The difference between MI and MNI/NVI is that the former has a slight preference for less converged variables. These measures only disagree in less than 4%, and when they do differ, these differences are very small. Consequently, LTGA was found to perform very similarly for these two measures on a set of three benchmark problems from linkage learning literature as well as on a combinatorial optimization problem: weighted MAXCUT. Only very few statistically significant differences could be found in the performance of LTGA using MI or NVI/MNI and even then the results were very close.

References

1. Bosman, P.A.N., Thierens, D.: Linkage neighbors, optimal mixing and forced improvements in genetic algorithms. In: Proc. of the Genetic and Evolutionary Computation Conf., GECCO 2012. ACM Press, New York (to appear, 2012)
2. Deb, K., Goldberg, D.E.: Sufficient conditions for arbitrary binary functions. Annals of Mathematics and Artificial Intelligence 10(4), 385–408 (1994)
3. Harik, G.R., Lobo, F.G., Sastry, K.: Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In: Pelikan, M., et al. (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, pp. 39–61. Springer, Berlin (2006)
4. Kraskov, A., Grassberger, P.: MIC: Mutual information based hierarchical clustering. In: Emmert-Streib, F., Dehmer, M. (eds.) Knowledge Incorporation in Evolutionary Computation, pp. 101–123. Springer, Berlin (2009)
5. Pelikan, M., Hauschild, M.W., Thierens, D.: Pairwise and problem-specific distance metrics in the linkage tree genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, pp. 1005–1012. ACM Press, New York (2011)
6. Pelikan, M., Sastry, K., Goldberg, D.E., Butz, M.V., Hauschild, M.: Performance of evolutionary algorithms on NK landscapes with nearest neighbor interactions and tunable overlap. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2009, pp. 851–858. ACM Press, New York (2009)
7. Rendl, F., Rinaldi, G., Wiegele, A.: Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. Math. Prog. 121(2), 307 (2010)
8. Rubinstein, R.Y.: Cross-entropy and rare events for maximal cut and partition problems. ACM Trans. on Modeling and Computer Simulation 12(1), 27–53 (2002)
9. Thierens, D.: The Linkage Tree Genetic Algorithm. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 264–273. Springer, Heidelberg (2010)
10. Thierens, D., Bosman, P.A.N.: Optimal mixing evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, pp. 617–624. ACM Press, New York (2011)

Evolvability Analysis of the Linkage Tree Genetic Algorithm

Dirk Thierens[1] and Peter A.N. Bosman[2]

[1] Institute of Information and Computing Sciences
Universiteit Utrecht, The Netherlands
D.Thierens@uu.nl
[2] Centre for Mathematics and Computer Science
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Peter.Bosman@cwi.nl

Abstract. We define the linkage model evolvability and the evolvability-based fitness distance correlation. These measures give an insight in the search characteristics of linkage model building genetic algorithms. We apply them on the linkage tree genetic algorithm for deceptive trap functions and the nearest-neighbor NK-landscape problem. Comparisons are made between linkage trees, based on mutual information, and random trees which ignore similarity in the population. On a deceptive trap function, the measures clearly show that by learning the linkage tree the problem becomes easy for the LTGA. On the nearest-neighbor NK-landscape the evolvability analysis shows that the LTGA does capture enough of the structure of the problem to solve it reliably and efficiently even though the linkage tree cannot represent the overlapping epistatic information in the NK-problem. The linkage model evolvability measure and the evolvability-based fitness distance correlation prove to be useful tools to get an insight into the search properties of linkage model building genetic algorithms.

1 Introduction

Linkage learning genetic algorithms aim to identify interacting or dependent problem variables that contribute to highly fit solutions. The goal is to build a linkage model of these interactions and use this model to generate, with high probability, new highly fit solutions.

To better understand how this class of algorithms searches for optimal solutions we introduce two measures: the linkage model evolvability and the evolvability-based fitness distance correlation. These measures have their origin in the evolvability measure and fitness distance correlation coefficient for general evolutionary algorithms. We exploit the particularities of linkage models to design more informative evolvability and correlation measures.

The paper is organized as follows. In the next section we describe the linkage tree learning and the optimal mixing evolutionary algorithm. Section 3 introduces the linkage model evolvability measure and the evolvability-based fitness

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 286–295, 2012.

distance correlation. In Section 4 we compute these measures for the LTGA on a deceptive trap function and a nearest-neighbor NK-landscape problem. Finally, Section 5 concludes the paper.

2 Linkage Tree Genetic Algorithm

We give a short review of the LTGA, for more details we refer the reader to [9].

The linkage tree (LT) is a hierarchical linkage model obtained by a bottom-up agglomerative hierarchical clustering algorithm starting from the set of problem variable singletons [2,8]. For a problem of length ℓ the linkage tree has ℓ leaf nodes (the clusters having a single problem variable) and $\ell-1$ internal nodes. The similarity measure is for instance the normalized mutual information between two subsets of variables. An efficient method to compute the similarity measure is the average linkage clustering or unweighted pair group method with arithmetic mean (UPGMA) [6,9]. Given a population of size N the LT can be built in $\mathcal{O}(N\ell^2)$ time using the reciprocal nearest-neighbor chain algorithm.

The LT specifies a set of problem variable subsets which are a specific example of the more general family of subsets (FOS). The class of linkage model building GAs we consider here specify their linkage model by this FOS model. Mathematically, the FOS model is a subset of the power set of the problem variables. This FOS model is used by the Gene Pool Optimal Mixing Evolutionary Algorithm (GOMEA) to generate new solutions [9]. Each subset of problem variables is used as a crossover mask. Each solution of the population is iteratively used as parent solution. For each parent solution the entire FOS set is traversed: for each problem variable subset in the FOS model a random solution is picked from the population as donor. The donor's values of the problem variables specified by the subset - or crossover mask - are copied to the parent solution. This new solution is evaluated and when it is an improvement of the parent, the offspring replaces the parent. Next , the traversal of the FOS model continues with the new solution. If there was no fitness improvement the FOS model is further traversed using the parent solution. New solutions are thus only accepted when they have a better fitness value than the parent solution. When the tree is completely traversed, the current parent solution is copied to the next generation's population. This tree traversal process is done for each solution in the current generation. The LT has $2\ell-2$ subsets in the tree (the top node is ignored because it contains all problem variables), so in one generation there are at most $N(2\ell-2)$ offspring generated and evaluated. This value is an upper boundary because before evaluating a new solution we always first check whether the generated solution is really different from the parent solution.

The reason for calling this operation optimal mixing is due to its inception in the original LTGA [8] where a two-parent crossover operator was used following the LT with intermediate evaluations to check for improvements. Given only two parents, traversing a FOS while performing crossover then ensures that all building blocks as described in the FOS wind up in one of the two parents and mixing therefore can be said to be optimal. However, it was recently shown that

considering a randomly selected new donor parent for each element in the FOS while performing crossover, which was called Gene-pool Optimal Mixing (GOM) is overall a more efficient manner of mixing because this removes the covariance in mixing that is inherent in using the same two parents for the entire crossover operation [9].

The use of OM allows making use of the linkage information as provided by all subsets in the FOS. OM further strongly increases the selection pressure in a building-block-wise manner instead of on a entire solution basis, the latter of which is a source of noise in deciding well between building blocks that increases the population size. It is through OM that the LTGA is capable of working with very small population sizes compared to most linkage learning EAs of the Estimation of Distribution Algorithm (EDA) type.

3 Linkage Model Evolvability Analysis

When applying a genetic algorithm - and more generally, any metaheuristic search algorithm - to a specific problem, we often would like to get a clear picture of how the search dynamics proceeds. We would like to possess a few measures that can give us a better understanding of what is going on during the search. Ideally, these measurements would help us explain why a certain algorithm succeeds in finding good solutions while others do not. In this paper we will use four measurements: two existing and two that we define specifically for linkage model building GAs where the model is a family of subsets (FOS).

Hamming Distance. The first and most obvious measure to trace the convergence progress is the minimum and median hamming distance from the solutions in the current population towards the global optimum.

Evolvability. The ultimate goal of search operators like crossover or mutation is to create new solutions that have a better fitness than their parent(s). The probability that this occurs has been called *probability of success* in the evolution strategy literature, while Altenberg called this the *evolvability* [1]. For an evolutionary algorithm to be successful it is important that the evolvability remains positive when the search has reached the higher regions of the fitness landscape, or when the parents have high fitness values. To compute this we split up the higher fitness values in bins and count the frequency of generating fitter offspring when the parent has a fitness value within the fitness bin's range. The parents are taken from actual runs, not from random samples. This is important as the key question here is whether the evolvability remains positive as the search approaches the most fit solutions in the search space.

Linkage Model Evolvability. The evolvability as a function of the parental fitness does not give any insight in the contribution of the different masks in the linkage FOS model. To investigate their importance in the search process we calculate the evolvability as a function of the size of the masks - or FOS subsets - of successive linkage trees during an actual LTGA run. For the benchmark

functions used in this paper, the successive application of five linkage trees is sufficient to reliably find the global optimum, at least if the model learning is done properly. In an actual LTGA run each mask of the tree is used N times (N being the population size): for each solution in the population the linkage tree is traversed and a random solution is picked as donor. Counting the evolvability based on these offspring represents a limited sample of the actual evolvability potential of the linkage tree for the current population. In order to get a less noisy measurement we have calculated the evolvability by taking each solution in the population as the donor solution, as opposed to a single random solution. This way each mask is used N^2 times and the number of improvements are counted. We also compute the relative number of improvements as the percentage of offspring more fit than their parent (excluding the donor) of all the offspring generated by a mask of a given size using a particular linkage tree. It is important to note that this calculation of the linkage masks' evolvability has no influence on the actual LTGA run. The offspring generated during this calculation are not used in the actual LTGA run.

Evolvability-Based Fitness Distance Correlation. Although the evolvability measures the potential of an evolutionary algorithm to keep finding new and better solutions it does not consider whether the population is actually converging towards the global optimal solution. An EA could well be capable of generating many better offspring during the search process but that is not a guarantee that it is actually getting any closer to the optimum. To measure the progress towards the optimum Jones and Forrest [4] introduced the concept of fitness distance correlation (FDC). As its name implies the FDC measures the correlation between the fitness value and the hamming distance towards the global optimal solution. A large negative correlation is seen as an indication that the GA would be guided towards the optimum by following a path of ever improving solutions. A large positive correlation is interpreted as a deceiving problem: following a path of better solutions would lead away from the optimal solution. The FDC is actually a search ignorant measure in the sense that it does not include any information about the capability of the genetic operators to generate improving solutions. FDC only looks at the representation and the fitness values of the solutions, not at the actual dynamics of the GA run. There is little to be gained from a high negative FDC measure if the genetic operators are unable to generate the high fitness solutions in the first place. Altenberg [1] discussed these limitations of a Hamming-distance based FDC and proposed two crossover-distance based FDC measures (XFDC). The XFDC measures aim to include the role of the genetic operators. The first XFDC defined the crossover distance as the number of discontinuities between 0s and 1s in a solution's bitstring. This measure is clearly only suitable for the specific test function used in that paper which is based directly on this number of discontinuities. A second more general crossover-based distance measure is computed by running crossover in reverse. Starting from the global optimum and its binary complement bitstrings are given a crossover distance of 1 when they are generated by a single

application of the crossover operator. A second set of strings is given a distance of 2 by applying crossover to the previous set. Continuing this way a sequence of populations is generated with increasing crossover distance. The XFDC measure is now computed by calculating the correlation coefficient between the crossover distance and the fitness value. It is clear that both XFDC measures are only rough approximations of the actual GA dynamics. The main problem of getting a more accurate measure is the vast amount of different crossover events that are possible.

For the LTGA however, the number of possible crossover events is much smaller. In fact, when computing the linkage model evolvability, we are already looking at all possible outcomes for a specific linkage FOS model and a given population. The only thing we need to add is the correlation with approaching the optimal solution. Therefore we define the evolvability-based fitness distance correlation (EFDC) as the correlation between the Hamming distance between the offspring and the optimal solution and the amount of fitness gain whenever an improvement occurred during the calculation of the linkage masks' evolvability. The EFDC is a much more informative measure of the search dynamics than the FDC or XFDC measures.

In [3] the fitness distance correlation is computed for fixed neighborhoods that match the structure of the fitness function. The EFDC however is computed during the search and depends on the specific linkage tree built each generation, thus capturing more of the dynamics of the search process.

4 Experimental Analysis

To test whether the evolvability measures do indeed provide any insight in the search behavior of the LTGA we compare the linkage tree with a randomly build tree on 2 benchmarks. First, we consider the deceptive trap function [2]. This function is interesting for our purposes here because a linkage learning algorithm must be able to learn the structure of the problem in order to find the optimal solution. A randomly constructed tree will be unable to do this, and the interesting question is how this gets reflected in the evolvability measures. Our second test function is the nearest-neighbor NK-landscape [7]. This function is interesting because the overlap of the subfunctions cannot be represented by a linkage tree, and yet the LTGA is capable of consistently finding the optimal solution. The key question is whether the evolvability measures can help explain why this is the case.

Evolvability measures should give insight in the particular behavior of an actual GA run. We therefore compute the values on one single run, and not average them out over a whole set of runs. Of course, this run should be representative and we compared the results on different runs. As it turned out, all runs had basically the same behavior and similar evolvability values, so we only report here the results of one single run for both the benchmark functions.

4.1 Deceptive Trap Function

The deceptive trap function used here consists of 10 subfunctions of length 5 (stringlength $\ell = 50$), fitness value of the subfunctions is 5 for the optimum and 4 for the deceptive attractor. The population size is 100. The initial population is generated by performing a single-pass bit-flip local search on a population of random solutions.

Table 1 shows the Hamming distance between the optimal string and the best and the median solution in the population for successive linkage trees. When the linkage tree is learned using normalized mutual information, the population quickly converges to the optimal solution. Both the minimum and median Hamming distance are reduced by each new generation, and the optimal solution is generated at the fifth generation. The table also shows the Hamming distance when the linkage tree is built using random numbers as similarity measure instead of normalized mutual information. Clearly, without linkage learning the search algorithm does not make a lot of progress in finding the optimal solution. Only looking at the Hamming distance however does not make it clear whether the search is not making progress at all or it is simply going the right direction but at a very small pace.

Table 1. Hamming distance towards the global optimum for the deceptive trap function

distance	**Linkage tree**					**Random tree**				
	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
minimum	25	20	15	5	0	30	30	30	30	30
median	45	40	30	20	10	45	45	40	40	35

Table 2 shows the frequency of improvements as a function of parental fitness. The fitness is divided in 10 bins, spanning the fitness range from the solution with all deceptive attractors to the global optimum. Clearly, the search without linkage learning is not able to make any substantial progress towards the optimal solution. The probability of generating better offspring is nearly zero, and no offspring with a fitness value of 45 or higher are created. When linkage learning is done properly we see that better offspring are created in a consistent way, and the evolvability remains positive with increasing parental fitness values.

The linkage model evolvability in Table 3 gives a more detailed picture of the evolvability in the linkage tree. The table only shows the masks where a fitness improvement takes place. For the linkage learning the masks' sizes are all multiples of five which reflects the building block length of the deceptive trap function. The percentage of improvements is quite high for all the masks in the table, indicating a very efficient search process.

For the random tree - this is, no linkage learning - there are very few fitness improvements, and the vast majority of them are achieved with crossover masks of length 48 and 49. For a stringlength of size 50 this basically means that the donor solution is better than the parent and the large masks are simply making an almost complete copy of the donor. Obviously, this does not lead to good novel solutions.

Table 2. Evolvability: frequency of improvements as a function of parental fitness

Fitness	**Linkage tree**					**Random tree**				
range	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
[40, 41[	0.29	0.27	0.26	0.22	–	0.03	0	0	–	–
[41, 42[	0.10	0.16	0.20	0.23	–	0	0	0	0	0.01
[42, 43[	0.01	0.05	0.15	0.21	0.17	0	0	0	0	0.01
[43, 44[	0.03	0.02	0.05	0.18	0.24	0	0	0	0	0
[44, 45[	0.01	0.02	0.03	0.12	0.19	–	0	0	0	0
[45, 46[	0	0.01	0.02	0.04	0.15	–	–	–	–	–
[46, 47[	–	0	0.01	0.03	0.14	–	–	–	–	–
[47, 48[	–	–	0	0.02	0.08	–	–	–	–	–
[48, 49[	–	–	–	0.01	0.02	–	–	–	–	–
[49, 50[	–	–	–	0	0.01	–	–	–	–	–

Table 3. Linkage model evolvability for the deceptive trap function

Linkage tree										
Mask size	Improvements					% improvements				
	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
5	10892	16808	21916	22648	16192	11%	17%	22%	23%	16%
10	7176	9474	8995	12726	7462	18%	24%	30%	32%	25%
15	5152	6032	10080	3661	5668	26%	30%	34%	37%	29%
20	2147	3318	3677	3846	–	22%	33%	37%	39%	–
30	–	–	3712	3931	3720	–	–	37%	40%	38%
35	3285	3663	–	–	–	33%	37%	–	–	–
40	–	–	–	–	3975	–	–	–	–	40%
45	–	–	–	4112	4002	–	–	–	41%	40%
Tot.	28652	39295	48380	50924	41019					
Random tree										
Mask size	Improvements					% improvements				
	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
22	269	–	–	–	–	1.3%	–	–	–	
29	–	62	–	–	–	–	0.01%	–	–	
39	–	–	130	–	110	–	–	0.01%	–	0.01%
41	–	–	–	40	–	–	–	–	0.00%	
42	–	–	–	40	–	–	–	–	0.00%	
43	–	–	130	–	–	–	–	0.01%	–	
44	–	–	–	–	310	–	–	–	–	0.03%
48	–	–	–	–	3351	–	–	–	–	34%
49	2095	–	–	–	–	21%	–	–	–	
Tot.	2364	62	260	80	3771					

Finally, Table 4 shows the EFDC measure. In case of linkage learning there is a perfect linear relationship between the amount of fitness improvement and the reduction in Hamming distance towards the optimal solution. This makes sense as the crossover masks exactly match the building blocks and fitness improvements of 1, 2, 3, ... correspond to Hamming distance reductions of 5, 10, 15, ... When there is no linkage learning there is usually also no correlation coefficient to compute since there are either zero or only one single pair of fitness improvement and Hamming distance value. Only in the first random linkage tree large masks can sometimes get a fitness improvement.

Table 4. Evolvability-based Fitness Distance Correlation for the deceptive trap function

Linkage tree					Random tree				
Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	–	–	–	–

4.2 Nearest-Neighbor NK-Landscape

The nearest-neighbor NK-landscape used here has stringlength $\ell = 50$, the sub-functions have length 5 bits, and the overlap is maximal - that is, 4 bits. The population size is $N = 200$.

Table 5. Hamming distance towards the global optimum for the NK-landscape

distance	Linkage tree					Random tree				
	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
minimum	7	3	0	0	0	7	7	7	7	7
median	20	18	15	12	6	21	20	19	19	18

Table 6. Evolvability: frequency of improvements as a function of parental fitness

Fitness	Linkage tree					Random tree				
Bins	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
1	0.21	–	–	–	–	0.06	–	–	–	–
2	0.17	–	–	–	–	0.08	–	–	–	–
3	0.16	0.26	–	–	–	0.06	0.09	–	–	–
4	0.10	0.12	–	–	–	0.04	0.04	0.05	–	–
5	0.07	0.11	0.10	–	–	0.03	0.03	0.03	0.02	0.00
6	0.04	0.08	0.06	0.10	–	0.02	0.02	0.02	0.02	0.02
7	0.04	0.04	0.05	0.10	0.13	0.03	0.01	0.01	0.01	0.01
8	0.02	0.02	0.03	0.05	0.11	0.00	0.00	0.01	0.00	0.01
9	0.00	0.02	0.02	0.03	0.06	–	–	–	–	–
10	–	0.01	0.02	0.02	0.02	–	–	–	–	–

As for the deceptive trap function, we look at the impact of linkage learning by comparing the results with a randomly constructed linkage tree. A linkage tree has $2\ell - 2$ nodes that are used as crossover masks.

Table 5 shows the Hamming distance for the linkage tree and the random tree models. As shown in previous work the linkage tree has no trouble in finding the optimal solution, while the random tree is getting nowhere.

Table 6 shows the evolvability as a function of the parental fitness. We have divided the fitness range between the fitness value *low* and the global optimal value in 10 bins of equal width. The fitness value *low* is the fitness of the least fit solution of a population of 100 single-pass bit-flipped solutions. Whenever an improving solution is generated the bin corresponding to the parent's fitness is updated. The evolvability values for the linkage tree model shows that the successive linkage trees are capable of generating new and more fit solutions with increasing parental fitness. On the contrary, the random trees are not able

Table 7. Linkage model evolvability for the NK-landscape

Linkage tree										
Mask size	Improvements					% improvements				
	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
1	31205	13358	6169	3650	2765	1.5%	0.6%	0.3%	0.2%	0.1%
2	29523	12345	5638	4019	3943	3.9%	1.6%	0.8%	0.5%	0.5%
3	17308	5446	4273	3694	1354	4.8%	1.7%	1.2%	1%	0.5%
4	12409	5278	6732	6479	6953	7.8%	3.3%	4.2%	4%	2.9%
5	14053	12519	8730	6580	2044	7.1%	6.3%	5.5%	5.5%	2.6%
6–10	36673	34266	18561	12871	16274	16%	11%	7%	5%	5.7%
11–20	16686	25813	33354	24045	48664	14%	34%	16%	22%	20%
21-30	6530	14800	–	46270	–	16%	37%	–	29%	–
31–40	14412	15726	15621	–	12502	36%	40%	39%	–	31%
41–50	–	–	–	–	–	–	–	–	–	–
Tot.	178799	139551	99078	107608	94499					
Random tree										
Mask size	Improvements					% improvements				
	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
1	41040	23516	14635	9587	7081	2.1%	1.2%	0.7%	0.5%	0.4%
2	16361	7792	6361	3330	2228	2.9%	1.4%	1.2%	0.5%	0.4%
3	7914	4648	4043	1333	1180	3.3%	1.7%	1.3%	0.4%	0.6%
4	4247	1246	3409	1355	1040	2.7%	1.6%	1.4%	0.9%	0.4%
5	2632	1859	–	1019	328	2.2%	1.6%	–	0.9%	0.3%
6–10	5246	4276	2738	2167	309	1.3%	0.9%	0.8%	0.6%	0.1%
11–20	1233	1121	821	586	241	0.4%	0.3%	0.3%	%	0.2%
21–30	98	279	448	287	567	0.2%	0.3%	0.4%	0.4%	0.3%
31–40	510	–	–	–	503	1.3%	–	–	–	0.6%
41–50	12323	6350	10226	–	10733	31%	16%	13%	–	6.7%
Tot.	91604	51087	42681	19664	24210					

Table 8. Evolvability-based Fitness Distance Correlation for the NK-landscape

Linkage tree					**Random tree**				
Tree 1	Tree 2	Tree 3	Tree 4	Tree 5	Tree 1	Tree 2	Tree 3	Tree 4	Tree 5
-0.26	-0.44	-0.47	-0.58	-0.41	-0.27	-0.13	-0.21	0.07	-0.32

to generate improving solutions above a certain fitness level. Successive trees are not able to generate new solutions that fall in the higher valued bins. The search clearly stagnates in the lower valued fitness bins.

In Table 7 we see the evolvability contributions of different crossover masks. The linkage tree model has a much higher evolvability than the random tree in both absolute as relative measures. It is interesting to see that all mask sizes contribute to the search, which shows that the linkage tree's capability of representing interacting problem variables at multiple levels is beneficial to the search.

Finally, Table 8 shows the evolvability-based fitness distance correlation. For the five successive linkage trees the EFDC remains significantly negative, meaning that fitness gains are correlated with reductions in Hamming distance to the optimal solution. For the random linkage trees the EFDC are also negative but have a lower value, except for the first generation trees. It appears that in the first generation the solutions can be easily improved and the Hamming distance towards the optimal solution is reduced. If we look again at Table 7 we see that most of these improvements for the random tree are obtained with masks of length

1 and 2, so the offspring are only 1 or 2 bits different from to their parents but do have a higher fitness and are mostly closer in Hamming distance to the optimal bitstring. It is also noteworthy that the EFDC for the fifth tree of the random models has a rather high correlation ($= -0.32$). Looking again at Table 7 reveals that almost half of the improvements are obtained by masks of size larger than 40: the high correlation can thus be explained by the copying effect of good donor solutions by large masks. Although these improvements increase the EFDC value they do not significantly contribute to finding new good solutions.

5 Conclusion

We have analyzed the evolvability of the linkage tree genetic algorithm. For this, we have defined the linkage model evolvability and the evolvability-based fitness distance correlation. We have seen how these measures give an insight in the performance of the LTGA. We have also made a comparison with a randomly constructed tree and discussed the differences observed in the evolvability measures. On a deceptive trap function, the measures clearly show that learning the linkage tree makes this an easy problem for the LTGA. On the nearest-neighbor NK-landscape the evolvability analysis shows that the LTGA does capture enough of the structure of the problem to solve it reliably and efficiently even though the linkage tree cannot represent the overlapping epistatic information in the NK-problem. We believe that measures like the linkage model evolvability and the evolvability-based fitness distance correlation are useful tools to describe and understand the characteristics of linkage model building genetic algorithms.

References

1. Altenberg, L.: Fitness distance correlation analysis: An instructive counterexample. In: Proc. 7th Intern. Conf. Genetic Algorithms, pp. 57–64. Morgan Kaufmann (1997)
2. Duque, T.S., Goldberg, D.E.: A new method for linkage learning in the ECGA. In: Proc. 11th Int. Conf. Genetic and Evol. Computation, pp. 1819–1820. ACM (2009)
3. Hauschild, M., Pelikan, M.: Advanced neighborhoods and problem difficulty measures. In: Krasnogor, et al. (eds.) [5], pp. 625–632. ACM (2011)
4. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 184–192. Morgan Kaufmann (1995)
5. Krasnogor, N., Lanzi, P.L. (eds.): Proc. 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011. ACM (2011)
6. Pelikan, M., Hauschild, M., Thierens, D.: Pairwise and problem-specific distance metrics in the linkage tree genetic algorithm. In: Krasnogor et al. [5], pp. 1005–1012
7. Pelikan, M., Sastry, K., Goldberg, D.E., Butz, M.V., Hauschild, M.: Performance of evolutionary algorithms on NK-landscapes with nearest neighbor interactions and tunable overlap. In: Raidl, G. (ed.) GECCO, pp. 851–858. ACM (2009)
8. Thierens, D.: The Linkage Tree Genetic Algorithm. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 264–273. Springer, Heidelberg (2010)
9. Thierens, D., Bosman, P.A.N.: Optimal mixing evolutionary algorithms. In: Krasnogor and Lanzi [5], pp. 617–624

Alternative Restart Strategies for CMA-ES*

Ilya Loshchilov[1,2], Marc Schoenauer[1,2], and Michèle Sebag[2,1]

[1] TAO Project-team, INRIA Saclay - Île-de-France
[2] Laboratoire de Recherche en Informatique (UMR CNRS 8623)
Université Paris-Sud, 91128 Orsay Cedex, France
FirstName.LastName@inria.fr

Abstract. This paper focuses on the restart strategy of CMA-ES on multi-modal functions. A first alternative strategy proceeds by decreasing the initial step-size of the mutation while doubling the population size at each restart. A second strategy adaptively allocates the computational budget among the restart settings in the BIPOP scheme. Both restart strategies are validated on the BBOB benchmark; their generality is also demonstrated on an independent real-world problem suite related to spacecraft trajectory optimization.

1 Introduction

The long tradition of performance of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm on real-world problems (with over 100 published applications [6]) is due among others to its good behavior on multi-modal functions. Two versions of CMA-ES with restarts have been proposed to handle multi-modal functions: IPOP-CMA-ES [2] was ranked first on the continuous optimization benchmark at CEC 2005 [4,3]; and BIPOP-CMA-ES [5] showed the best results together with IPOP-CMA-ES on the black-box optimization benchmark (BBOB) in 2009 and 2010.

This paper focuses on analyzing and improving the restart strategy of CMA-ES, viewed as a noisy hyper-parameter optimization problem in a 2D space (population size, initial step-size). Two restart strategies are defined. The first one, NIPOP-aCMA-ES (*New* IPOP-aCMA-ES), differs from IPOP-CMA-ES as it simultaneously increases the population size and decreases the step size. The second one, NBIPOP-aCMA-ES, allocates computational power to different restart settings depending on their current results. While these strategies have been designed with the BBOB benchmarks in mind [8], their generality is shown on a suite of real-world problems [16].

The paper is organized as follows. After describing the weighted active $(\mu/\mu_w, \lambda)$-CMA-ES and its current restart strategies (section 2), the proposed restart schemes are described in section 3. Section 4 reports on their experimental validation. The paper concludes with a discussion and some perspectives for further research.

* Work partially funded by FUI of System@tic Paris-Region ICT cluster through contract DGT 117 407 *Complex Systems Design Lab* (CSDL).

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 296–305, 2012.

2 The Weighted Active $(\mu/\mu_w, \lambda)$-CMA-ES

The CMA-ES algorithm is a stochastic optimizer, searching the continuous space $\mathbb{R}^D$ by sampling λ candidate solutions from a multivariate normal distribution [10,9]. It exploits the best μ solutions out of the λ ones to adaptively estimate the local covariance matrix of the objective function, in order to increase the probability of successful samples in the next iteration. The information about the remaining (worst $\lambda - \mu$) solutions is used only implicitly during the selection process.

In active $(\mu/\mu_I, \lambda)$-CMA-ES however, it has been shown that the worst solutions can be exploited to reduce the variance of the mutation distribution in unpromising directions [12], yielding a performance gain of a factor 2 for the active $(\mu/\mu_I, \lambda)$-CMA-ES with no loss of performance on any of tested functions. A recent extension of the $(\mu/\mu_w, \lambda)$-CMA-ES, *weighted* active CMA-ES [11] (referred to as aCMA-ES for brevity) shows comparable improvements on a set of noiseless and noisy functions from the BBOB benchmark suite [7]. In counterpart, aCMA-ES no longer guarantees the covariance matrix to be positive definite, possibly resulting in algorithmic instability. The instability issues can however be numerically controlled during the search; as a matter of fact they are never observed on the BBOB benchmark suite.

At iteration t, $(\mu/\mu_w, \lambda)$-CMA-ES samples λ individuals according to

$$\boldsymbol{x}_k^{(t+1)} \sim \mathcal{N}\left(\boldsymbol{m}^{(t)}, {\sigma^{(t)}}^2 \boldsymbol{C}^{(t)}\right), \quad k = 1 \ldots \lambda, \tag{1}$$

where $\mathcal{N}(\boldsymbol{m}, \boldsymbol{C})$ denotes a normally distributed random vector with mean $\boldsymbol{m}$ and covariance matrix $\boldsymbol{C}$.

These λ individuals are evaluated and ranked, where index $i : \lambda$ denotes the i-th best individual after the objective function. The mean of the distribution is updated and set to the weighted sum of the best μ individuals ($\boldsymbol{m} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}^{(t)}$, with $w_i > 0$ for $i = 1 \ldots \mu$ and $\sum_{i=1}^{\mu} w_i = 1$).

The active CMA-ES only differs from the original CMA-ES in the adaptation of the covariance matrix $\boldsymbol{C}^{(t)}$. Like for CMA-ES, the covariance matrix is computed from the best μ solutions, $\boldsymbol{C}_\mu^+ = \sum_{i=1}^{\mu} w_i \frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}^t}{\sigma^t} \times \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}^t)^T}{\sigma^t}$. The main novelty is to exploit the worst solutions to compute $\boldsymbol{C}_\mu^- = \sum_{i=0}^{\mu-1} w_{i+1} \boldsymbol{y}_{\lambda-i:\lambda} \boldsymbol{y}_{\lambda-i:\lambda}^T$, where $\boldsymbol{y}_{\lambda-i:\lambda} = \frac{\left\| {C^{t}}^{-1/2} (\boldsymbol{x}_{\lambda-\mu+1+i:\lambda} - \boldsymbol{m}^t) \right\|}{\left\| {C^{t}}^{-1/2} (\boldsymbol{x}_{\lambda-i:\lambda} - \boldsymbol{m}^t) \right\|} \times \frac{\boldsymbol{x}_{\lambda-i:\lambda} - \boldsymbol{m}^t}{\sigma^t}$. The covariance matrix estimation of these worst solutions is used to decrease the variance of the mutation distribution along these directions:

$$\begin{aligned} \boldsymbol{C}^{t+1} &= (1 - c_1 - c_\mu + c^- \alpha_{old}^-)\, \boldsymbol{C}^t + \\ &+ c_1 \boldsymbol{p}_c^{t+1} {\boldsymbol{p}_c^{t+1}}^T + (c_\mu + c^-(1 - \alpha_{old}^-))\, \boldsymbol{C}_\mu^+ - c^- \boldsymbol{C}_\mu^-, \end{aligned} \tag{2}$$

where $\boldsymbol{p}_c^{t+1}$ is adapted along the evolution path and coefficients c_1, c_μ, c^- and α_{old}^- are defined such that $c_1 + c_\mu - c^- \alpha_{old}^- \leq 1$. The interested reader is referred to [10,11] for a more detailed description of these algorithms.

As mentioned, CMA-ES has been extended with restart strategies to accommodate multi-modal fitness landscapes, and to specifically handle objective functions with many local optima. As observed by [9], the probability of reaching the optimum (and the overall number of function evaluations needed to do so) is very sensitive to the population size. The default population size $\lambda_{default}$ has been tuned for uni-modal functions; it is hardly large enough for multi-modal functions. Accordingly, [2] proposed a "doubling trick" restart strategy to enforce global search: the restart $(\mu/\mu_w, \lambda)$-CMA-ES with increasing population, called IPOP-CMA-ES, is a multi-restart strategy where the population size of the run is doubled in each restart until meeting a stopping criterion.

The BIPOP-CMA-ES instead considers two restart regimes. The first one, which corresponds to IPOP-CMA-ES, doubles the population size $\lambda_{large} = 2^{i_{restart}}\lambda_{default}$ in each restart $i_{restart}$ and uses a fixed initial step-size $\sigma^0_{large} = \sigma^0_{default}$.
The second regime uses a small population size λ_{small} and initial step-size σ^0_{small}, which are randomly drawn in each restart as:

$$\lambda_{small} = \left\lfloor \lambda_{default} \left(\frac{1}{2} \frac{\lambda_{large}}{\lambda_{default}} \right)^{U[0,1]^2} \right\rfloor, \quad \sigma^0_{small} = \sigma^0_{default} \times 10^{-2U[0,1]} \qquad (3)$$

where $U[0,1]$ stands for the uniform distribution in $[0,1]$. Population size λ_{small} thus varies $\in [\lambda_{default}, \lambda_{large}/2]$. BIPOP-CMA-ES launches the first run with default population size and initial step-size. In each restart, it selects the restart regime with less function evaluations. Clearly, the second regime consumes less function evaluations than the doubling regime; it is therefore launched more often.

3 Alternative Restart Strategies

3.1 Preliminary Analysis

The restart strategies of IPOP- and BIPOP-CMA-ES are viewed as a search in the hyper-parameter space.

IPOP-CMA-ES only aims at adjusting population size λ. It is motivated by the results observed on multi-modal problems [9], suggesting that the population size must be sufficiently large to handle problems with global structure. In such cases, a large population size is needed to uncover this global structure and to lead the algorithm to discover the global optimum. IPOP-CMA-ES thus increases the population size in each restart, irrespective of the results observed so far; at each restart, it launches a new CMA-ES with population size $\lambda = \rho_{inc}^{i_{restart}}\lambda_{default}$ (see $\circ$ on Fig. 1). Factor ρ_{inc} must be not too large to avoid "overjumping" some possibly optimal population size λ^*; it must also be not too small in order to reach λ^* in a reasonable number of restarts. The use of the doubling trick ($\rho_{inc} = 2$) guarantees that the loss in terms of function evaluations (compared to the "oracle" restart strategy which would directly set the population size to the optimal value λ^*) is about a factor of 2.

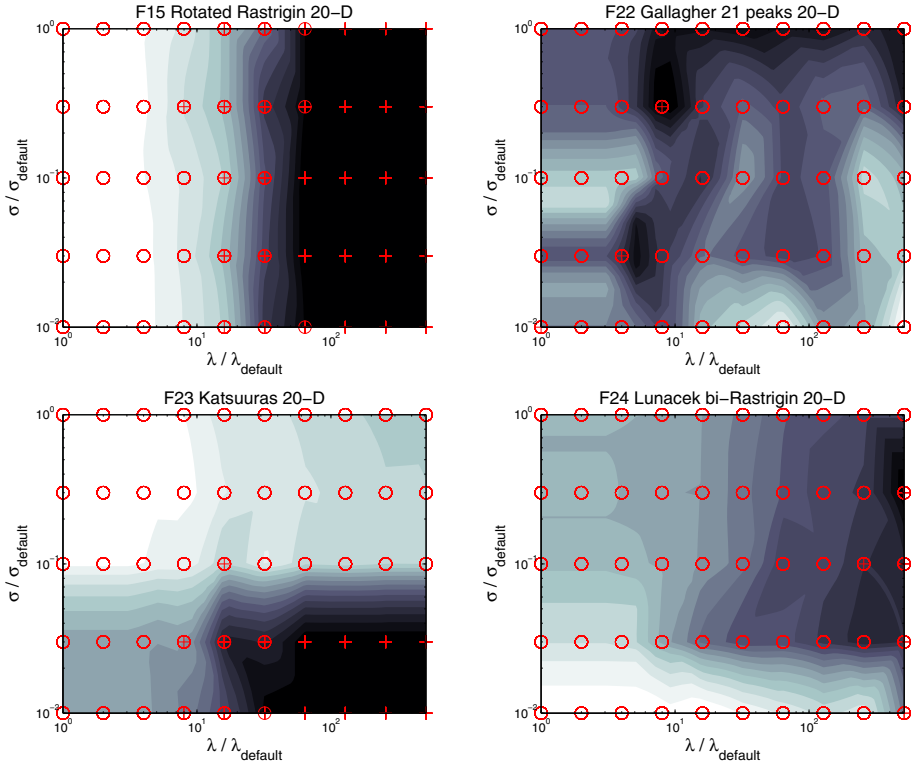


Fig. 1. Restart performances in the 2D hyper-parameter space (population size and initial mutation step size in log. coordinates). For each objective function (20 dimensional Rastrigin - top-left, Gallagher 21 peaks - top-right, Katsuuras - bottom-left and Lunacek bi-Rastrigin bottom-right), the median best function value out of 15 runs is indicated. Legends indicate that the optimum up to precision $f(x) = 10^{-10}$ is found always (+), sometimes ($\oplus$) or never ($\circ$). Black regions are better than white ones.

On the Rastrigin 20-D function, IPOP-CMA-ES performs well and always finds the optimum after about 5 restarts (Fig. 1, top-left). The Rastrigin function displays indeed a global structure where the optimum is the minimizer of this structure. For such functions, IPOP-CMA-ES certainly is the method of choice. For some other functions such as the Gallagher function, there is no such global structure; increasing the population size does not improve the results. On Katsuuras and Lunacek bi-Rastrigin functions, the optimum can only be found with small initial step-size (lesser than the default one); this explains why it can be solved by BIPOP-CMA-ES, sampling the two-dimensional (λ, σ) space.

Actually, the optimization of a multi-modal function by CMA-ES with restarts can be viewed as the optimization of the function $h(\theta)$, which returns the optimum found by CMA-ES defined by the hyper-parameters $\theta{=}(\lambda, \sigma)$. Function $h(\theta)$, graphically depicted in Fig. 1 can be viewed as a black box, computationally

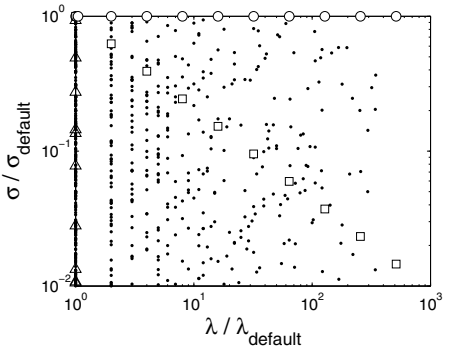


Fig. 2. An illustration of λ and σ hyper-parameters distribution for 9 restarts of IPOP-aCMA-ES ($\circ$), BIPOP-aCMA-ES ($\circ$ and $\cdot$ for 10 runs), NIPOP-aCMA-ES ($\square$) and NBIPOP-aCMA-ES ($\square$ and many $\triangle$ for $\lambda/\lambda_{default} = 1$, $\sigma/\sigma_{default} \in [10^{-2}, 10^{0}]$). The first run of all algorithms corresponds to the point with $\lambda/\lambda_{default} = 1$, $\sigma/\sigma_{default} = 1$.

expensive and stochastic function (reflecting the stochasticity of CMA-ES). Both IPOP-CMA-ES and BIPOP-CMA-ES are based on implicit assumptions about the $h(\theta)$: IPOP-CMA-ES achieves a deterministic uni-dimensional trajectory, and BIPOP-CMA-ES randomly samples the 2-dimensional search space.

Function $h(\theta)$ also can be viewed as a multi-objective fitness, since in addition to the solution found by CMA-ES, $h(\theta)$ could return the number of function evaluations needed to find that solution. $h(\theta)$ could also return the computational effort SP1 (i.e. the average number of function evaluations of all successful runs, divided by proportion of successful runs). However, SP1 can only be known for benchmark problems where the optimum is known; as the empirical optimum is used in lieu of true optimum, SP1 can only be computed *a posteriori*.

3.2 Algorithm

Two new restart strategies for CMA-ES, respectively referred to as NIPOP-aCMA-ES and NBIPOP-aCMA-ES, are presented in this paper.

If the restart strategy is restricted to the case of increasing of population size (IPOP), we propose to use NIPOP-aCMA-ES, where we additionally decrease the initial step-size by some factor $\rho_{\sigma dec}$. The rationale behind this approach is that the CMA-ES with relatively small initial step-size is able to explore small basins of attraction (see Katsuuras and Lunacek bi-Rastrigin functions on Fig. 1), while with initially large step-size and population size it will neglect the local structure of the function, but converge to the minimizer of the global structure. Moreover, initially, relatively small step-size will quickly increase if it makes sense, and this will allow the algorithm to recover the same global search properties than with initially large step-size (see Rastrigin function on Fig. 1).

NIPOP-CMA-ES thus explores the two-dimensional hyper-parameter space in a deterministic way (see $\square$ symbols on Fig. 2). For $\rho_{\sigma dec} = 1.6$ used in this

study, NIPOP-CMA-ES thus reaches the lower bound ($\sigma = 10^{-2}\sigma_{default}$) used by BIPOP-CMA-ES after 9 restarts, expectedly reaching the same performance as BIPOP-CMA-ES albeit it uses only a large population.

The second restart strategy, NBIPOP-aCMA-ES, addresses the case where the probability to find the global optimum does not much vary in the (λ, σ) space. Under this assumption, it makes sense to have many restarts for a fixed budget (number of function evaluations). Specifically, NBIPOP-aCMA-ES implements the competition of the NIPOP-aCMA-ES strategy (increasing λ and decreasing initial σ^0 in each restart) and a uniform sampling of the σ space, where λ is set to $\lambda_{default}$ and $\sigma^0 = \sigma^0_{default} \times 10^{-2U[0,1]}$ The selection between the two (NIPOP-aCMA-ES and the uniform sampling) depends on the allowed budget like in NBIPOP-aCMA-ES. The difference is that NBIPOP-aCMA-ES adaptively sets the budget allowed to each restart strategy, where the restart strategy leading to the overall best solution found so far is allowed twice ($\rho_{budget} = 2$) a budget compared to the other strategy.

4 Experimental Validation

The experimental validation of NIPOP-aCMA-ES and NBIPOP-aCMA-ES investigates the performance of the approach comparatively to IPOP-aCMA-ES and BIPOP-aCMA-ES on BBOB noiseless problems and one black-box real-world problem related to spacecraft trajectory optimization. The default parameters of CMA-ES [11,5] are used. This section also presents the first experimental study of BIPOP-aCMA-ES[1], the active version of BIPOP-CMA-ES [5].

4.1 Benchmarking with BBOB Framework

The BBOB framework [7] is made of 24 noiseless and 30 noisy functions [8]. Only the noiseless case has been considered here. Furthermore, only the 12 multi-modal functions among these 24 noiseless functions are of interest for this study, as CMA-ES can solve the 12 other functions without any restart.

With same experimental methodology as in [7], the results obtained on these benchmark functions are presented in Fig. 4 and Table 1. The results are given for dimension 40, because the differences are larger in higher dimensions. The **expected running time (ERT)**, used in the figures and table, depends on a given target function value, $f_t = f_{opt} + \Delta f$. It is computed over all relevant trials as the number of function evaluations required in order to reach f_t, summed over all 15 trials, and divided by the number of trials that actually reached f_t [7].

NIPOP-aCMA-ES. On 6 out of 12 test functions (f_{15},f_{16},f_{17},f_{18},f_{23},f_{24}) NIPOP-aCMA-ES obtains the best known results for BBOB-2009 and BBOB-2010 workshops. On f_{23} Katsuuras and f_{24} Lunacek bi-Rastrigin, NIPOP-aCMA-ES has a speedup of a factor from 2 to 3, as could have been expected. It performs

[1] For the sake of reproducibility, the source code for NIPOP-aCMA-ES and NBIPOP-aCMA-ES is available at https://sites.google.com/site/ppsnbipop/

unexpectedly well on f_{16} Weierstrass functions, 7 times faster than IPOP-aCMA-ES and almost 3 times faster than BIPOP-aCMA-ES. Overall, according to Fig. 4, NIPOP-aCMA-ES performs as well as BIPOP-aCMA-ES, while restricted to only one regime of increasing population size.

NBIPOP-aCMA-ES. Thanks to the first regime of increasing population size, NBIPOP-aCMA-ES inherits some results of NIPOP-aCMA-ES. However, on functions where the population size does not play any important role, it performs significantly better than BIPOP-aCMA-ES. This is the case for f_{21} Gallagher 101 peaks and f_{22} Gallagher 21 peaks functions, where NBIPOP-aCMA-ES has a speedup of a factor of 6. It seems that the adaptive choice between two regimes works efficiently on all functions except on f_{16} Weierstrass. In this last case, NBIPOP-aCMA-ES mistakingly prefers small populations, with a loss factor 4 compared to NIPOP-aCMA-ES. According to Fig. 4, NBIPOP-aCMA-ES performs better than BIPOP-aCMA-ES on weakly structured multi-modal functions, showing overall best results for BBOB-2009 and BBOB-2010 workshops in dimensions 20 (results not shown here) and 40.

Due to space limitations, the interested reader is referred to [13] for a detailed presentation of the results.

4.2 Interplanetary Trajectory Optimization

The NIPOP-aCMA-ES and NBIPOP-aCMA-ES strategies, designed for the BBOB benchmark functions, can possibly *overfit* this benchmark suite. In order to test the generality of these strategies, a real-world black-box problem is considered, pertaining to a completely different domain: Advanced Concepts Team of European Space Agency is making available several difficult spacecraft trajectory optimization problems as black box functions to invite the operational research community to compare different derivative-free solvers on these test problems [16].

The following results consider the 18-dimensional bound-constrained black-box function "TandEM-Atlas501", that defines an interplanetary trajectory to Saturn from the Earth with multiple fly-bys, launched by the rocket Atlas 501. The final goal is to maximize the mass $f(x)$, which can be delivered to Saturn using one of 24 possible fly-by sequences with possible maneuvers around Venus, Mars and Jupiter.

The first best results was found for a sequence Earth-Venus-Earth-Earth-Saturn ($f_{max} = 1533.45$) in 2008 by B. Addis et al. [1]. The best results so far ($f_{max} = 1673.88$) was found in 2011 by G. Stracquadanio et al. [15].

All versions of CMA-ES with restarts have been launched with a maximum budget of 10^8 function evaluations. All variables are normalized in the range $[0, 1]$. In the case of sampling outside of boundaries, the fitness is penalized and becomes $f(x) = f(x_{feasible}) - \alpha \left\| x - x_{feasible} \right\|^2$, where $x_{feasible}$ is the closest feasible point from point x and α is a penalty factor, which was arbitrarily set to 1000.

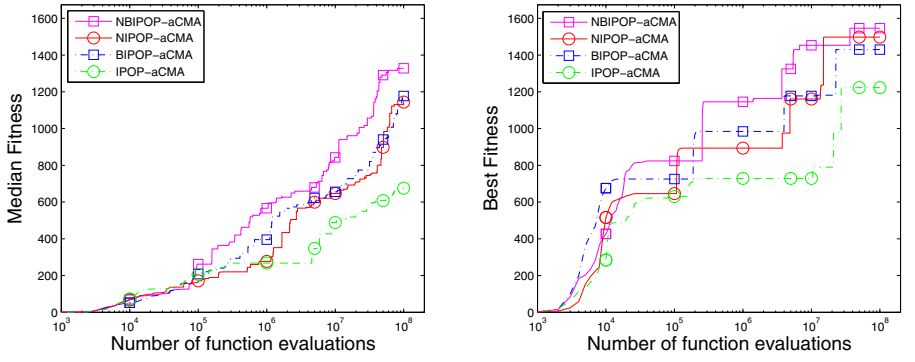


Fig. 3. Comparison of all CMA-ES restart strategies on the Tandem fitness function (mass): median (left) and best (right) values out of 30 runs

As shown on Fig. 3, the new restart strategies NIPOP-aCMA-ES and NBIPOP-aCMA-ES respectively improve on the former ones (IPOP-aCMA-ES and BIPOP-aCMA-ES); further, NIPOP-aCMA-ES reaches same performances as BIPOP-aCMA-ES.

The best solution found by NBIPOP-aCMA-ES[2] improves on the best solution found in 2008, while it is worse than the current best solution, which is blamed on the lack of problem specific heuristics [1,15], on the possibly insufficient time budget (10^8 fitness evaluations), and also on the lack of appropriate constraint handling heuristics.

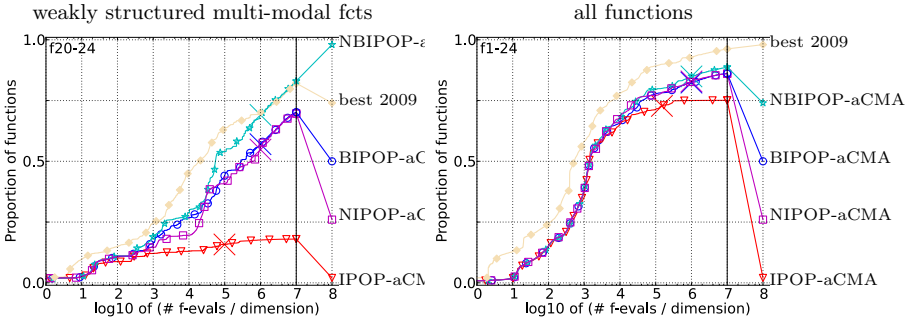


Fig. 4. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and weakly structured multi-modal subgroup in 40-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target.

[2] x =[0.83521, 0.45092, 0.50284, 0.65291, 0.61389, 0.75773, 0.43376, 1, 0.89512, 0.77264, 0.11229, 0.20774, 0.018255, 6.2057e-09, 4.0371e-08, 0.2028, 0.36272, 0.32442]; fitness(x) = mass(x) = 1546.5.

Table 1. Overall results on multi-modal functions $f3-4$ and $f15-24$ in dimension $d = 40$: Expected running time (ERT in number of function evaluations) divided by the respective best ERT measured during BBOB-2009 for precision Δf ranging in 10^i, $i = 1 \ldots -7$. The median number of conducted function evaluations is additionally given in *italics*, if $\mathrm{ERT}(10^{-7}) = \infty$. #succ is the number of trials that reached the final target $f_{\mathrm{opt}} + 10^{-8}$. Best results are printed in bold. For a more detailed (statistical) analysis of results on BBOB problems, please see [13]. Statistically significantly better entries (Wilcoxon rank-sum test with $p = 0.05$) are indicated in bold. The interested reader is referred to [13] for the statistical analysis and discussion of these results.

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f3	15526	15602	15612	15646	15651	15656	15/15
BIPOP-a	**2395**	∞	∞	∞	∞	∞ *4e7*	0/15
IPOP-aC	∞	∞	∞	∞	∞	∞ *6e6*	0/8
NBIPOP-	8177	∞	∞	∞	∞	∞ *4e7*	0/15
NIPOP-a	4615	∞	∞	∞	∞	∞ *4e7*	0/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f4	15536	15601	15659	15703	15733	2.8e5	6/15
BIPOP-a	∞	∞	∞	∞	∞	∞ *4e7*	0/15
IPOP-aC	∞	∞	∞	∞	∞	∞ *6e6*	0/8
NBIPOP-	∞	∞	∞	∞	∞	∞ *4e7*	0/15
NIPOP-a	∞	∞	∞	∞	∞	∞ *4e7*	0/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f15	1.9e5	7.9e5	1.0e6	1.1e6	1.1e6	1.1e6	15/15
BIPOP-a	1.2	1.1	1.1	1.1	1.1	1.1	15/15
IPOP-aC	**0.72**	**0.43**	0.60	0.61	0.62	0.63	8/8
NBIPOP-	1.0	0.71	0.75	0.76	0.77	0.77	15/15
NIPOP-a	0.92	0.61	**0.55**	**0.56**	**0.57**	**0.58**	15/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f16	5244	72122	3.2e5	1.4e6	2.0e6	2.0e6	15/15
BIPOP-a	1.3	0.96	0.80	0.54	0.50	0.51	15/15
IPOP-aC	**0.91**	1.1	1.0	0.51	1.4	1.4	8/8
NBIPOP-	0.97	0.78	0.34	0.38	0.46	0.74	15/15
NIPOP-a	1.2	**0.65**	**0.23**	**0.21**	**0.16**	**0.18**	15/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f17	399	4220	14158	51958	1.3e5	2.7e5	14/15
BIPOP-a	1.1	0.64	1.6	1.1	1.4	0.87	15/15
IPOP-aC	1.0	0.52	1.3	1.3	**0.97**	0.83	8/8
NBIPOP-	1.0	0.57	1.2	1.2	1.0	0.81	15/15
NIPOP-a	**0.97**	**0.52**	**0.97**	**1.00**	1.1	**0.70**	15/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f18	1442	16998	47068	1.9e5	6.7e5	9.5e5	15/15
BIPOP-a	**0.94**	**0.51**	1.0	0.98	0.88	0.67	15/15
IPOP-aC	0.96	0.68	1.0	**0.66**	**0.45**	0.48	8/8
NBIPOP-	1.0	0.97	1.1	0.93	0.57	0.53	15/15
NIPOP-a	0.95	0.58	**0.75**	0.71	0.50	**0.42**	15/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f19	1	1	1.4e6	2.6e7	4.5e7	4.5e7	8/15
BIPOP-a	**396**	6.7e4	0.87	1.2	1.0	1.0	9/15
IPOP-aC	462	**4.4e4**	**0.57**	**0.34**	**0.20**	**0.20**	8/8
NBIPOP-	424	8.3e4	0.97	0.81	1.1	1.1	9/15
NIPOP-a	436	8.2e4	1.9	0.48	0.32	0.32	15/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f20	222	1.3e5	1.6e8	∞	∞	∞	0
BIPOP-a	4.0	9.0	0.34	.	.	.	0/15
IPOP-aC	**3.9**	8.1	**0.18**	.	.	.	0/8
NBIPOP-	4.0	8.5	0.39	.	.	.	0/15
NIPOP-a	4.0	**6.5**	0.32	.	.	.	0/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f21	1044	21144	1.0e5	1.0e5	1.0e5	1.0e5	26/30
BIPOP-a	7.5	60	37	37	37	37	15/15
IPOP-aC	7.1	421	∞	∞	∞	∞ *3e6*	0/8
NBIPOP-	**4.9**	**10**	**5.1**	**5.1**	**5.1**	**5.1**	15/15
NIPOP-a	14	440	173	172	171	171	12/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f22	3090	35442	6.5e5	6.5e5	6.5e5	6.5e5	8/30
BIPOP-a	**12**	343	201	200	200	199	4/15
IPOP-aC	144	**93**	∞	∞	∞	∞ *3e6*	0/8
NBIPOP-	12	112	**32**	**32**	**32**	**32**	12/15
NIPOP-a	179	583	∞	∞	∞	∞ *4e7*	0/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f23	7.1	11925	75453	1.3e6	3.2e6	3.4e6	15/15
BIPOP-a	8.4	**7.8**	**1.3**	1.9	1.00	0.99	15/15
IPOP-aC	9.2	∞	∞	∞	∞	∞ *4e6*	0/8
NBIPOP-	8.6	10	1.6	1.3	0.58	0.59	15/15
NIPOP-a	**5.9**	61	11	**0.72**	**0.36**	**0.38**	15/15

Δf_{opt}	1e1	1e0	1e-1	1e-3	1e-5	1e-7	#succ
f24	5.8e6	9.8e7	3.0e8	3.0e8	3.0e8	3.0e8	1/15
BIPOP-a	3.6	1.4	∞	∞	∞	∞ *4e7*	0/15
IPOP-aC	∞	∞	∞	∞	∞	∞ *1e7*	0/8
NBIPOP-	2.1	0.19	0.97	0.97	0.97	0.97	2/15
NIPOP-a	**1.2**	**0.15**	**0.44**	**0.44**	**0.44**	**0.44**	4/15

5 Conclusion and Perspectives

This paper contribution regards two new restart strategies for CMA-ES. NIPOP-aCMA-ES is a deterministic strategy simultaneously increasing the population size *and* decreasing the initial step-size of the Gaussian mutation. NBIPOP-aCMA-ES implements a competition between NIPOP-aCMA-ES and a random sampling of the initial mutation step-size, adaptively adjusting the computational budget of each one depending on their current best results. Besides the extensive validation of NIPOP-aCMA-ES and NBIPOP-aCMA-ES on the BBOB benchmark, the generality of these strategies has been tested on a new problem, related to interplanetary spacecraft trajectory planning.

The main limitation of the proposed restart strategies is to quasi implement a deterministic trajectory in the θ space. Further work will consider $h(\theta)$ as yet another expensive noisy black-box function, and the use of a CMA-ES in the hyper-parameter space will be studied. The critical issue is naturally to keep

the overall number of fitness evaluations beyond reasonable limits. A surrogate-based approach will be investigated [14], learning and exploiting an estimate of the (noisy and stochastic) $h(\theta)$ function.

References

1. Addis, B., Cassioli, A., Locatelli, M., Schoen, F.: Global optimization for the design of space trajectories. Optimization On Line, 11 (2008)
2. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005), pp. 1769–1776. IEEE Press (2005)
3. García, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC 2005 special session on real parameter optimization. Journal of Heuristics 15, 617–644 (2009)
4. Hansen, N.: Compilation of results on the 2005 CEC benchmark function set (May 2006)
5. Hansen, N.: Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In: Rothlauf, F. (ed.) GECCO Companion, pp. 2389–2396. ACM (2009)
6. Hansen, N.: References to CMA-ES applications (2009), http://www.lri.fr/~hansen/cmaapplications.pdf
7. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2012: Experimental setup. Technical report, INRIA (2012)
8. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA (2009) (updated, February 2010)
9. Hansen, N., Kern, S.: Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 282–291. Springer, Heidelberg (2004)
10. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
11. Hansen, N., Ros, R.: Benchmarking a weighted negative covariance matrix update on the BBOB-2010 noiseless testbed. In: GECCO 2010: Proceedings of the 12th Annual Conference Comp. on Genetic and Evolutionary Computation, pp. 1673–1680. ACM, New York (2010)
12. Jastrebski, G.A., Arnold, D.V.: Improving evolution strategies through active covariance matrix adaptation. In: IEEE Congress on Evolutionary Computation, CEC 2006, pp. 2814–2821 (2006)
13. Loshchilov, I., Schoenauer, M., Sebag, M.: Black-box Optimization Benchmarking of NIPOP-aCMA-ES and NBIPOP-aCMA-ES on the BBOB-2012 Noiseless Testbed. In: GECCO 2012: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. ACM (page to appear, 2012)
14. Loshchilov, I., Schoenauer, M., Sebag, M.: Self-Adaptive Surrogate-Assisted Covariance Matrix Adaptation Evolution Strategy. In: GECCO 2012 Proceedings. ACM (page to appear, 2012)
15. Stracquadanio, G., La Ferla, A., De Felice, M., Nicosia, G.: Design of robust space trajectories. In: Research and Development in Intelligent Systems XXVIII, pp. 341–354. Springer (2011)
16. Vinko, T., Izzo, D.: Global Optimisation Heuristics and Test Problems for Preliminary Spacecraft Trajectory Design. Technical Report GOHTPPSTD, European Space Agency (2008)

Are State-of-the-Art Fine-Tuning Algorithms Able to Detect a Dummy Parameter?*

Elizabeth Montero[1], María-Cristina Riff[1], Leslie Pérez-Caceres[2], and Carlos A. Coello Coello[3]

[1] Departamento de Informática
Universidad Técnica Federico Santa María
Valparaíso, Chile
{elizabeth.montero,maria-cristina.riff}@inf.utfsm.cl
[2] Université Libre de Bruxelles
Bruxelles, Belgium
leslie.perez.caceres@ulb.ac.be
[3] CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07360, Mexico
ccoello@cs.cinvestav.mx

Abstract. Currently, there exist several offline calibration techniques that can be used to fine-tune the parameters of a metaheuristic. Such techniques require, however, to perform a considerable number of independent runs of the metaheuristic in order to obtain meaningful information. Here, we are interested on the use of this information for assisting the algorithm designer to discard components of a metaheuristic (e.g., an evolutionary operator) that do not contribute to improving its performance (we call them "ineffective components"). In our study, we experimentally analyze the information obtained from three offline calibration techniques: F-Race, ParamILS and Revac. Our preliminary results indicate that these three calibration techniques provide different types of information, which makes it necessary to conduct a more in-depth analysis of the data obtained, in order to detect the ineffective components that are of our interest.

Keywords: fine-tuning methods, algorithm design process, ineffective operators.

1 Introduction

We are currently involved in a project whose goal is to propose strategies to assist the decision-making process of designers of metaheuristic algorithms. As designers decide to add new components (e.g., a new evolutionary operator) to a certain metaheuristic, the fine-tuning process gets more complex. This is due to the

* Partially supported by Fondecyt Project no. 1120781, CONACYT/CONICYT Project no. 2010-199 and CONACyT Project no. 103570.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 306–315, 2012.

highly nonlinear interactions that normally occur among the different parameters of a metaheuristic. Here, we are precisely interested in devising strategies that can help us to detect the components of a metaheuristic that are really crucial for its performance, so that the fine-tuning process can get reduced to a minimum. It is worth noting that other decisions related to the parameters of a metaheuristic algorithm are made during the design process. These decisions are not only concerned with finding the best possible parameter values, but also with deciding which parameters must be empirically tuned, and which ones can take either a fixed value or a value that can be varied or adapted online during the search process [8,6,4,7]. Over the years, there have been several efforts to develop automated fine-tuning methods (see for example [12,3,1,5]). Such methods require thousands of runs (and therefore, large amounts of time) in order to obtain good quality parameter configurations for a metaheuristic. The information that is extracted from this exhaustive process could be, however, very useful for improving the design of the metaheuristic itself. Since that is one of the main goals of this work, we conduct here an in-depth analysis of the information obtained with three well-known fine-tuning methods (ParamILS, F-Race and Revac), aiming to detect ineffective components in the algorithms being fine-tuned. In order to evaluate the output information obtained from the fine-tuning methods being analyzed, we adopted *Ant Solver*[1] [11], which is a well-known ant colony optimization algorithm that has been a popular choice for solving constraint satisfaction problems. It is important to emphasize that our goal here is not to find the best possible solutions to the problems being analyzed, but to detect ineffective components of the algorithm being analyzed, based on the information obtained from its systematic fine-tuning process. For this sake, we include a dummy operator in the code of the Ant Solver. Evidently, such a dummy operator is meant to be ineffective, because it doesn't perform any meaningful task within the algorithm. Its only purpose is to validate our methodology to detect ineffective components of a metaheuristic. The remainder of this paper is organized as follows. The next section provides a short description of the fine-tuning methods adopted for our analysis. Section 3 describes the Ant Solver algorithm adopted for our case study. Section 3.1 discusses the incorporation of a dummy operator into the Ant Solver algorithm. The instances used for our analysis are briefly explained in Section 4. In Section 5, we describe the experiments performed and the results obtained. Finally, Section 6 provides our main conclusions and some possible paths for future research.

2 Fine-Tuning Techniques

The fine-tuning techniques described next are strategies designed to automatically search for the best configuration of parameter values for a stochastic based method. Given a heuristic algorithm with k parameters, a fine-tuning technique searches for the parameter configuration $\theta^* = \{p_1, \ldots, p_k\}$ that provides the best performance of the algorithm. When talking about parameters, we refer to two main sets of elements:

[1] The authors thank Cristine Solnon for kindly providing us the source code of the *Ant Solver*.

- **Categorical parameters:** These are processes or functions that are required in an algorithm but that can be implemented in different ways. For example, the selection mechanism of an evolutionary algorithm.
- **Numerical parameters:** These are parameters expressed with real numbers or integers. For example the population size for an evolutionary algorithm.

The main difference between categorical and numerical parameters is that the latter are searchable (i.e., it is possible to define a distance measure between two different values of the parameter). In contrast, in categorical parameters it is not possible to define the distance between two "values".

2.1 F-Race

The F-Race method was proposed by Birattari et al. [2]. F-Race is a specific racing method specially adapted to fine-tune stochastic search methods. It uses Friedman two-ways analysis of variance by ranks to compare sets of candidate parameter configurations. This is a non-parametric test based on ranking, thus it does not require the formulation of a hypothesis on the distribution of the observations. Moreover, ranking based tests are very useful in fine-tuning problems because they implement a block design, which considers the different problem instances and the random seeds as sources of variation. The performance difference between configurations is analyzed using a hypothesis test. F-Race stops either when there is only one parameter configuration remaining, or when some predefined number of runs has been completed. The F-Race method defines three parameters: the initial number of runs without elimination of calibrations, the confidence level for the tests and the maximum budget. It also requires the range levels for each parameter. The number of levels of all parameters determines the size of the initial set of candidate parameter configurations.

2.2 Revac

The Relevance Estimation and Value Calibration (Revac) of evolutionary algorithms method was proposed by Eiben & Nannen [9]. Revac can be seen as an estimation of distribution algorithm [10]. It works with a set of parameter configurations as its population. For each parameter, it starts the search process with a uniform distribution of values within a given range. As the process advances, Revac performs transformation operations (crossover and mutation) with the aim of reducing each parameter distribution to a range of values that provide the best performance. Revac stops after performing 1000 runs of the fine-tuned algorithm. This approach defines 4 parameters: population size, step size of the crossover operator, step size of the mutation operator and the maximum number of iterations.

2.3 ParamILS

The Parameter Iterated Local Search (ParamILS) strategy was proposed in [5]. It works as an iterated local search algorithm which starts with a default parameter configuration and iteratively improves the configuration performance

searching in the neighborhood of the configuration at hand. At each iteration, it performs random perturbations to the configuration at hand, and then applies the local search process and compares the outcome to the performance of the best parameter configuration that has been found so far. There are two well known versions of this approach: BasicILS and FocusedILS. These versions differ in the comparison procedure of parameter configurations they use. The ParamILS method defines four parameters: the amount of random solutions of the first phase, the amount of random solutions of each iteration, a restart probability and the maximum budget.

2.4 Comparison of Fine-Tuning Methods

All the techniques considered here need the definition of an interval of values for each parameter to be fine-tuned. Furthermore, F-Race and ParamILS require the definition of a set of countable values for each parameter (S_i). The number of configurations evaluated by F-Race grows exponentially on the size of each S_i. F-Race evaluates all of these configurations at least r times, whereas ParamILS reduces the number of evaluated configurations by exploring the most promising parameter configurations. Revac and ParamILS are stochastic search methods, then they are sensitive to the random seed adopted for the search process. F-Race is not a stochastic method but it defines a set of random seeds to execute the algorithm to be fine-tuned and these seeds could have an impact on the fine-tuning process. ParamILS provides as its output the best performing configuration, while Revac determines, for each parameter, an interval of values, and F-Race reports the set of the best performing configurations obtained. Revac is not able to search for categorical parameters, because its transformation process is performed on a continuous search space. However, both ParamILS and F-Race are able to tackle both categorical and continuous spaces. Table 1 summarizes the main features of tuning techniques.

Table 1. Features of the fine-tuning methods adopted

Method	**F-Race**	**Revac**	**ParamILS**
Type	Experimental Design	Search Based	Search Based
Initial input	Set of values	Interval/precision	Set of values
Expected output	Set of best configurations	An interval of values for each parameter	Best configuration
# parameters	3	4	4
Scope	Categorical/Numerical	Numerical	Categorical/Numerical
Stop criterion	Max runs or one configuration left	Max iterations	Max runs/time

3 Ant Solver

For our experiments we used an Ant Colony Optimization (ACO) based approach called *Ant Solver* [11]. This algorithm was proposed to solve constraint

satisfaction problems (CSP). Ant Solver searches for a solution that minimizes the number of violated constraints. At each step of Ant Solver, each ant constructs a complete assignment for the CSP, and the pheromone trails are updated at the end of each cycle as usually done in ACO algorithms. The pheromone is laid on a binary graph whose vertices (X_i, v) represent the assignment of value v to variable X_i and the edges between two vertices represent those simultaneous assignment of values. Ant solver includes *pre-* and *post-* processing features that use a min-conflicts based local search procedure. The pre-processing phase performs local search repeatedly to collect information that is used to initialize pheromone trails and the post-processing phase performs local search after each ant has constructed a complete assignment. In our experiments, the pre- and post-processing procedures were disabled in order to analyze the behavior of the ACO algorithm alone. Ant Solver has four parameters: α, β, ρ and $nAnts$. α and β determine, respectively, the weight of the pheromone and the weight of heuristics in the computation of transition probabilities, ρ represents the level of pheromone evaporation and $nAnts$ corresponds to the number of ants used. We added an operator (which is described next) and, consequently, a new parameter for testing the fine-tuning techniques previously indicated.

3.1 Dummy Operator

Since our hypothesis was that a fine-tuning technique can provide information about ineffective components of an algorithm, we decided to add a dummy operator to the Ant Solver in order to validate it. This dummy operator takes an assignment and returns it without making any further changes. Its execution is controlled by a parameter δ that indicates its execution probability. In a real scenario, the operators that do not help in the search process use resources and spend time. Therefore, in order to simulate this behavior, the dummy operator is set to consume 1% of the constraint checks allowed in the execution in order to represent these costs.

4 Instances

The CSP instances used in this paper correspond to 3-coloring problems. Such instances were generated using a simple heuristic that generates instances that have at least one solution. The construction heuristic works as follows: first, the problem variables are separated in three disjoint sets, and then the variables are iteratively connected exclusively with variables in different sets, forbidding the intra set connections. This process continues until a given average connection level is met and the problem instance is obtained. The instances generated for these experiments have 400 and 500 variables and a connection average of 3.

5 Experiments

For our experiments we used a public domain implementation of ParamILS available at: `www.cs.ubc.ca/labs/beta/Projects/ParamILS`. We also adopted the authors' implementation of F-Race and our own implementation of Revac. The source code of F-Race, Revac and the Ant Solver are available at:

www.inf.utfsm.cl/~emontero. For all our experiments we tuned the probability of application of the dummy operator. Each tuning process was executed 5 times in order to obtain representative results. The rest of the parameter were set as follows: $nAnts = 15$, $\alpha = 2.00$, $\beta = 10.00$, and $\rho = 0.01$ according to recommendations of author in [11].

The hardware platform adopted for the experiments was a PC with an Intel Corei7-920, having 4GB of RAM, and using the Linux Mandriva 2010 operating system. Two sets of experiments were conducted:

- An analysis of the information obtained by the three fine-tuners when solving problems with 400 variables from the *Test Suite 1*.
- An analysis of the information obtained by the three fine-tuners when solving problems with 500 variables from the *Test Suite 2*.

Table 2. Performance measure for instances from the Test Suite 1

p_dummy	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
fitness	0	0	11.3	18.9	21.8	25.6	26.6	27.7	32.4	32.7	32.7
conflict checks E+008	0.34	3.17	5.40	5.76	5.78	5.78	5.79	5.80	5.80	5.82	5.78

Fine-tuning methods use the number of violated constraints of the best solution found as the evaluation criteria to assess the performance of the Ant Solver.

5.1 Ineffective Operator

The objective of this experiment is to assess if the fine-tuning techniques adopted are able to provide information that allows us to infer that the algorithm design includes an ineffective operator. The expected result is that the probability assigned to the operator is zero. As indicated before, we included a dummy operator for this experiment. This operator receives a candidate solution and does not perform any change to it.

5.2 Performance Analysis

Here, we present a set of experiments which aim to understand the noticeable effect that can have an ineffective operator in the algorithm. For this purpose, we measure the quality of the solutions found and the number of conflict checks performed for different values of the dummy operator rate. Table 2 shows these values for instances of 400 variables and Table 3 shows them for instances of 500 variables. We can observe in Table 2 that the average performance of Ant Solver increases as the dummy rate decreases. It is important to note that the performance for rates 0.0 and 0.1 is the same. This is because in both cases, Ant Solver has a sufficient budget of evaluations to search until the best solution is found. However, the number of conflict checks performed by the second case is almost 10 times the number of checks performed by the first one. The higher the values of the dummy probability, the larger becomes the number of resources

consumed by the dummy operator and, consequently, the problem instances can no longer be solved. As the value of the dummy operator gets larger, the worse is the performance of the Ant Solver. In Table 3, we can observe the performance of the Ant Solver when dealing with instances of 500 variables. In this case, the algorithm clearly shows that using the dummy operator strongly increases the number of constraints checks. The higher the dummy probability, the worse becomes the performance of the Ant Solver.

Table 3. Performance measure for instances in Test Suite 2

p_dummy	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
fitness	0	2.5	31.9	46.1	49.5	50.6	52.2	55.2	58.8	61.4	62.6
conflict checks E+008	0.59	6.53	8.98	9.04	9.03	9.04	9.05	9.06	9.07	9.07	9.14

5.3 Test Suite 1

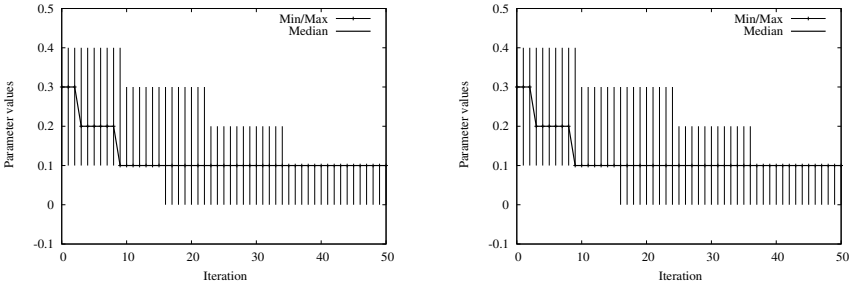
Here, we detail the results obtained using the Test Suite 1 composed by 10 instances of 400 variables as described in Section 4.

Analysis of Results for F-Race. The F-Race algorithm started the fine-tuning process with a set of 11 parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$. After finishing the first phase of 5 runs without elimination of configurations, F-Race discarded 9 configurations and kept only two of them $S' = \{0.0, 0.1\}$. Then, it ended with the set S' containing both configurations. This means that both parameter configurations are equivalent (i.e., the incorporation of the dummy operator at very low rates (lower or equal than 0.1) does not affect the performance of Ant Solver). F-Race required 550 Ant Solver runs in order to detect the ineffective operator.

Analysis of Results for Revac. Revac started the fine-tuning process with an initial interval of values in the range $[0.0, 1.0]$. The convergence process performed by Revac to fine-tune the dummy parameter is shown in Figure 1(a). This plot shows the median, the minimum and the maximum values of the ranges of values for the parameter at each iteration. Here, we can see that at the first iteration, the range of parameter values has already been reduced to $[0.1, 0.4]$, but it is still required to perform more iterations to refine this range of values. As shown in Figure 1(a), Revac required 35 iterations (around 1350 runs of Ant Solver) to converge to the range of values that performs the best for the dummy probability $[0.0, 0.1]$.

Analysis of Results for ParamILS. For ParamILS, we also considered 11 parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$ and the initial configuration was set to 0.5. After 100 runs of Ant Solver, ParamILS was able to change the value from 0.5 to 0.1. ParamILS ended with a value of 0.1, because the performance of Ant Solver with a probability of 0.1 for the dummy operator is equivalent to the performance obtained with a probability of 0.0.

Discussion. It is important to remark that the three fine-tuning methods adopted here were able to minimize the effect of an ineffective operator included



(a) Test Suite 1 (b) Test Suite 2

Fig. 1. Convergence of Revac for Test Suite 1 and 2

in the algorithm. ParamILS was clearly the most efficient, because both Revac and F-Race require the execution of an initialization phase. In this particular set of instances, we observed in Section 5.2 that the performance of Ant Solver is equivalent when using a probability of either 0.0 or 0.1 for the dummy operator. This is because the algorithm is able to solve the problem even with a dummy probability of 0.1. Considering this specific problem, we suggest that when the algorithm designer doubts about the effectiveness of a component, he can use ParamILS giving zero as the first value for the parameter that controls such a component. ParamILS will then be able to change this value to another one if it can obtain a significantly better result. In our tests, we began by assigning 0.5 to this parameter value, and then, when ParamILS decreases this value and it finds out that using 0.1 produces good results, it stops searching.

5.4 Test Suite 2

Here, we summarize the results obtained during the fine-tuning processes of the instances from the Test Suite 2. This test suite is composed by 10 problem instances of 3-coloring problem of 500 variables each.

Analysis of Results for F-Race. F-Race started the fine-tuning process with a set of 11 parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$. After finishing the first phase of 5 runs without elimination of configurations, F-Race discarded 9 configurations and kept only two of them $S' = \{0.0, 0.1\}$. At the next race, these two configurations were compared and the dummy value 0.0 showed a better performance than the configuration 0.1. In this case, the process ended after 570 Ant Solver runs.

Analysis of Results for Revac. Revac started the fine-tuning process with an interval of values in the range $[0.0, 1.0]$. The convergence process of Revac for fine-tuning the dummy parameter for the instances in Test Suite 2 is shown in Figure 1(b). Here, we can see that at the first iteration the range of parameter values has been reduced to the same range as that for Test Suite 1. The entire convergence process is very similar to the process corresponding to the Test Suite 1. This is because both processes were performed considering the same random

seed. There are, however, some small differences. For example, in this case, the final range reduction took place at iteration 37. This means that Revac required around 1370 runs of Ant Solver to converge to the range of values $[0.0, 0.1]$.

Analysis of Results for ParamILS. In this case, we also considered 11 possible parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$ and the initial value for the dummy rate was set to 0.5. The parameter changed to the value 0.1 after 100 Ant Solver runs, and it changed to 0.0 after 590 Ant Solver runs. The parameter value did not change during the rest of the fine-tuning process.

Discussion. In this case, the three fine-tuning methods were able to detect the ineffective operator included in the algorithm. ParamILS and F-Race were both the most efficient. The initialization phase of Revac was again detrimental in its competitiveness, but not for F-Race. This set of instances constitutes a more typical example of the scenarios that fine-tuning methods could face when searching for ineffective operators. In this case, the performance of the algorithm can be considered as the only indicator of the quality of the search that the algorithm is performing.

5.5 Final Remarks

Considering the two fine-tuning scenarios analyzed here, it is important to notice that the first one is more complex, since in that case the performance of the algorithm is not enough to categorically determine the elimination of the dummy operator. For the first scenario, a multistage procedure could be performed in order to analyze different quality measures of the search process in order to debug the design of the algorithm. Some good practices could also be considered for the application of fine-tuning methods to identify ineffective operators. For example, let's consider as our initial solution a configuration that could discard an operator (operator rate set to 0.0) in ParamILS. Only if such operator shows to be useful for the algorithm, either isolated or combined with other operators, ParamILS will incorporate it. For Revac and F-Race, their initial phases could be oriented to better analyze configurations discarding the use of the operator analyzed. Only when the fine-tuning method decided that these operators are useful for the algorithm, their operator rates would be fine-tuned.

6 Conclusions and Future Work

In this paper we have proposed the use of the information obtained by fine-tuning techniques for assisting the design process of metaheuristics. We have shown the way in which this information can be used to identify ineffective components (an operator, in this case).

Our experiments indicated that ParamILS was the best technique at identifying this situation in a more efficient way. Revac and F-race required more resources because of their expensive initialization phases. In our experiments the three fine-tuners were allowed to execute a fixed maximum of executions of the Ant Solver. The three fine-tuners studied here can be stopped at any time before reaching the maximum number of evaluations. However, in our experiments

we waited until completing all the executions, so that a fair comparison of the tuning approaches could be done.

As part of our future work, we would like to study more recent fine-tuners such as sequential parameter optimization and the irace methods. Moreover, we aim to develop mechanisms that allow the collaboration of different fine-tuning techniques as a way of assisting the design of heuristic algorithms. In this case, however, the aim would be to detect effective components, instead of ineffective ones. We would also like to study other interesting aspects related to the algorithm design process, such as the identification of more than one ineffective operator, and the identification of opposite behavior of operators in the presence of noise and also considering continuous fitness landscapes.

References

1. Bartz-Beielstein, T., Lasarczyk, C.W.G., Preuss, M.: Sequential Parameter Optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, pp. 773–780 (2005)
2. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 11–18. Morgan Kaufmann, New York (2002)
3. Eiben, A.E., Schut, M.C.: New Ways to Calibrate Evolutionary Algorithms. In: Advances in Metaheuristics for Hard Optimization, pp. 153–177. Springer, Heidelberg (2008)
4. Fialho, Á., Da Costa, L., Schoenauer, M., Sebag, M.: Analyzing bandit-based adaptive operator selection mechanisms. Annals of Mathematics and Artificial Intelligence – Special Issue on Learning and Intelligent Optimization (2010)
5. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proceedings of the Twenty-Second Conference on Artifical Intelligence, pp. 1152–1157 (2007)
6. Maturana, J., Lardeux, F., Saubion, F.: Autonomous operator management for evolutionary algorithms. Journal of Heuristics 16, 881–909 (2010)
7. Montero, E., Riff, M.C., Neveu, B.: C-strategy: A Dynamic Adaptive Strategy for the CLONALG Algorithm. Transactions on Computational Sciences, Special Issue 8, 41–55 (2010)
8. Montero, E., Riff, M.C.: On-the-fly calibrating strategies for evolutionary algorithms. Information Sciences 181(3), 552–566 (2011)
9. Nannen, V., Eiben, A.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Joint International Conference for Artificial Intelligence (IJCAI), pp. 975–980 (2007)
10. Pelikan, M., Goldberg, D.E., Lobo, F.G.: A Survey of Optimization by Building and Using Probabilistic Models. Computational Optimization and Applications 21(1), 5–20 (2002)
11. Solnon, C.: Ants can solve constraint satisfaction problems. IEEE Transactions on Evolutionary Computation 6(4), 347–357 (2002)
12. Yuan, Z., de Oca, M.A.M., Stützle, T., Birattari, M.: Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. IRIDIA - Technical Report Serie TR/IRIDIA/2011-017, Université Libre de Bruxelles (Agosto 2011)

Compressed Network Complexity Search

Faustino Gomez, Jan Koutník, and Jürgen Schmidhuber

IDSIA
USI-SUPSI
Manno-Lugano, CH
{tino,hkou,juergen}@idsia.ch

Abstract. Indirect encoding schemes for neural network phenotypes can represent large networks compactly. In previous work, we presented a new approach where networks are encoded indirectly as a set of Fourier-type coefficients that decorrelate weight matrices such that they can often be represented by a small number of genes, effectively reducing the search space dimensionality, and speed up search. Up to now, the complexity of networks using this encoding was fixed *a priori*, both in terms of (1) the number of free parameters (topology) and (2) the number of coefficients. In this paper, we introduce a method, called Compressed Network Complexity Search (CNCS), for automatically determining network complexity that favors parsimonious solutions. CNCS maintains a probability distribution over complexity classes that it uses to select which class to optimize. Class probabilities are adapted based on their expected fitness. Starting with a prior biased toward the simplest networks, the distribution grows gradually until a solution is found. Experiments on two benchmark control problems, including a challenging non-linear version of the helicopter hovering task, demonstrate that the method consistently finds simple solutions.

1 Introduction

Indirect or generative encoding schemes for neural network phenotypes [2–4,9,11] offer the potential of allowing very large networks to be represented compactly. In previous work [5,6], we presented a new encoding where network weight matrices are represented indirectly as a set of Fourier-type coefficients that are transformed into weight values via an inverse Fourier transform, so that evolutionary search is conducted in the frequency-domain instead of weight space. If adjacent weights in the matrices are correlated, then this regularity can be encoded using fewer coefficients than weights, effectively reducing the search space dimensionality. For problems exhibiting a high-degree of redundancy, this "compressed" approach can result in an order of magnitude fewer free parameters and significant speedup [5].

Up to now the complexity of networks using this encoding was fixed *a priori*, both in terms of (1) the number of free parameters or topology and (2) the number of coefficients (compression ratio). In this paper, we introduce a

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 316–326, 2012.

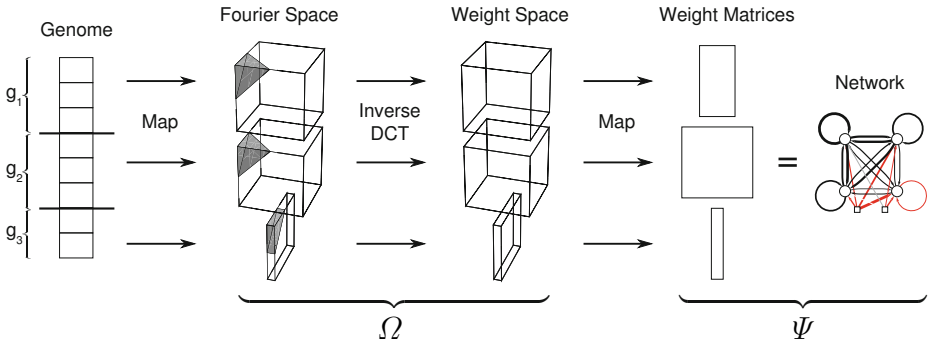


Fig. 1. Decoding the compressed networks. The figure shows the three step process involved in transforming a genome of frequency-domain coefficients into a recurrent neural network. First, the genome (left) is divided into k chromosomes, one for each of the weight matrices specified by the network architecture, Ψ. Each chromosome is mapped, by Algorithm 1, into a coefficient array of a dimensionality specified by Ω. In this example, an RNN with two inputs and four neurons is encoded as 8 coefficients. There are $k = |\Omega| = 3$, chromosomes and $\Omega = \{3, 3, 2\}$. The second step is to apply the inverse DCT to each array to generate the weight values, which are mapped into the weight matrices in the last step.

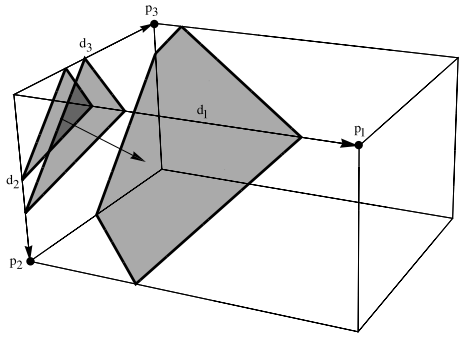
method inspired by universal search [7], called Compressed Network Complexity Search (CNCS), that automatically determines network complexity, favoring parsimonious solutions. CNCS maintains a probability distribution over complexity classes, which it uses to select which class to optimize. The probability of a given class is adapted based on the expected fitness of individuals sampled from it. Starting with a prior biased toward the simplest networks, the distribution adapts gradually until a solution is found.

The idea of enforcing parsimony in neuroevolution has been explored previously [15,16], usually by adding a regularization term to the fitness function that penalizes complexity. Our approach is more in line with NEAT [10] where simple networks are favored by starting evolution with a population of *directly* encoded networks that have minimal topologies.

The next section describes how networks are encoded in the frequency domain. Section 3, introduces the complexity search method, CNCS. Section 4, presents experiments applying CNCS to the octopus arm task with high-dimen-sional actions to determine the number of coefficient genes used to represent networks; and in section 5 it is used to search for both the number of neurons (topology) and coefficients, for networks controlling a challenging version of the Helicopter Hovering benchmark.

2 DCT Network Representation

Networks are encoded as a string or *genome*, $\boldsymbol{g} = \{g_1, \ldots, g_k\}$, consisting of k substrings or *chromosomes* of real numbers representing Discrete Cosine Transform



Algorithm 1: Coefficient mapping(g, d)

$j \leftarrow 0$
$K \leftarrow \text{sort}(\text{diag}(d) - \mathbb{I})$
for $i = 0$ **to** $|d| - 1 + \sum_{n=1}^{|d|} d_n$ **do**
  $l \leftarrow 0$
  $s_i \leftarrow \{e | \sum_{k=1}^{|d|} e_{\xi_j} = i\}$
  while $|s_i| > 0$ **do**
    $ind[j] \leftarrow \underset{e \in s_i}{\text{argmin}} \left\| e - K[l\text{++} \bmod |d|] \right\|$
    $s_i \leftarrow s_i \setminus ind[j\text{++}]$
for $i = 0$ **to** $|ind|$ **do**
  if $i < |g|$ **then**
    coeff_array$[ind[i]] \leftarrow c_i$
  else
    coeff_array$[ind[i]] \leftarrow 0$

Fig. 2. Mapping the coefficients. The cuboidal array is filled with the coefficients from chromosome g one simplex at a time, according to Algorithm 1, starting at the origin and moving to the opposite corner one simplex at a time.

(DCT) coefficients. The number of chromosomes is determined by the choice of network architecture, Ψ, and data structures used to decode the genome, specified by $\Omega = \{D_1, \ldots, D_k\}$, where D_m, $m = 1..k$, is the dimensionality of the coefficient array for chromosome m. The total number of coefficients, $C = \sum_{m=1}^{k} |g_m| \ll N$ (where N is the number of weights), is user-specified (for a compression ratio of N/C), and the coefficients are distributed evenly over the chromosomes. Which frequencies should be included in the encoding is unknown. The approach taken here restricts the search space to *band-limited* neural networks where the power spectrum of the weight matrices goes to zero above a specified limit frequency, c_ℓ^m, and chromosomes contain all frequencies up to c_ℓ^m, $g_m = (c_0^m, \ldots, c_\ell^m)$.

Figure 1 illustrates the procedure used to decode the genomes. In this example, a fully-recurrent neural network (on the right) is represented by $k = 3$ weight matrices, one for the input layer weights, one for the recurrent weights, and one for the bias weights. The weights in each matrix are generated from a different chromosome which is mapped into its own D_m-dimensional array with the same number of elements as its corresponding weight matrix; in the case shown, $\Omega = \{3, 3, 2\}$: 3D arrays for both the input and recurrent matrices, and a 2D array for the bias weights.

In previous work [5], the coefficient matrices were 2D, where the simplexes are just the secondary diagonals; starting in the top-left corner, each diagonal is filled alternately starting from its corners. However, if the task exhibits inherent structure that cannot be captured by low frequencies in a 2D layout, more compression can potentially be gained by organizing the coefficients in higher-dimensional arrays.

Each chromosome is mapped to its coefficient array according to Algorithm 1 which takes a list of array dimension sizes, $d = (d_1, \ldots, d_{D_m})$ and the chromosome, g_m, to create a total ordering on the array elements, $e_{\xi_1,\ldots,\xi_{D_m}}$. In the first

Algorithm 2. CNCS($\mathcal{D}$,f,s,n,σ_θ)

while $\neg$*converged* **do**
  for $k = 1$ **to** s **do**
    $\mathbf{x}_k \sim \mathcal{D}$ //draw sample
    $(\boldsymbol{\mu}_{\mathbf{x}_k}, \boldsymbol{\sigma}_{\mathbf{x}_k}) \leftarrow \text{SNES}(f, \boldsymbol{\mu}_{\mathbf{x}_k}, \boldsymbol{\sigma}_{\mathbf{x}_k}, \lambda(C_k), n)$
    $\phi_{\mathbf{x}_k} \leftarrow f(\boldsymbol{\mu}_{\mathbf{x}_k})$ //store fitness
  foreach $\mathbf{x}_i \in \mathcal{D}$ **do**
    $$g(\mathbf{x}_i) \leftarrow \begin{cases} \sum_{\forall \mathbf{x}_j \in \mathcal{D}} \phi_{\mathbf{x}_j} \frac{1}{h^d} \mathcal{K}\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right) & \max(\boldsymbol{\sigma}_{\mathbf{x}_i}) > \sigma_\theta \\ 0 & \text{otherwise} \end{cases}$$
  foreach $\mathbf{x}_i \in \mathcal{D}$ **do**
    $p(\mathbf{x}_i) \leftarrow \frac{g(\mathbf{x}_i)}{\sum_{\forall \mathbf{x}_j \in \mathcal{D}} g(\mathbf{x}_j)}$ //normalize

loop, the array is partitioned into $(D_m - 1)$-simplexes, where each simplex, s_i, contains only those elements e whose Cartesian coordinates, $(\xi_1, \ldots, \xi_{D_m})$, sum to integer i. The elements of simplex s_i are ordered in the `while` loop according to their distance to the corner points, p_i (i.e. those points having exactly one non-zero coordinate; see example points for a 3D-array in figure 2), which form the rows of matrix $K = [p_1, \ldots, p_m]^T$, sorted in descending order by their sole, non-zero dimension size. In each loop iteration, the coordinates of the element with the smallest Euclidean distance to the selected corner is appended to the list *ind*, and removed from s_i. The loop terminates when s_i is empty.

After all of the simplexes have been traversed, the vector *ind* holds the ordered element coordinates. In the final loop, the array is filled with the coefficients from low to high frequency to the positions indicated by *ind*; the remaining positions are filled with zeroes. Finally, a D_m-dimensional inverse DCT transform is applied to the array to generate the weight values, which are mapped to their position in the corresponding 2D weight matrix. Once the k chromosomes have been transformed, the network is complete.

3 Compressed Network Complexity Search

The basic idea of CNCS is to discover networks with minimal complexity by running multiple independent evolutionary processes in parallel, one for each complexity class, allocating run-time to each according to an adaptive probability mass function, $\mathcal{D}$. Algorithm 2 describes CNCS in pseudocode. The algorithm is initialized with a prior distribution over the complexity classes (C, Ψ), where C is the number of coefficients used to encode the network, and Ψ is the number of neurons (equivalently, the topology). In order to bias the search toward low-complexity solutions, $\mathcal{D}$ should be initialized with a prior that gives high probability to small networks (low Ψ), represented by the fewest number of coefficients (low C).

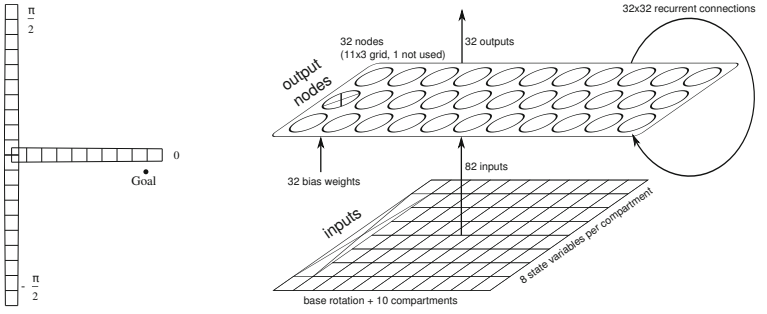


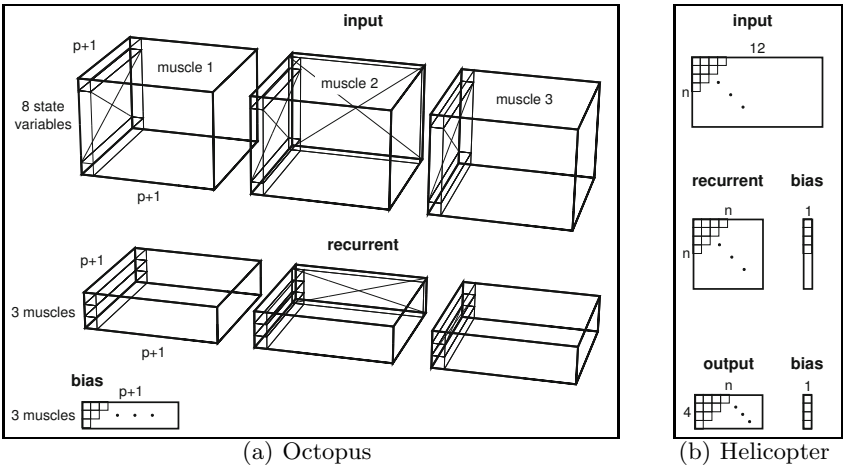
Fig. 3. Octopus arm task. (a) A flexible arm consisting of p compartments, each with 3 muscles, must be controlled to touch a goal location with the arm tip from 3 different initial positions, $-\pi/2$, 0 and $\pi/2$. (b) The arm is controlled by a fully recurrent network with 32 neurons, one for each action (muscle). This topology is fixed, and only the number of coefficients used to represent its weights is determined automatically by CNCS.

Each $\mathbf{x}_\iota = (C_\iota, \Psi_\iota)$ pair in $\mathcal{D}$ has its own dedicated search algorithm used to optimize that particular configuration. In the current implementation we use Separable Natural Evolution Strategies (SNES; [13]), an efficient variant in the NES [12] family of black-box optimization algorithms. In each generation, SNES samples a population of λ individuals, computes a Monte Carlo estimate of the fitness gradient, transforms it to the natural gradient and updates the search distribution parameterized by a mean vector, $\boldsymbol{\mu}$, and *diagonal* covariance matrix, $\boldsymbol{\sigma}$ (see [12] for a full description of NES). The SNES search distribution associated with configuration $\mathbf{x}_\iota$ has mean $\boldsymbol{\mu}_{\mathbf{x}_\iota}$ and covariance $\boldsymbol{\sigma}_{\mathbf{x}_\iota}$.

Each iteration, CNCS draws s samples from $\mathcal{D}$, and runs the SNES corresponding to each sample for n generations, after which the search distribution $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ and its expected fitness value ϕ are saved. The distribution $\mathcal{D}$ is then re-estimated using a multivariate Parzen window estimator with radial-symmetric Gaussian kernel $\mathcal{K}$ [8]. First, the values $g(\mathbf{x})$ are computed by applying the kernel weighted by the normalized fitnesses, $\phi_{\mathbf{x}_j}$ (the first `forall` loop), where h is the kernel width, and d is the dimensionality of $\mathcal{D}$, e.g. 2 when estimating C and Ψ (the SNES distributions that have converged, $\max(\boldsymbol{\sigma}) \leq \sigma_\theta$, are assigned a g value of 0). Then the g values are normalized into probabilities, and the cycle repeats. The algorithm terminates when all search distributions (within the bounds of $\mathcal{D}$) have converged, or either the desired fitness or the maximum number of iterations has been reached.

4 Octopus Arm Control

The octopus arm consists of p compartments floating in a 2D water environment (see figure 3a). Each compartment has a constant volume and contains three



(a) Octopus (b) Helicopter

Fig. 4. Network decoding schemes. (a) for the octopus arm networks the genome is split into 3 chromosomes, Ω=$\{4, 4, 2\}$. The coefficients in the chromosome used to generate the input weight matrix are placed in a 4D array, and a 2D array for the bias weights. (b) for the helicopter networks there are 5 chromosomes, Ω=$\{2, 2, 1, 2, 1\}$: a 2D input and recurrent and output arrays, and 1D arrays for the input and output bias weights.

controllable muscles (dorsal, transverse and ventral). The goal of the task to reach a target position with the tip of the arm, starting from three different initial positions, by contracting the appropriate muscles at each 1s step of simulated time. While initial positions $-\pi/2$ and $\pi/2$ look symmetrical, they are actually quite different due to gravity. The state of a compartment is described by the x, y-coordinates of two of its corners plus their corresponding x and y velocities. Together with the arm base rotation, the arm has $8p+2$ state variables. Though there are $3p + 2$ muscles, the task is normally simplified by aggregating them into 8 "meta"-actions that contract groups of muscles simultaneously (i.e. all dorsal, all transverse, etc.). Here, instead we use the more difficult configuration where the "raw" actions are controlled directly, so that each muscle must be coordinated with the others to move the arm.

4.1 Setup

For the $p = 10$ compartment arm used in these experiments, a network with 32 neurons (one for each raw action) is sufficient to perform well on this task. Therefore, CNCS is used only to search for the number of coefficients, so that the distribution, $\mathcal{D}$, is one-dimensional, with a uniform prior over $C = 1..10$, a sample size of $s = 1$, and number of SNES generations per CNCS update, $n = 1$. The kernel width was $h = 7$, and the convergence condition was set to $\sigma_\theta = 0.01$. Each SNES used a population size of 16. Five experiments were run, for 4 thousand iterations each.

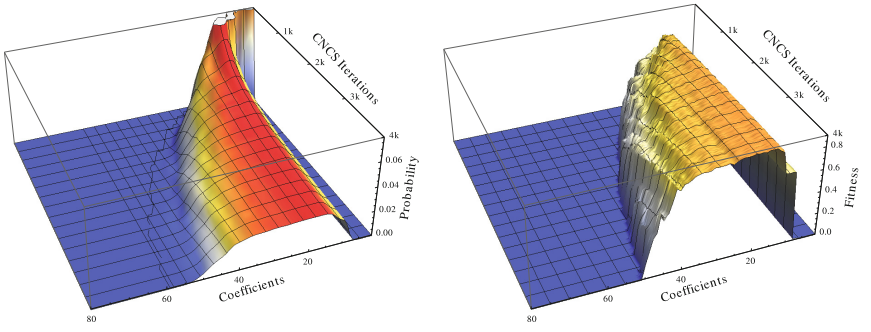


Fig. 5. Octopus arm results. The 3D plots show how (a) the distribution over the number of coefficients, C, adapts over time in CNCS, and (b) configurations with a better fitness are explored. The final distribution after 4,000 iterations is focused between $C = 15$ and $C = 50$, and peaks at $C = 18$, for a compression ratio of 204:1; 3680 weights/18 coefficients and fitness reaching 0.88.

The octopus arm was controlled using the fully-recurrent network architecture shown in figure 3b which is decoded using the following scheme, $\Omega = \{4, 4, 2\}$, depicted in figure 4a: the genome is partitioned into $k = 3$ chromosomes, mapped into three arrays: (1) a 4D $8\times(p+1)\times3\times(p+1)$ array that contains input weights for a $3\times(p+1)$ grid of neurons, one for each raw action, (2) a $3\times(p+1)\times3\times(p+1)$ recurrent weight array, and (3) and a $3\times(p+1)$ bias array. The dimension size of 3 in these arrays refers to the number of muscles per compartment.

The fitness was computed as the average of the following score over three trials: $\max\left[1 - \frac{t}{T}\frac{d}{D}, 0\right]$, where t is the number of time steps before the arm touches the goal, T is the maximum number of time steps in a trial, d is the final distance of the arm tip to the goal and D is the initial distance of the arm tip to the goal. Each of the three trials starts with the arm in a different configuration (see figure 3a). This fitness measure is different to the one used in [14], because minimizing the integrated distance of the arm tip to the goal causes greedy behaviors. In the viscous fluid environment of the octopus arm, a greedy strategy using the shortest length trajectory does not lead to the fastest movement: the arm has to be compressed first, and then stretched in the appropriate direction. Our fitness function favors behaviors that reach the goal within a small number of time steps.

4.2 Results

Figure 5 shows how the distribution (a) over coefficients, C, and the fitness of each configuration (b) adapts over the course of 4,000 iterations of CNCS (averaged over 20 runs). The distribution is initialized with a prior that favors networks with low complexity, expressed solely with C (top-right corner of the graphs). The expected fitness forms a ridge with a peak between $C = 15$ and $C = 18$ with a moderate slope on the left, towards higher C. This focuses the search between $C = 15$ and $C = 50$. The expected fitness for $C < 15$

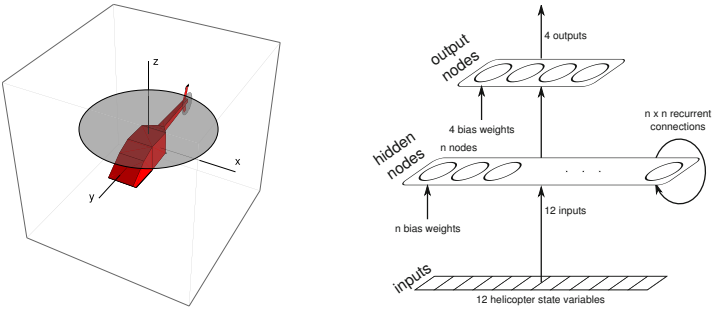


Fig. 6. Helicopter hovering task with gusting wind. (a) The helicopter must be controlled to stay within 20m from its initial position at the origin (as shown). Unlike the standard version of the task where wind blows at a constant velocity, the wind here blows in gusts with much higher velocity, forcing the controller to react quickly to sudden changes in wind. (b) The helicopter is controlled by a simple recurrent network (SRN). Both the number of neurons and the number of coefficients are determined automatically by CNCS.

drops significantly lowering the probability of sampling configurations in this area. Reasonable fitness of 0.75 was reached at iteration 350 using less than 15 coefficients. A fitness of 0.88 that is close to optimum was reached at iteration 2560 using 18 coefficients. The weight matrices of such networks were compressed down from 3680 weights (compression ratio of 204:1).

5 Helicopter Hovering with Gusting Wind

The standard Helicopter Hovering benchmark involves maintaining the position of a simulated XCell Tempest [1] as close as possible to origin of a bounded 3D space (see figure 6a). The helicopter model consists of 12 state variables: the coordinates and angular rotations in 3-space and their derivatives; and 4 control variables: longitudinal and latitudinal cyclic pitch and tail, and main rotor collective pitch. The fitness is the sum of squares of all state variables over the course of a flight lasting t time steps. If the helicopter moves more that 20m away from the origin in any direction or its velocity exceeds 5 m/s, then it is considered to have crashed, and the trial is terminated. The fitness is normalized between 0 and 1 and the minimum over 5 trials is used.

The original 2008 RL competition version of this problem featured wind along the x- and y-axes with a drag of up to 5m/s, which is initialized at random in the beginning of each trial. With this setup, it turns out that a linear controller can be easily trained to solve the task. Therefore, in order to make the task more challenging, requiring non-linear control, the original wind model was modified so that instead of constant wind, strong "gusts" buffet the helicopter at random. Wind gusts occur in both x and y directions with probability 0.4 and a velocity of 20m/s, which decays exponentially after striking the helicopter. The gusting

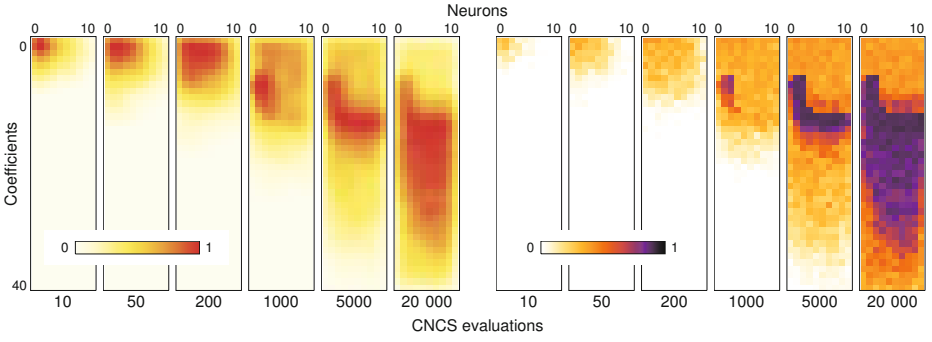


Fig. 7. Complexity search distribution for helicopter hovering task. The figure shows how the distribution $\mathcal{D}$ (left) and fitness (right) evolve over time, averaged across 20 runs. The algorithm first explores simple networks, with only a few neurons, represented by a small number of DCT coefficients (around 500 steps), then gradually spreads the distribution toward more complex configurations, identifying two clusters with high fitness (networks with 3 neurons defined with 8 coefficients and 3-node networks defined with 12 coefficients).

wind makes the task significantly harder—a linear controller cannot cope with the abrupt wind perturbations. Because higher velocities are required to control the helicopter under these conditions, the limit on the velocity was removed.

5.1 Setup

The helicopters were controlled using simple recurrent networks (SRN/Elman; figure 6b). The decoding scheme for the genomes, $\Omega = \{2, 2, 1, 2, 1\}$, is depicted in figure 4b. The original benchmark involves flights of $t = 6000$ time steps (equivalent to 60s flight). For the task with gusting wind used here, this was reduced to 100 both for efficiency, and because, due to the severity of the wind, such a short trial is enough to evaluate relative controller competence.

CNCS was used to search for both the network topology, Ψ (number of neurons), and number of coefficients, C. The complexity distribution, $\mathcal{D}$, was initialized with a uniform prior over the range $\{1, 2\}$ for both Ψ and C, i.e. $p(C, \Psi) = 0.25$, $C, \Psi = 1, 2$, and a sample size $s = 2$. As in the octopus task, $h = 7$, $\sigma_\theta = 0.01$, $n = 1$, and each SNES used a population size of 16. A total of 20 experiments were run, for 20 thousand iterations each.

5.2 Results

Figure 7 shows how the distribution over (Ψ, C) configurations (top row) and the fitness of each configuration (bottom row) adapts over the course of 20k iterations of CNCS (averaged over 20 runs). The distribution is initialized with a prior that concentrates on networks with the lowest complexity (upper-left corner of the graphs). Gradually, as the distribution expands, it finds high fitness individuals with 1 to 3 neurons, using ≈ 8 coefficients, at around iteration

1000. These networks are simple both in terms of their topology (model complexity) and in the regularity of their weight matrices (compression ratio of 8:1, 64 weights/8 coefficients). The distribution then focuses on this area, moving away from configurations with fewer coefficients ($C < 8$) as they cannot express the level of complexity required for nets with more that 3 neurons. At this point, the distribution begins to follow a narrow, high-fitness corridor, adding coefficients to networks with 2 and 3 neurons, until it reaches $C = 12$ (≈ 2000 iterations) and starts to grow the size of networks. The shape of the distribution at 20k iterations emerged consistently for all runs, with a maximum relative entropy between the distributions of any two runs of only 0.038.

6 Discussion and Future Work

CNCS consistently found low-complexity solutions for the two tasks tested. The octopus arm reaching task was successfully solved with networks having 3680 weights that were generated using just 18 DCT coefficients, and networks with 2 neurons (64 weights) represented by 8 DCT coefficients were found in the early stage of the CNCS for the Helicopter Hovering task. Helicopter networks were progressively improved by increasing the number of DCT coefficients beyond 12, and broadening the search to more hidden neurons.

The experimental results show that updating the distribution on complexity classes elegantly addresses the question of how to configure the evolutionary search. Running all configurations in parallel would be prohibitive, whereas CNCS quickly adapts the search distribution towards promising configurations.

A potential drawback of CNCS lies in wide valleys of low fitness that span across complexity space. One has to ensure, that the width of the smoothing kernel used is wider than the potential valley. Otherwise, the distribution will never reach across to sample networks of higher complexity. A possible solution could be to use a variable kernel width for each complexity class based on e.g. the number of samples evaluated from that class.

Future experiments will test the generalization of the evolved controllers to verify whether complexity is correlated with robustness. We expect that the small networks, although they have slightly worse fitness during the training (like those small networks that can control the helicopter to hover), will generalize better.

Acknowledgments. This research was funded by SNF grant 200020-125038/1.

References

1. Abbeel, P., Ganapathi, V., Ng, A.Y.: Learning vehicular dynamics, with application to modeling helicopters. In: NIPS (2005)
2. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with Analog Genetic Encoding. 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. 671–680. Springer, Heidelberg (2006)

3. Gruau, F.: Cellular encoding of genetic neural networks. Technical Report RR-92-21, Ecole Normale Superieure de Lyon, Institut IMAG, Lyon, France (1992)
4. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. Complex Systems 4, 461–476 (1990)
5. Koutník, J., Gomez, F., Schmidhuber, J.: Evolving neural networks in compressed weight space. In: Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2010 (2010)
6. Koutník, J., Gomez, F., Schmidhuber, J.: Searching for minimal neural networks in fourier space. In: Proc. of the 4th Conf. on Artificial General Intelligence (2010)
7. Levin, L.A.: Universal sequential search problems. Problems of Information Transmission 9(3), 265–266 (1973)
8. Parzen, E.: On estimation of a probability density function and mode. The Annals of Mathematical Statistics 33(3), 1065–1076 (1962)
9. Schmidhuber, J.: Discovering neural nets with low Kolmogorov complexity and high generalization capability. Neural Networks 10(5), 857–873 (1997)
10. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10, 99–127 (2002)
11. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life 9(2), 93–130 (2003)
12. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural Evolution Strategies. In: Proceedings of the Congress on Evolutionary Computation (CEC 2008), Hongkong. IEEE Press (2008)
13. Wierstra, D., Schaul, T., Sun, T.G.Y., Schmidhuber, J.: Natural evolution strategies. Technical report (2011), arXiv:1106.4487v1
14. Woolley, B.G., Stanley, K.O.: Evolving a Single Scalable Controller for an Octopus Arm with a Variable Number of Segments. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 270–279. Springer, Heidelberg (2010)
15. Zhang, B.-T., Muhlenbein, H.: Evolving optimal neural networks using genetic algorithms with Occam's razor. Complex Systems 7, 199–220 (1993)
16. Zhang, B.-T., Muhlenbein, H.: Balancing accuracy and parsimony in genetic programming. Evolutionary Computation 3, 17–38 (1995)

Single Node Genetic Programming on Problems with Side Effects

David Jackson

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

Abstract. Single Node Genetic Programming (SNGP) offers a new approach to GP in which every member of the population consists of just a single program node. Operands are formed from other members of the population, and evolution is driven by a hill-climbing approach using a single reversible operator. When the functions being used in the problem are free from side effects, it is possible to make use of a form of dynamic programming, which provides huge efficiency gains. In this research we turn our attention to the use of SNGP when the solution of problems relies on the presence of side effects. We demonstrate that SNGP can still be superior to conventional GP, and examine the role of evolutionary strategies in achieving this.

1 Introduction

Single Node Genetic Programming (SNGP) [1] is a newly-introduced form of GP in which each individual in the population consists of just a single program node drawn from the terminal or function set of the problem we are attempting to solve. The operands for a node are other members of the population. As such, it could be argued that the population forms one large graph; however, we do not treat it as such during evolution. In addition to the attributes encoding connections with other members, each individual has data structures recording the outputs it produces when evaluated; more importantly, it has its own distinct fitness value. It therefore makes much more sense to view the population as a set of graphs, with each individual holding the root node of an expression or program to be evaluated.

This approach is vastly different from other forms of GP, including those in which alternatives to conventional tree-based structures are employed. Most systems treat individuals as distinct, separate structures (although some hierarchical approaches have made use of limited forms of interconnectedness between members). Even when the values of internal graph nodes become important (such as in Oltean's Multi-Expression Programming [2,3]), the population is still constructed as a set of multi-node graphs, each unconnected to the others.

Structural considerations are not the only differences between SNGP and other forms of GP, however. In conventional GP and most other variants, the evolutionary operators are reproduction by cloning, recombination via subtree or segment crossover, and mutation. By contrast SNGP has only one evolutionary operator, and

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 327–336, 2012.

its effects are reversible. SNGP uses a form of hill-climbing in which evolutionary changes which do not lead to improvements are undone.

In the initial experiments we have performed, SNGP has demonstrated substantial improvements over conventional GP in terms of solution rates, execution times, and sizes of solutions obtained. One of the reasons its performance is so good is that it is able to exploit a form of dynamic programming in which the outputs obtained during the evaluation of each node are recorded as an attribute of that node. This means that any function requiring the values of its operand sub-graphs merely has to fetch those values directly from the vectors in which they are stored, without the necessity for further node evaluations.

An approach such as this works well for problems in which the nodes are purely functional and do not have side effects, i.e. it is the outputs of the function that are of interest, and these outputs depend only on the inputs supplied. Examples of such problems in GP are symbolic regression and various Boolean problems such as even-parity.

It is a different matter when problems rely on behavioural side effects. Such problems make use of nodes which modify state or have some form of interaction with an external world. This usually means that behaviour is reliant not just on current inputs but also on previous history. A good example of this type of problem in the context of GP is the Santa Fe artificial ant problem [4], in which the aim is to evolve a program that guides an agent along a trail of 'food' particles. In this problem, function outputs are of no relevance. Instead, the fitness of a program is ascertained via the side effects of those functions on a model of the ant's world. Because of this, the type of dynamic programming previously used in SNGP cannot be employed. Note that this does not necessarily imply that SNGP is not capable of solving such problems, merely that its efficiency will be hampered in doing so, since full evaluation is required of the tree rooted at each individual.

In this paper, then, we investigate the extent to which SNGP is a suitable system for the evolutionary solution of problems with side effects. As we shall see, this work entails re-examination of the evolutionary strategy used to drive SNGP.

2 Related Work

SNGP is, of course, not the first approach to deviate from Koza-style GP [4], in which programs are stored as tree structures and evolutionary operators work by swapping subtrees or replacing them with new, randomly-generated subtrees. In linear GP [5], for example, programs are simply sequences of individual instructions; and 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 of memory variables.

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) [6]. 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) [7] uses a grid representation to hold graph-structured programs. Individuals are still subject to (suitably modified) crossover and 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) [8], 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 [9]. In the GRAPE (GRAph structured Program Evolution) approach [10], 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. In linear-tree GP [11], 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 [12]. In the MIOST system [13], 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) [2,3], 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. As we shall see, an SNGP population can be viewed as being analogous to a single MEP individual, although the mechanics of evolution are very different.

3 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.

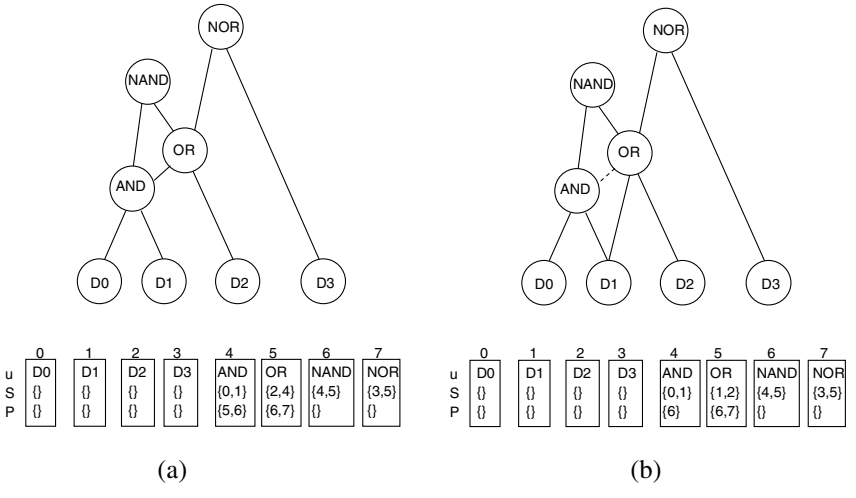


Fig. 1. SNGP graph structure and effects of the *smut* operator

Fig. 1(a) 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.

If the type of problem is one in which functions and terminals do not have side effects, then we can employ a form of dynamic programming which substantially enhances the efficiency of SNGP. 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. Of course, when side effects are present, as they are in the problems studied later in this paper, the use of such a mechanism is ruled out, and every node contained in a graph must be evaluated fully.

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 modified to refer to a different member of the population (but still lower down in the position order). Figure 1(b) shows how this operator is applied. 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 1(b), 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. In Figure 1(a), 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 function is invoked more than once.

Evolution of an SNGP population is driven using a hill-climbing approach. Whenever the smut operator is applied, the fitnesses of the affected individuals are re-assessed. If fitness has not improved, then the modifications made by smut are reversed. To make this more efficient, the old outputs (if outputs are being recorded) 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.

This begs the question of how we determine whether fitness has improved. In the original version of SNGP, this was ascertained by considering fitnesses across the population as a whole. In this strategy, the aim is to drive down the aggregate fitness of the population (and therefore the average fitness). More formally, and assuming that lower fitness values are better, the aim is to minimize Σr_i. One of the things called into question during the research described here is whether that is always the best strategy.

4 Experimentation

For the purposes of evaluation in situations where the dynamic programming approach previously employed is not possible, we have chosen three problems which rely on side effects during program execution.

The first of these problems is the Santa Fe artificial ant problem [4], which is commonly used in assessing the effectiveness of GP algorithms and is known to be difficult to solve [14]. The second test problem we have used is that of navigating a maze. Although less well-known than the ant problem, it has been used as the subject for research on introns in several studies [15-17]. In our third problem, the aim is to evolve programs which are capable of parsing arithmetic and logical expressions. The output of a successful parser is the postfix (Reverse Polish) form of each expression, but the need to manipulate a stack during execution means that functions must rely on side effects to achieve this aim. Full details of this problem can be found elsewhere [18].

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 50, 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 creates 25,000 individuals via crossover or reproduction, we will set the upper limit on the number of *smut* applications to 25,000.

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

As mentioned in an earlier section, the criterion used in the original version of SNGP to decide whether to reverse the effects of an evolutionary operation is based on the average fitness of the population. If the average fitness (or, to be more precise, the aggregate fitness) worsens, the operation is undone. Henceforth, we will use the notation SNGP/A to refer to the approach when it makes use of an evolutionary strategy based on *Average* fitness. An alternative strategy is to use best fitness, rather than average fitness, as our criterion: if the best fitness in the population worsens, reverse the operation. The term SNGP/B will be used to refer to SNGP when it makes use of a strategy based on *Best* fitness.

Table 2. Comparisons of SNGP with standard GP on example problems

Problem	**System**	**Soln. rate (%)**	**Effort (evals/soln) (x 10^6)**	**Time 100 runs (secs)**	**Av. soln size**	**Max. soln size**	**Min. soln size**
Ant	GP	9	308	50	51	199	25
	SNGP/A	16	2014	298	14	24	7
	SNGP/B	56	260	131	12	21	7
Maze	GP	47	8	7	1987	5620	722
	SNGP/A	98	18	18	33	45	10
	SNGP/B	60	15	9	24	37	9
Parse	GP	32	415	101	399	1154	58
	SNGP/A	45	4406	1073	19	30	9
	SNGP/B	83	355	159	19	34	11

Table 2 compares SNGP against standard GP for our problems. In relation to performance, three measures are used: solution rate, computational effort, and execution time. The solution rate is simply the percentage of full solutions obtained over 100 runs of the problem. For computational effort it would be possible to count fitness evaluations, but since the sizes of SNGP programs and the way in which they are executed differs enormously from standard GP, we chose a different measure that is more reflective of the effort involved. We count the total number of program node evaluations over all runs and divide this by the number of solutions obtained, thus

giving a notion of effort per solution. Finally, timings are given for execution of 100 runs. These were taken on a PC with an Intel Core i7 quad-core processor running at 2.8GHz. Both the SNGP and GP systems were written in C and compiled using Microsoft Visual Studio as single-threaded processes running under identical load conditions.

The first thing to note is that, at least in terms of solution rate, both forms of SNGP perform very much better than conventional GP. In comparing SNGP strategies, it would seem that, for two of our problems (artificial ant and parsing), strategy B is the one to opt for. Even though this strategy takes longer to execute 100 runs than standard GP, its much higher solution rate means that less effort is required per solution. Strategy A, on the other hand, requires an inordinate amount of effort to find its solutions, entailing lengthy run times. For the maze navigation problem the situation is perhaps not as clear-cut. Strategy A has a much higher solution rate, and in fact is able to discover solutions on almost every run. However, it requires slightly more effort per solution to achieve this, and requires double the execution time for the runs. The choice of a winner here rests on whether one prefers lots of solutions, or fewer solutions in a faster time.

An important point to make here is that the performance differences between SNGP and standard GP have been verified as statistically significant. This has been done by recording the fitness values of the best programs found in each run (whether forming a solution or not), and then performing a t-test on these data with p=0.05.

Turning to solution sizes, there is no contest. SNGP clearly outperforms standard GP for all three problems, with little to choose from between the two evolutionary strategies. In the case of the maze and regression problems, the solutions found by SNGP are many times smaller than those found by standard GP.

The compactness of the SNGP programs merits further discussion. Since any SNGP program is built only from the nodes contained in the population members, it cannot be larger than the population size. In the experiments described here, this means an upper bound of 50 nodes in any program. That said, the graph-like nature of these programs allows code re-use that is not present in conventional GP trees and which would otherwise require many more nodes to implement. For example, one 40-node SNGP solution to the maze problem would require 4953 nodes if written out as a tree-based GP expression.

Key to the solution sizes obtained is the size of the population, which by definition in SNGP acts as a constraint. In the experiments above, the population size N was set arbitrarily at 50. However, this is not necessarily an optimum for each problem. In contrast to standard GP, one of the advantages of SNGP is that it can find solutions even when N is set very low. Usually this means that fewer solutions will be found, but they will be smaller programs, found in a shorter time. Hence, via the single parameter N, one can tune the system to select an appropriate balance of solution count, program size and execution time. Sometimes, however, this tuning can lead to surprising results. For example, setting N to just 20 for the ant problem using strategy B leads to seven times as many solutions as standard GP, and at a third of the computational cost per solution.

5 Conclusions

Single Node Genetic Programming (SNGP) is a new approach to GP in which each individual in the population consists of just a single program node. Where a node requires operands, these are drawn from other members of the population. Evolution is driven by a hill-climbing mechanism that uses a single reversible operator.

The research described in this paper began as a test of how well SNGP could cope with problems in which the functions and terminals used to build programs rely on side effects for their behaviour. In previous experiments, side effects were not present, and so it was possible to make use of a form of dynamic programming in which the outputs of subtrees could be cached for use as operands by higher-level functions, thereby leading to enormous efficiency gains. Without the opportunity to make use of this mechanism, it was known that efficiency would suffer, but it was hoped that SNGP would be at least competitive with conventional GP systems.

In the event, our experimentation showed that SNGP can be superior in terms of solution-finding performance, computational effort, and solution size, with the caveat that is influenced heavily by the choice of evolutionary strategy. We investigated two strategies: one which promotes average fitness across the population, and the other concentrating on best fitness amongst individuals. In a sense, the first strategy could be described as altruistic, with the other being more selfish. In previous work, the altruistic strategy was generally found to lead to better results. In the work described in this paper, however, the selfish appears better, at least for two of the three problems. What we do not know is why this difference should exist. We hope to carry out further investigations to provide some insight, and in particular to assist us in deciding on a strategy to employ based purely on an *a priori* description of a problem.

Other research lined up for the future includes an investigation into the dynamics of SNGP, to discover how it is able to find solutions so readily with such small populations. We also wish to explore ways to exploit the parallelism inherent in both the SNGP system and in the programs it evolves. And in the same way that we have explored alternative evolutionary strategies here, we want to evaluate the effects of using different operators, initialisation procedures and algorithms.

References

1. Jackson, D.: A New, Node-Focused Model for Genetic Programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 49–60. Springer, Heidelberg (2012)
2. 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)
3. 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)
4. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)

5. Brameier, M., Banzhaf, W.: Linear Genetic Programming. Springer, Heidelberg (2007)
6. 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)
7. Poli, R.: Parallel Distributed Genetic Programming. In: Corne, D., et al. (eds.) New Ideas in Optimization, pp. 779–805. McGraw-Hill Ltd., UK (1999)
8. 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)
9. 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)
10. 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)
11. 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)
12. 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)
13. Galvan-Lopez, E.: Efficient Graph-Based Genetic Programming Representation with Multiple Outputs. International Journal of Automation and Computing 5(1), 81–89 (2008)
14. Langdon, W.B., Poli, R.: Why Ants are Hard. In: Koza, J.R., et al. (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference, pp. 193–201. Morgan Kaufmann (1998)
15. Jackson, D.: Dormant Program Nodes and the Efficiency of Genetic Programming. In: Beyer, H.-G., et al. (eds.) Proc. Genetic and Evolutionary Computing Conf (GECCO 2005), Washington DC, pp. 1745–1751. ACM Press, New York (2005)
16. Langdon, W.B., Soule, T., Poli, R., Foster, J.: The Evolution of Size and Shape. In: Spector, L., et al. (eds.) Advances in Genetic Programming, vol. 3, pp. 163–190. MIT Press, Cambridge (1999)
17. Soule, T.: Code Growth in Genetic Programming. PhD Thesis, University of Idaho (1998)
18. Jackson, D.: Parsing and Translation of Expressions by Genetic Programming. In: Beyer, H.-G., O'Reilly, U.-M. (eds.) Proc. Genetic and Evolutionary Computation Conf (GECCO), Washington, DC, pp. 1681–1688. ACM Press, New York (2005)

Generalized Compressed Network Search

Rupesh Kumar Srivastava, Jürgen Schmidhuber, and Faustino Gomez

IDSIA
USI-SUPSI
Manno-Lugano, CH
{rupesh,juergen,tino}@idsia.ch

Abstract. This paper presents initial results of Generalized Compressed Network Search (GCNS), a method for automatically identifying the important frequencies for neural networks encoded as Fourier-type coefficients (i.e. "compressed" networks [7]). GCNS is a general search procedure in this coefficient space – both the number of frequencies and their value are automatically determined by employing the use of variable-length chromosomes, inspired by messy genetic algorithms. The method achieves better compression than our previous approach, and promises improved generalization for evolved controllers. Results for a high-dimensional Octopus arm control problem show that a high fitness 3680-weight network can be encoded using less than 10 coefficients using the frequencies identified by GCNS.

1 Introduction

Indirect or generative encoding schemes for neural network phenotypes [1,5,6,9] offer the potential of allowing very large networks to be represented compactly. In previous work [7], we showed that encoding neural network weight matrices indirectly as a set of Fourier-type coefficients can reduce the search space dimensionality and help to discover more 'regular' networks which are simpler in the Kolmogorov sense (the program required to encode them is much shorter). Such networks are expected to have better generalization capabilities [9].

However, up to now, this "compressed" network search has been restricted to band-limited networks where the genome includes all frequencies up to a specified limit frequency. This means that more genes must be searched than may be necessary, because only a few, select frequencies may be needed to represent a good network. In this work, we implement a more general approach which automatically determines the subset of frequencies and their amplitudes using a genetic algorithm with variable size chromosomes, where each gene specifies a frequency number as well as amplitude value. Taking inspiration from the messy genetic algorithms [2], cut and splice operators are used instead of crossover. By resolving the overspecification and underspecification problems arising from this less restrictive encoding, we are able to find genomes which represent high fitness networks using very few frequencies. Initial results are very encouraging: we are able to identify isolated frequencies which appear to contribute significantly to fitness, and which are not easily identified otherwise.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 337–346, 2012.

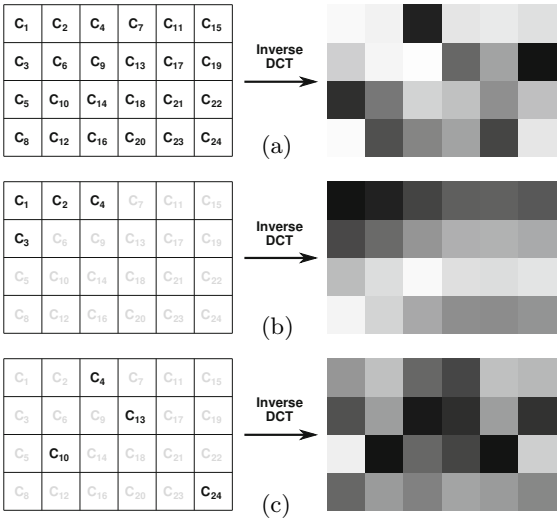


Fig. 1. DCT network representation. The left column shows three different types of 2D frequency-domain coefficient arrays. The coefficients are arranged along the second diagonals, going from upper-left corner, to the bottom right corner. Each diagonal is filled from the edges to the center starting on the side that corresponds to the longer dimension. The right column shows the weight matrix resulting from applying the inverse DCT transform; gray-scale levels denote the weight values (black = low, white = high). In (a) all frequencies are present, so that all possible weight matrices can be represented. (b) Shows a band-limited weight matrix where only the first four coefficients from (a) are used, as in [7]. The weights in (b) are more spatially correlated than those in (a). (c) Shows a weight matrix encoded by a subset of frequencies from (a). GCNS searches this space of coefficient subsets (power set) of (a).

2 DCT Network Representation

The Discrete Cosine Transform (DCT) representation for neural networks, first introduced in [8], encodes network weight matrices in the frequency domain by using genomes of DCT coefficients. The motivation is that if weights that are near each other in the matrix are correlated, then the representation of the matrix in the frequency domain should require fewer parameters (coefficients[1]) than the number of weights in the matrix, thereby reducing the dimensionality of the search space.

In this paper, all of the networks are fully connected recurrent neural networks (FRNNs) with i inputs, and single layer of n neurons where some of the neurons are treated as output neurons. An FRNN consists of three weight matrices: an $n \times i$ input matrix, $\mathbf{I}$, an $n \times n$ recurrent matrix, $\mathbf{R}$, and a bias vector $\mathbf{t}$ of length n. These three matrices are combined into one $n \times (n+i+1)$ matrix, and encoded

[1] In this paper, we will use the terms 'frequency' and 'coefficient' interchangeably. To be precise, every frequency is associated with a coefficient which expresses its energy content.

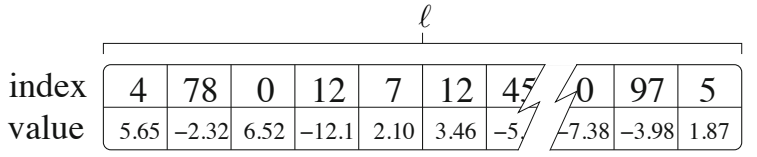


Fig. 2. GCNS coefficient genome. Each gene consists of two entries, the index of the DCT coefficient in the coefficient array, and the value of the coefficient. The same index can appear more than once in the genome, and genomes have variable length, ℓ.

indirectly using $c \leq N$ DCT coefficients, where N is the total number of weights in the network.

Figure 1 illustrates the relationship between the coefficients and weights for a hypothetical 4×6 weight matrix (e.g. a network with four neurons each with six weights). The left side of the figure shows three weight matrix encodings that use different coefficients. Generally speaking, coefficient c_i is considered to be more significant (associated with a lower frequency) than c_j, if $i < j$. The right side of the figure shows the weight matrices that are generated by applying the inverse DCT transform to the coefficients. In the first case (a), all 24 coefficients are used, so that any possible 4×6 weight matrix can be represented. The particular weight matrix shown was generated from random coefficients in $[-20, 20]$. In (b), each c_i has the same value as in (a), but the full set has been truncated up to the first four lowest frequencies, favoring smoother matrices. This is the approach taken in [7] where a limit frequency c_ℓ (c_4 in the example) is specified by the user, and genomes of length ℓ are evolved. In (c), the coefficient matrix again has only four non-zero coefficients, but the coefficients are not restricted to a band-limited spectrum; they can be at any frequency. The genomes evolved by GCNS search this less constrained space.

3 Generalized Compressed Network Search

Generalized Compressed Network Search (GCNS) attempts to simultaneously find the number of coefficients required to represent a high fitness network, their indices (2D frequency), and their values. Variable size chromosomes are used where each gene has two elements: the coefficient index and the value (see figure 2). The coefficient index determines the position of the coefficient in the coefficient matrix which is transformed into the network via the inverse DCT.

The overspecification problem (some genes can have multiple copies in the genome) is handled as in messy genetic algorithms [2,3]. If a coefficient index appears multiple times in a genome, only its first value, reading from left to right, gets expressed in the phenotype. This results in an *intra-chromosomal dominance operator*. The problem of underspecification (some of the frequencies do not appear in a particular genome) elegantly resolves itself due to the nature of the encoding: if a particular coefficient number does not appear in the genome, it is muted in the phenotype i.e. its value is taken to be zero.

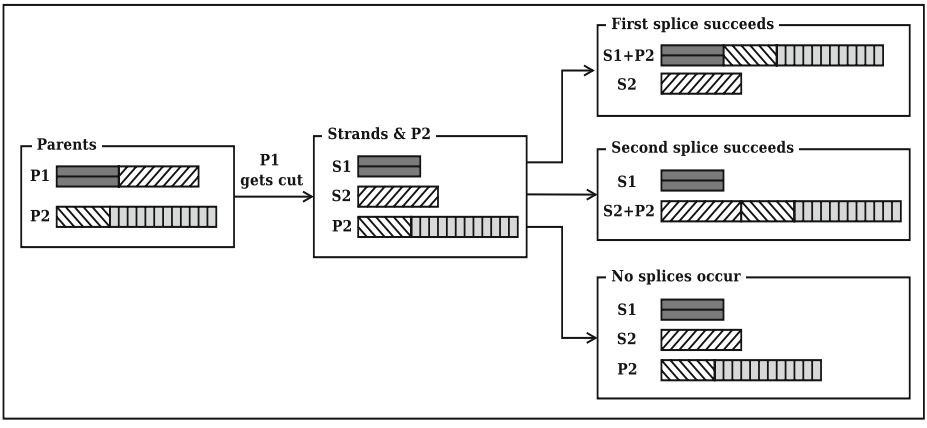
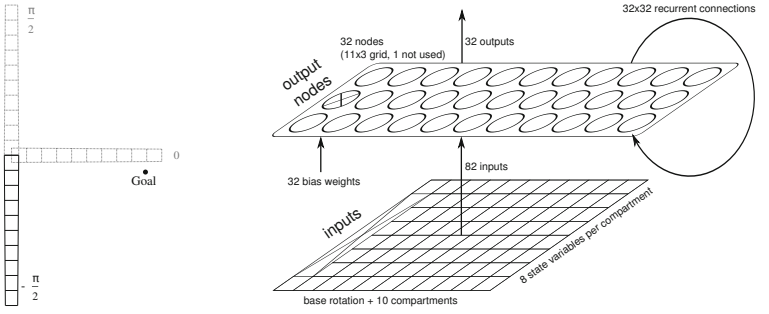


Fig. 3. Cut and Splice. This schematic shows the effect of application of the cut and splice operators on a set of two parent genomes. In the case shown, only P1 gets cut resulting in three chromosomes (strands S1, S2 and parent P2). Then splice is applied with probability p_s. If the first splice succeeds, then S1 gets spliced with P2, leaving S2 as a separate genome. If first splice does not occur, another splice between S2 and P2 can lead to the two children shown if it succeeds. If both splices do not succeed, S1, S2 and P2 become the final children as shown. Similar possibilities exist for other cases of the parents getting cut.

GCNS starts with an initial parent population of size *popsize* with genomes of variable lengths containing frequency indices and values randomly chosen in a given range. At each generation, the child population is formed from the parent population by applying the 'cut' and 'splice' operators in groups of two to randomly chosen members from the parent population (without replacement). The process of applying cut and splice is a generalization of the crossover operation to the variable length genome case, and can yield one to four children from two parents. First, it is determined whether one, both or none of the two parents will be cut. The probability of cut is given by $p_c * (l - 1)$ where l is the length of the genome and p_c is a parameter. The location of the cut on a genome is randomly chosen over its length. At this intermediate step, there are two to four chromosomes present depending on the number of cuts that occur. The splice operator then joins together pairs of chromosomes with probability p_s, resulting in either one (splice succeeds) or two (splice fails) children for each splice. Figure 3 shows the recombination of the parent genomes in an example scenario. As shown, when only parent 1 is cut, three possible sets of children can result after splicing. The other scenarios are handled similarly.

After cutting and splicing, mutation is applied to each coefficient index (with probability p_{m_i}) and value (with probability p_{m_v}) by drawing new values from Gaussian distributions centered at their current values and having fixed standard deviations. The value of p_{m_i} is kept much lower than p_{m_v} so that new frequencies are introduced only sporadically, allowing the algorithm to focus on refining the selected coefficients. In all our runs, the standard deviations were taken to be 5 and 10 for the indices and values, respectively.



(a) (b)

Fig. 4. Octopus arm task. (a) A flexible arm consisting of p compartments, each with 3 muscles, must be controlled to touch a goal location with the arm tip from the $-\pi/2$ position. The other two standard, initial positions were not used (see text). (b) The arm is controlled by a fully recurrent network with 32 neurons, one for each action (muscle). This topology is fixed, and only the number of coefficients used to represent its weights is determined automatically by GCNS.

After all the children have been evaluated, the best *popsize* members from the combined parent and child populations are chosen as the parents for the next generation. The algorithm terminates after the specified number of generations.

4 Octopus Arm Control

The octopus arm consists of n compartments floating in a 2D water environment [10]. Each compartment has a constant volume and contains three controllable muscles (dorsal, transverse and ventral). The state of a compartment is described by the x, y-coordinates of two of its corners plus their corresponding x and y velocities. Together with the arm base rotation, the arm has $8n + 2$ state variables and $3n + 2$ control variables. The goal of the task is to reach a goal position with the tip of the arm, starting from different initial positions, by contracting the appropriate muscles at each 1s step of simulated time. The standard setup uses 3 initial positions (figure 4); here, only one initial position was used for training (the arm starts hanging straight down), since it turns out the other two (indicated in gray, figure 4) are very easy to solve, and successful networks tend to generalize to them. The fitness function is given by $(1 - (t * d)/(T * D))$ where t is the number of time steps taken to reach the goal, d is final arm tip distance to the goal, T maximum the number of time steps in a trial, and D is the initial arm tip distance to the goal.

4.1 Setup

GCNS was run 30 times with $popsize = 100$, $ngen = 150$, $p_c = 0.2$, $p_s = 0.8$, $p_{m_i} = 0.1$ and $p_{m_v} = 0.8$. The initial population contained genomes of random length, ℓ, ranging from 2 to 20 genes, with indices chosen at random from $[1, 100]$,

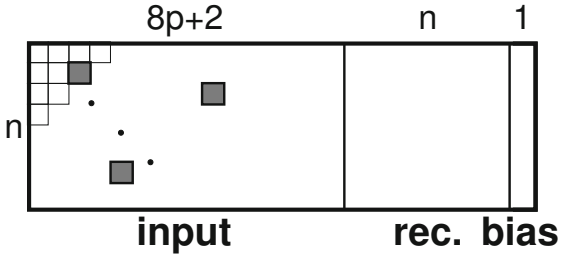


Fig. 5. Coefficient matrix. There is one coefficient matrix for all of the weights in the network. The small boxes in the upper left-hand corner denote coefficients organized in the usual way (as in [4,7]) in the matrix, from this corner to the opposite corner, in order of increasing frequency. In GCNS, genomes can instead contain frequencies from anywhere in the (bounded) spectrum (denoted by gray boxes), without having to include all lower frequencies. When the iDCT is applied to this matrix, a matrix of weights of the same size is generated, and sliced into three sub-matrices (indicated by vertical lines): one for the input weights, one for the recurrent weights, and a bias vector.

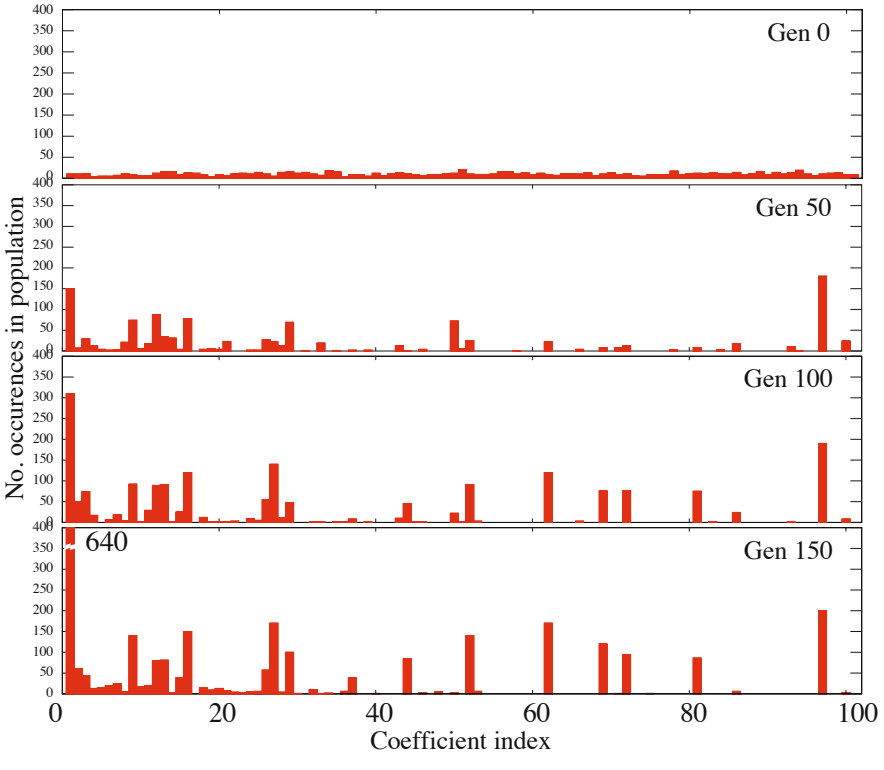


Fig. 6. Evolution of population frequency content. Each plot is a histogram of the coefficient indices (2D DCT frequencies) in the GCNS population of a typical run at particular point during evolution. In the initial random population (Gen 0), each frequency occurs about as often as any other. By generation 50 (Gen 50), about 20 frequencies have started to dominate the population.

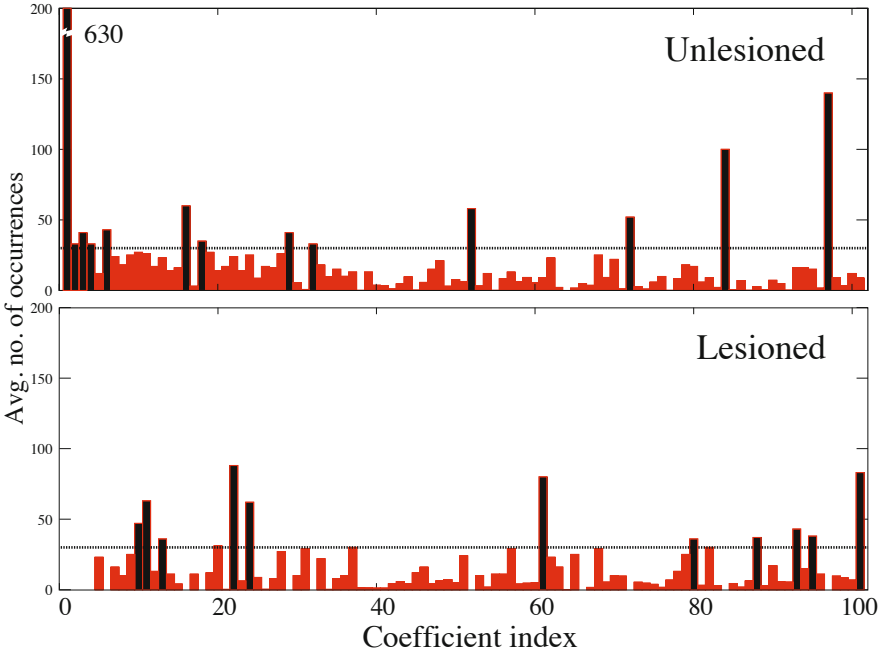


Fig. 7. Average frequency content in final populations. Each histogram shows the indices present in the final population averaged over the 30 runs. In the upper, *unlesioned* plot, the indices marked in black are those that cross the chosen threshold of 30 (horizontal line). These frequencies are muted in the *lesioned* runs (lower plot), where alternative solution indices emerge to compensate for those that are lesioned.

and values chosen at random from $[-30, 30]$. With this setup, no one genome contains all of the 100 available frequencies, but with very high probability all frequencies are present in the population.

4.2 Results

The mean best fitness over 30 runs was 0.95, while the average number of expressed genes (i.e. non-dominated) in the best genomes was 9.8, one-third the number required in [7] to achieve similar fitness. It is important to point out that our objective here is not to demonstrate raw performance, but to determine whether a small basis (set of frequencies) can be discovered and parameterized consistently.

Figure 6 shows how the frequency content in the population declines over the course of evolution as the search converges to just a few frequencies for the behavior of a typical run. Interestingly, we found that in addition to the fundamental frequency (index 1, which we expected), almost all of the most fit networks contained either index 84 or 97, with large values. The 2D cosine functions represented by these indices seem to capture a basic regularity inherent in the task, given the network architecture used.

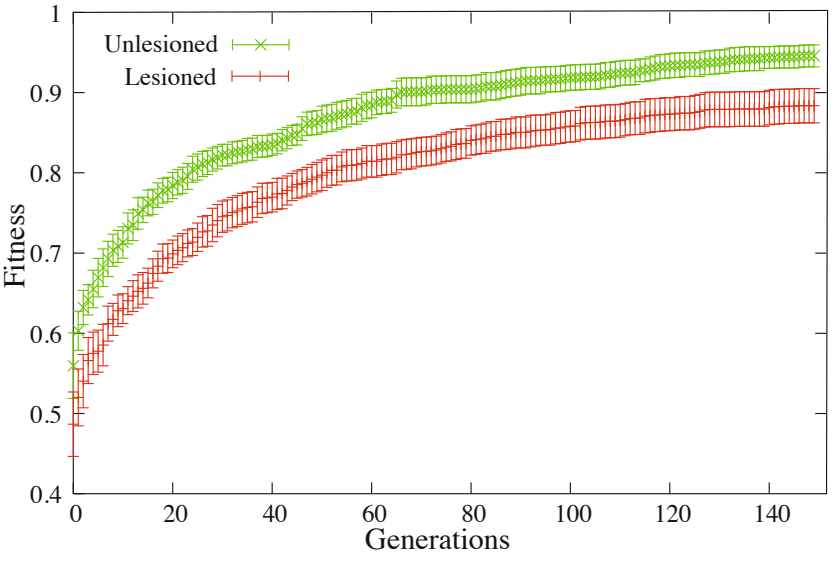


Fig. 8. Performance on Octopus arm task. The plot shows the fitness of the best network in each generation, averaged of 30 runs, with 95% confidence intervals. The upper curve is for the unlesioned case where all 100 coefficients are active in the evolving genomes; the lower curve is for the lesioned case where the 13 most common frequency indices found in the final populations of the unlesioned runs are "muted". Removing these frequencies from the set of available alleles slows down the search, forcing GCNS to find alternative solutions consisting of frequencies that are more difficult to set properly.

Figure 9 shows the weight matrices of three difference networks with high fitness and their GCNS genomes. All three have a very regular structure. The third network (figure 9c) can be completely specified with just 10 numbers, for a compression ratio of $3680/10 = 368$.

4.3 Lesion Study

In order to determine whether the frequencies that were found consistently in highly fit networks are somehow "special" in that it is easier to find good values for them, the experiments were run again, but this time the frequencies occurring most often in the final populations of the previous experiments (indices: 0, 1, 2, 3, 5, 15, 17, 28, 31, 51, 71, 83, 96) were not allowed to be expressed (figure 7, top). Any time one of these frequencies occurred in a genome its value was *muted* by setting it to zero.

Figure 8, compares the performance of GCNS using these lesioned genomes versus the unlesioned genomes in the first experiments. Without access to the lesioned frequencies, fitness improves more slowly reaching an average of 0.88. To do this, the lesioned runs are forced to use alternative frequencies (indices: 9, 10, 12, 21, 23, 60, 79, 87, 92, 94, 100; indicated in black in the bottom plot of figure 7) that take longer to set properly.

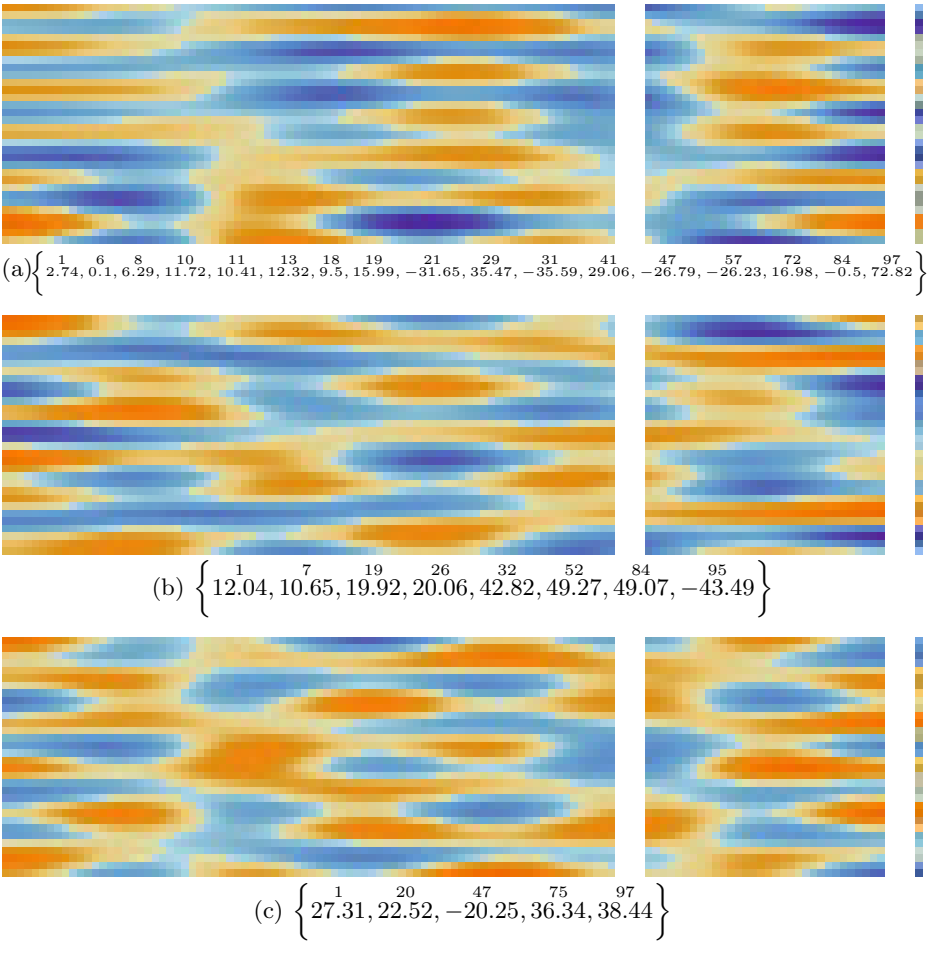


Fig. 9. Low-complexity weight matrices. Each row shows the weight matrices of a successful network (refer to figure 5 for a description of each sub-matrix). Colors indicate weight values. The genome used to generate the network is shown below each image.

5 Discussion and Future Work

The indirect encoding of neural networks using Fourier-type coefficients is promising since this scheme can reduce the dimensionality of the search space by orders of magnitude and allow very compact representations of the networks. If a particular problem suggests that a high degree of redundancy is expected in the network, this encoding can efficiently exploit this regularity. This has been demonstrated previously by searching for a fixed set of frequencies [4,7]. The present work aims to address some key issues of dealing with this encoding scheme.

First, the previous study used a fixed set of contiguous frequencies to encode the networks. In one run, a set of coefficient values corresponding to these frequencies was identified. However, it is uncertain what is the number of frequencies sufficient to encode a high fitness network. Thus, several runs must be repeated with increasing number of frequencies to ensure that sufficiently high fitness networks can be found. Moreover, the sufficient set of frequencies for a particular problem may not consist of contiguous frequencies and thus a higher degree of compression is possible if this restriction of contiguity can be lifted. GCNS addresses both these issues: it restricts neither the number nor separation of the frequencies, and as expected, leads to higher compression.

Although there is no explicit importance given to simpler representations (lesser number of unique frequencies) in GCNS, the cut and splice operators coupled with the elitist nature of the algorithm ensure that genomes become longer only if required. Thus, if a particular problem does not allow high compression, GCNS will utilize more frequencies until the complexity required can be expressed. Further research in this direction is underway.

Acknowledgments. This research was funded by the following EU projects: IM-CLeVeR (FP7-ICT-IP-231722) and WAY (FP7-ICT-288551).

References

1. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with analog genetic encoding. In: PPSN, pp. 671–680 (2006)
2. Goldberg, D., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis, and first results. Complex systems 3(5), 493–530 (1989)
3. Goldberg, D.E., Deb, K., Kargupta, H., Harik, G.: Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In: Proc. of the Fifth International Conference on Genetic Algorithms, pp. 56–64. Morgan Kaufmann (1993)
4. Gomez, F., Koutník, J., Schmidhuber, J.: Compressed Network Complexity Search. In: Coello Coello, C.A., et al. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 316–326. Springer, Heidelberg (2012)
5. Gruau, F.: Cellular encoding of genetic neural networks. Technical Report RR-92-21, Ecole Normale Superieure de Lyon, Institut IMAG, Lyon, France (1992)
6. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. Complex Systems 4, 461–476 (1990)
7. Koutník, J., Gomez, F., Schmidhuber, J.: Evolving neural networks in compressed weight space. In: Proc. of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 619–626. ACM (2010)
8. Koutník, J., Gomez, F., Schmidhuber, J.: Searching for minimal neural networks in fourier space. In: Proc. of the 4th Conf. on Artificial General Intelligence (2010)
9. Schmidhuber, J.: Discovering neural nets with low Kolmogorov complexity and high generalization capability. Neural Networks 10(5), 857–873 (1997)
10. Yekutieli, Y., Sagiv-Zohar, R., Aharonov, R., Engel, Y., Hochner, B., Flash, T.: Dynamic model of the octopus arm. I. Biomechanics of the octopus reaching movement. Journal of Neurophysiology 94(2), 1443–1458 (2005)

Analyzing Module Usage in Grammatical Evolution

John Mark Swafford, Erik Hemberg, Michael O'Neill, and Anthony Brabazon

Natural Computing Research & Applications Group
Complex and Adaptive Systems Laboratory
University College Dublin, Ireland
john-mark.swafford@ucdconnect.ie
{erik.hemberg,m.oneill,anthony.brabazon}@ucd.ie

Abstract. Being able to exploit modularity in genetic programming (GP) is an open issue and a promising vein of research. Previous work has identified a variety of methods of finding and using modules, but little is reported on how the modules are being used in order to yield the observed performance gains. In this work, multiple methods for identifying modules are applied to some common, dynamic benchmark problems. Results show there is little difference in the performance of the approaches. However, trends in how modules are used and how "good" individuals use these modules are seen. These trends indicate that discovered modules can be used frequently and by good individuals. Further examination of the modules uncovers that useful as well as unhelpful modules are discovered and used frequently. The results suggest directions for future work in improving module manipulation via crossover and mutation and module usage in the population.

1 Introduction

A recent survey by O'Neill et al. [13], cites modularity as an important open topic in genetic programming (GP) [7]. Some of the earliest work in this area shows how GP representations which enable and/or exploit some form of modularity may outperform and scale better than standard GP on certain benchmark problems [8]. Since the early 1990s, numerous methods have been implemented with varying levels of success. While these methods can be valuable and yield significant performance improvements over standard GP, little has been reported on how the discovered modules are used and contribute to the population's fitness.

Studies of modularity in dynamic environments (also an open issue [13]) are also sparse. Dynamic environments provide a testbed that more resembles real-world problems which are rarely static, like most problems GP researchers tackle. Recent work by Kashtan et al. [6] and O'Neill et al. [11] shows dynamic environments which vary the fitness function over time can even speed up evolution. While dynamic environments is a large topic in GP, no work has been published, to the author's knowledge, examining how incorporating modularity changes GP's performance in dynamic environments.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 347–356, 2012.

The experiments carried out for this work use the popular grammar-based form of GP, grammatical evolution (GE) [12]. To study how enabling and exploiting different forms of modularity impact GE's search in dynamic environments, methods for identifying and making modules available to the population described by Swafford et al. [16] are used. For this work, modules are defined as encapsulated sub-derivation trees taken from the full derivation trees of GE individuals.

In the rest of this work, the effects of modules on GE's search will be presented. First, Sect. 2 describes some of the relevant previous work in dynamic environments and identifying and using modules. Next, Sect. 3 explains how modules are identified and made available for use by the population. Section 4 outlines the various experimental setups used. Following the explanation of the experimental design, Sect. 5 details the results of this study and discusses their meaning. Finally, Sect. 6 draws together some conclusions and proposes ideas for more future work.

2 Previous Work

To this day, there have been numerous approaches for identifying and using modules. Some of the best known of these are Koza's automatically defined functions [8], Angeline and Pollack's Genetic Library Builder [1], Rosca and Ballard's Adaptive Representation [14], and Walker and Miller's Embedded Cartesian GP [17]. Each of these are valuable for defining how modules may be identified and the benefits they give to the evolving population. However, little is said about the modules are actually used by the population. Harper and Blair [3] touch on this issue on their work with Dynamically Defined Functions (DDFs). They give an example how individuals which do not use any DDFs are quickly weeded out of the population and replaced with individuals using one or two DDFs. While this is useful, a more in-depth examination of how modules (in this case DDFs) are used could provide insight into how they could be better exploited to maximize the performance gain they provide. Miller et al. [9] also spend some time examining the frequency of use of certain sub-programs in their study of evolutionary design of digital circuits.

Little work has gone into understanding how modularity enhances or inhibits evolutionary search in dynamic environments. Some of the earliest approaches to modularity are tested on a dynamic problem (the Pac-man game) [8,14] and outperform standard GP on this problem. But no attention is given to how the addition of modules actually encourage the discovery of better solutions. More related work by Kashtan et al. [4,5,6] examine how evolution in dynamic environments can lead to more modular solutions and speed up evolution under certain conditions. They provide a useful analysis of their findings, but do not incorporate any mechanism for identifying and promoting the use of modules. For a more comprehensive survey of work in dynamic environments, see Dempsey et al. [2].

3 Module Identification Methods

Numerous methods for identifying modules and making them available to individuals during evolution have been developed by previous researchers. The experiments presented here use the methods described by Swafford et al. [15,16]. For the proceeding methods for discovering modules, a module is defined as encapsulated sub-derivation trees taken from the full derivation trees of GE individuals. These approaches for finding modules are briefly summarized as follows:

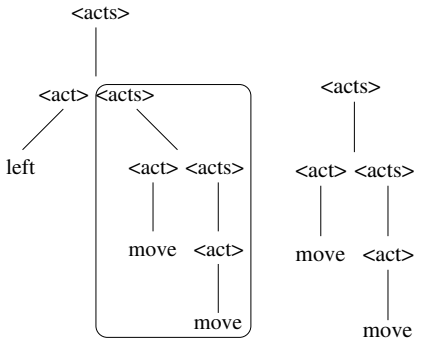
Mutation Identification (M-ID): An individual is taken from the population and a node on its derivation tree is randomly picked. This is the candidate module. It is replaced 50 times with randomly created sub-derivation trees of the same size. For each replacement, the entire individual is re-evaluated and the updated fitness is recorded. For each random sub-derivation tree inserted into the individual, the difference between the individual's original fitness and updated fitness is saved. If the original fitness is better than 75% of the updated fitness values, the candidate module is saved and the mean of the fitness differences is used as the module's fitness.

Insertion Identification (I-ID): First, 50 test individuals are generated using the same initialization method as the population. Next, the fitness of each is calculated. A candidate module is randomly picked from an individual and is inserted into each of the test individuals to replace a random sub-derivation tree with the same depth. Then, the test individuals are re-evaluated. When the candidate module is inserted into each of the test individuals, the difference between the original and updated fitness is saved. If the candidate module improves the fitness of 75% of the test individuals, it is saved and the mean of the fitness differences is used as the module's fitness.

Frequency Identification (F-ID): This method counts the occurrence of every sub-derivation tree in the population, except for single non-terminals. The most common sub-derivation trees are used as modules and given fitness values based on their frequency: $\frac{\#\text{ of occurrences}}{\text{total }\#\text{ of sub-trees}}$.

Random Identification (R-ID): A random sub-derivation tree is picked from each individual in the population and a module is created out of it. Because the module is not evaluated, the parent individual's fitness is used as the module's fitness.

For the following experiments modules are only selected from the top 15% of individuals. Once the modules have been identified a mechanism for making them available to the population is needed. In GE, adding them to the grammar used to create individuals in the population is a simple and effective method to resolve this issue. As with the methods for identifying modules, the manner in which modules are inserted into and removed from the grammar is also borrowed from Swafford et al. [16]. This is summarized in Fig. 1. The number of modules allowed in the grammar at one time has been limited to 20 based on results reported by Swafford et al. [15]. If more than 20 modules have been identified,



(a) A sample individual (b) A sample module

```
<acts> ::= <act> | <act> <acts>
<act>  ::= move | left | right
```

(c) Original Grammar

```
<acts>         ::= <act> | <act> <acts>
                 | <acts_mod_lib>
<act>          ::= move | left | right
<acts_mod_lib> ::= <mod_0>
<mod_0>        ::= move move
```

(d) Updated Grammar

Fig. 1. These figures show how the grammar is modified when a module is added to it

they are ranked based on their fitness values assigned upon their creation and the best 20 of these are kept. If a module is removed from the grammar, and individuals still use it, every occurrence of the module is expanded into the full sub-tree used to create that module. This prevents the phenotypes of individuals from changing when the grammar is modified.

4 Experimental Setup

This work examines the hypothesis that modules can be useful in dynamic environments as they may be able to find and encapsulate sub-solutions that are useful across multiple fitness scenarios. In order to do this, an easy, medium, and hard instance of each of the following common benchmark problems are used: Symbolic Regression (x^5-2x^3+x, $x^6-2x^4+x^2$, $x^7-2x^5+x^3$), Even Parity (7, 8, 9), and Lawn Mower (8×8, 12×12, 14×14). Parameters for specifying how often and the manner in which the fitness function changes are borrowed from Murphy et al. [10]. The number of generations between fitness function changes are 5 and 20. Each of these period lengths was used with random and cyclic changes. Modules identification also occurs every 5 and 20 generations, meaning modules are identified at the beginning of each fitness period. However, this does not allow time for evolution to adjust to the new fitness function before modules are selected from the population. In response to this, staggered module identification steps are also used. Instead of searching for modules at the beginning of each fitness period, the initial module identification starts 2 generations into the first fitness period and then continues every 5. A similar staggered approach starting at generation 10 and continuing every 20 generations is also employed. More frequent and random changes in the fitness function mean GE has little time to adjust to the new environment. Longer fitness periods allow GE more time to adjust to the new target. This variety of fitness and module identification steps allows for testing how well the discovered modules helps GE recover from changes in the target functions.

5 Results and Discussion

This section answers the question of how modules alter (or not) the behavior of GE in dynamic environments. First, modules' frequency, lifetimes, and the fitness of individuals using them are examined. Hand picked modules are also shown for the purpose of understanding what kinds of modules are being discovered and used. Then, a short explanation of how the various methods for identifying modules helps or hinders GE's search capabilities in the dynamic environments.

5.1 Frequency and Lifetime of Modules in the Population

To begin the analysis of the various modular approaches, the frequency of module usage in the population is shown in a heatmap in Fig. 2. This reveals trends in how many individuals are using modules and how long modules are used. This figure shows the results of a Symbolic Regression instance where the fitness function target changes randomly every 20 generations and modules are identified every 20 generations starting at generation 10. Modules are identified using the M-ID approach. Out of the 1519 modules discovered across all 50 runs of this variation, only 442 modules are present in the heatmap. These modules appear in at least 50% of the population. Out of all the modules in Fig. 2 only a small portion of the modules identified in early generations are heavily present throughout multiple module identification and replacements. This suggests these modules contain information that is useful in all of the fitness instances of this particular problem. Through crossover, mutation, and selection operations, some of these modules are able to move from being used by less then 10% of the population to over 90%. Figure 2 also shows that the majority of modules are identified, used frequently, and then are used rarely, if at all, after one or two fitness periods. It also shows a scattered few modules are being used by a large portion of the population for only 5–25 generations before their usage plummets. This indicates that being used by a large percentage of the population does not ensure longevity across many generations.

Only examining the frequency and longevity of use of modules does not paint a full picture of how they are used. To further understand this, Fig. 3 shows the average fitness of every individual each module appears in. The modules in Fig. 3 and Fig. 2 are the same. An interesting characteristic is the similarity between Fig. 2 and Fig. 3. This similarity shows a strong correlation between modules getting used frequently and modules being used by highly fit individuals. A comparison of Figs. 2 and 3 also shows that better individuals are often the only individuals that use modules for long periods of time before and after they are used more widely in the population. Another notable trait of Fig. 3 is the darker colored cells immediately after the fitness period changes. This signifies a drop in the population's fitness as they are adjusting to a new target.

However, Figs. 2 and 3 only show results for a single variation on one problem. Changing the length of the fitness periods and frequency of identifying modules yields no significant changes in the correlations noted above. Using the R-ID and F-ID module identification methods also show similar correlations between

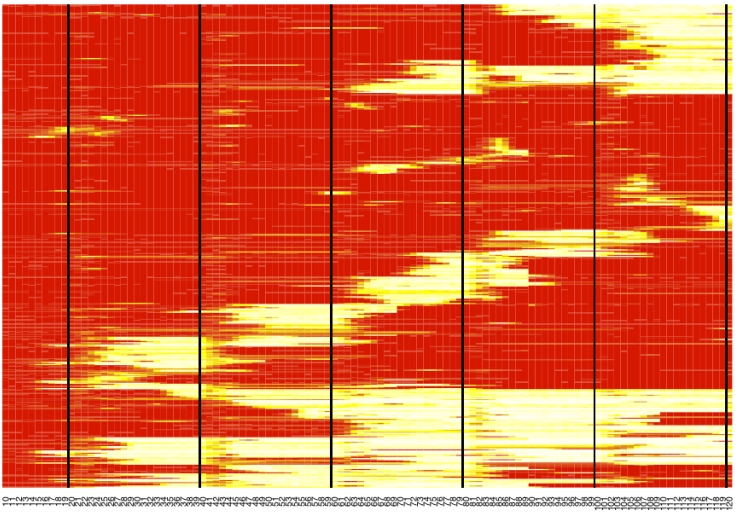


Fig. 2. This figure show module usage across all runs on the Symbolic Regression problem using the M-ID approach, identifying modules every 20 generations starting from generation 10 with the fitness function changing randomly every 20 generations. Each row (the y axis) represents a module that is used by at least 50% of the population at any point in a run and each column (the x axis) corresponds to a single generation. The cells represent the percentage of the population that uses the module in each generation. The black vertical lines indicate generations when the fitness function changes. Dark red colored cells indicate that modules are not being used by any individuals. White colors denote that a module is used by a large percentage or all of the population. The modules have been clustered together with modules that were used similarly to make the graph easier to read. Generations 0–9 have been omitted because module identification has not yet occurred in those generations.

the frequency of module usage and fitness of individuals using those modules. The exception to this is the insertion (I-ID) method for finding modules. It finds very few modules in general. Many runs using the I-ID method find very few modules, and even fewer of these are used largely by the population. The small number of modules found by I-ID is due to the fact that it requires modules to be beneficial in multiple individuals, not only one. In many instances, no modules are used by more than 50% of the population. This suggests that this approach is inappropriate for the instances of the Symbolic Regression problem examined.

Analyzing the Even Parity problem in the same manner, Figs. 4(a) and 4(b) show similar behavior to the Symbolic Regression problem. Modules that are used more frequently tend to also be used in individuals with better fitness. But there are also modules being used in good individuals and few other individuals in the population. Another similarity is that the I-ID approach finds remarkably fewer modules than M-ID, R-ID, and F-ID. In the case of the Lawn Mower problem, Figs. 4(c) and 4(d) tell a different story. Any correlation between how frequently the modules are used and the fitness of individuals using them appears to be absent. Figure 4(c) shows modules being used by both large and small

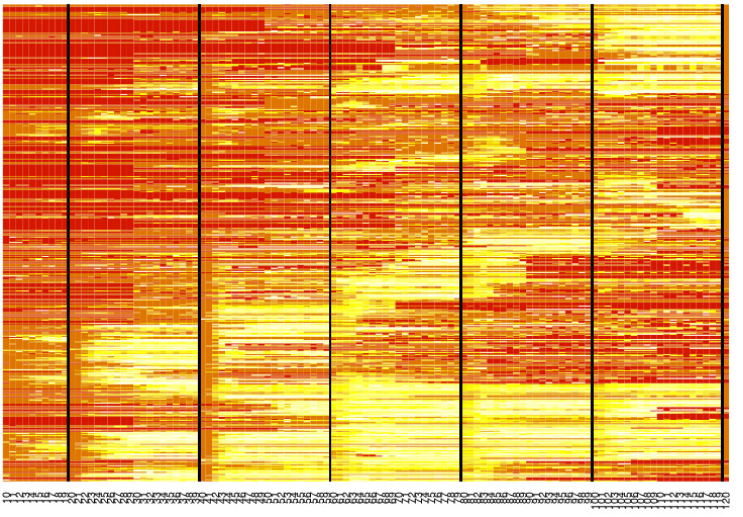


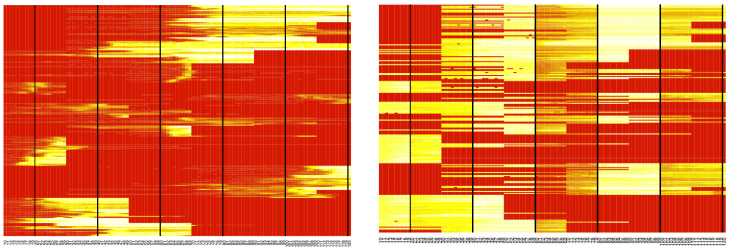
Fig. 3. This figure shows the average fitness of individuals that use a given module. The module identification and fitness period parameters are used in this figure are the same those in Fig. 2. Fitness values have all been normalized between 0 and 1. White and pale yellow colored cells represent modules being used by individuals with good fitness values. Orange cells mean modules get used in individuals with worse fitness. Red cells indicate that modules are not used at all at that generation.

percentages of the population. But the same modules in Fig. 4 are constantly used by the best individuals in the population, regardless of how frequently they are used. The most likely reason for this is the nature of the Lawn Mower problem itself. As modules are able to encapsulate multiple mowing instructions into a single production, individuals using modules can more easily cover larger areas of the lawn than individuals that must combine single terminal symbols to cover the same area. Taking this under consideration, it is easy to understand why modules are frequently being used by the best individuals.

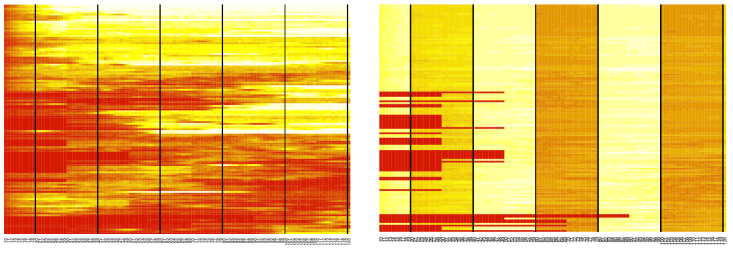
Examining how modules are used in this way naturally leads to two questions:

1. What kinds of modules are being discovered?
2. Do the modules being discovered and used improve GE's fitness?

To answer the first question, a small selection of modules from the Symbolic Regression runs shown above can be examined. The following modules are described in detail because their clear utility and their longevity of use. The first two modules to be examined are `*--xx+11/*1x/11` and `*+11x`, which simplify to $-2x$ and $2x$ respectively. These two modules can be discussed together as their stories are similar and it is easy to see how they could potentially be used as building blocks for the target solutions. Both of these modules were discovered at generation 10. Neither of these modules are immediately adopted by large amounts of individuals. For many generations, they fluctuate in usage percentage from at 0.008% to 70% of the population. By generation 28, both modules



(a) Parity - Usage (b) Parity - Fitness



(c) Lawn Mower - Usage (d) Lawn Mower - Fitness

Fig. 4. This figure shows how frequently modules are used and the average fitness of individuals modules are used in for the Even Parity and Lawn Mower problems. These problem instances use the same parameters as Figs. 2 and 3

are being used by at least 90% of the population. They also experience lulls in usage at different generations where they are used by as little as 5% or 10% of individuals before becoming more prominent in the population again. A third module, `*xx`, or x^2, was also discovered and used in a similar manner. The difference in this particular module is that it never experiences the drop in usage the others do. Once more than 90% of the population uses this module, never again do less than 94% of individuals use it.

When examining the modules used by large portions of the population, another notable trend was seen. Many popular modules simply reduced to 0. Another observation about the modules discovered is that some of the frequently used ones have no apparent usefulness. A possibility for this is that they do not damage or change the fitness of individuals. More investigation is needed to give a definitive reason why these modules are used frequently. A possible cause of this is that it is very difficult to find good modules and the module identification methods sometimes find bad modules.

5.2 Modularity and Fitness

Different approaches to modularity lead to differences in GE's performance. These approaches are compared using the metrics presented by Murphy et al. [10] are used (*draw down*, *area under the curve*, and *fall off*). The methods for identifying modules from Sect. 3 are compared to standard GE and GE with ADFs. The observed data shows that among the methods examined, there is no single

best approach on any of the problems except the Lawn Mower, where GE with ADFs is the best performing approach. As seen in Sect. 5.1, modules are being identified and often used. This suggests that even though modules are being used frequently, they are being used sub-optimally or are not good modules.

6 Conclusion and Future Work

This work examined the effects of incorporating four approaches to modularity on three common benchmark problems in GP: Symbolic Regression, Lawn Mower, and Even Parity. Each of these problems was studied as a dynamic problem, where targets of increasing levels of difficulty were used. The results observed show no large difference between any of the approaches in terms of increasing performance over standard GE. However, the Symbolic Regression and Even Parity instances did show interesting trends in how often modules were used by the population and the fitness of individuals using those modules. The data suggests that modules used by a large percentage of the population also tend to be used by individuals with high fitness. Further examining a selection of the modules being used frequently also shows how potentially helpful modules are being found, used frequently, and used by highly fit individuals. On the other hand, a number of useless modules are being used in the same way. These results point towards a number of possibilities for future work.

One potentially extension from this work would be ensuring that modules are being used in contexts where they can be the most useful. Two of the approaches for identifying modules estimate how well sub-derivation trees perform in particular contexts. If those sub-derivation trees become modules and are used in other contexts, they may be harmful instead of helpful. When estimating the worth of sub-derivation trees, many are passed over as they are deemed unworthy of becoming a module. These may even be harmful sub-derivation trees. Another interesting vein of research could be considering these "bad" sub-derivation trees as anti or taboo modules. It may be beneficial to guide search away from these structures as they are not considered to contain helpful information.

Acknowledgments. This work is funded by the University College Dublin School of Computer Science & Informatics and Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References

1. Angeline, P.J., Pollack, J.B.: The evolutionary induction of subroutines. In: Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, pp. 236–241. Lawrence Erlbaum, Bloomington (1992)
2. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations in Grammatical Evolution for Dynamic Environments. Springer (2009)
3. Harper, R., Blair, A.: Dynamically defined functions in grammatical evolution. In: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, pp. 9188–9195. IEEE Press, Vancouver (2006)

4. Kashtan, N., Parter, M., Dekel, E., Mayo, A.E., Alon, U.: Extinctions in heterogeneous environments and the evolution of modularity. Evolution 63, 1964–1975 (2009)
5. Kashtan, N., Mayo, A.E., Kalisky, T., Alon, U.: An analytically solvable model for rapid evolution of modular structure. PLoS Comput. Biol. 5(4), e1000355 (2009)
6. Kashtan, N., Noor, E., Alon, U.: Varying environments can speed up evolution. Proceedings of the National Academy of Sciences 104(34), 13711–13716 (2007)
7. Koza, J.R.: Genetic Programming: on the Programming of Computers by Means of Natural Selection. MIT Press (1992)
8. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge (1994)
9. Miller, J.F., Job, D., Vassilev, V.K.: Principles in the evolutionary design of digital circuitspart ii. Genetic Programming and Evolvable Machines 1, 259–288 (2000), doi:10.1023/A:1010066330916
10. Murphy, E., O'Neill, M., Brabazon, A.: A comparison of ge and tage in dynamic environments. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1387–1394. ACM, New York (2011)
11. O'Neill, M., Nicolau, M., Brabazon, A.: Dynamic environments can speed up evolution with genetic programming. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO 2011, pp. 191–192. ACM, New York (2011)
12. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers (2003)
13. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genetic Programming and Evolvable Machines 11, 339–363 (2010), doi:10.1007/s10710-010-9113-2
14. Rosca, J.P., Ballard, D.H.: Discovery of subroutines in genetic programming, pp. 177–201. MIT Press, Cambridge (1996)
15. Swafford, J.M., Hemberg, E., O'Neill, M., Nicolau, M., Brabazon, A.: A non-destructive grammar modification approach to modularity in grammatical evolution. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1411–1418. ACM, Dublin (2011)
16. Swafford, J.M., Nicolau, M., Hemberg, E., O'Neill, M., Brabazon, A.: Comparing methods for module identification in grammatical evolution. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO 2012. ACM, Philadelphia (2012)
17. Walker, J., Miller, J.: The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. IEEE Transactions on Evolutionary Computation 12(4), 397–417 (2008)

On the Anytime Behavior of IPOP-CMA-ES

Manuel López-Ibáñez, Tianjun Liao, and Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
{manuel.lopez-ibanez,tliao,stuetzle}@ulb.ac.be

Abstract. Anytime algorithms aim to produce a high-quality solution for any termination criterion. A recent proposal is to improve automatically the anytime behavior of single-objective optimization algorithms by incorporating the hypervolume, a well-known quality measure in multi-objective optimization, into an automatic configuration tool. In this paper, we show that the anytime behavior of IPOP-CMA-ES can be significantly improved with respect to its default parameters by applying this method. We also show that tuning IPOP-CMA-ES with respect to the final quality obtained after a large termination criterion leads to better results at that particular termination criterion, but worsens the performance of IPOP-CMA-ES when stopped earlier. The main conclusion is that IPOP-CMA-ES should be tuned with respect to the anytime behavior if the exact termination criterion is not known in advance.

Keywords: Anytime algorithms, automatic parameter tuning, continuous optimization.

1 Introduction

In many practical situations, an optimization algorithm may be terminated at any time, and, hence, it should return as high-quality solutions as possible for a wide range of possible termination criteria. Algorithms that better satisfy this property are said to have better anytime behavior [14].

When designing a new algorithm or tuning its parameters, the classical way to assess its anytime behavior is either by comparing plots of the solution-quality over time, called SQT curves [5], or by measuring performance at a different number of targets, for example, measuring solution quality after a given number of function evaluations. The benefit of the graphical comparison of SQT plots is that one gets the whole picture and it is less biased by the choice of the targets. However, a graphical comparison is intrinsically subjective. In contrast, measuring performance at different targets is an objective comparison. However, one still needs to aggregate the possibly conflicting results for each target in order to compare multiple algorithms. In this paper, we apply a new alternative, which consists in evaluating the anytime behavior as a bi-objective optimization problem. In particular, the hypervolume, a well-known quality measure in multi-objective optimization, may be used to assign a single numerical value to the anytime behavior of an algorithm's run. This technique allows us to apply automatic configuration tools for automatically improving the anytime behavior of optimization algorithms.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 357–366, 2012.

CMA-ES [3] is a state-of-the-art algorithm for continuous optimization. Recently, a variant of CMA-ES with incremental population has been proposed [1]; we refer to this variant as IPOP-CMA-ES. The authors of IPOP-CMA-ES show that it outperforms the classical CMA-ES with restarts that keeps the population size fixed for a wide range of functions and allocated number of function evaluations. Therefore, one can say that IPOP-CMA-ES shows already a good anytime behavior. In this paper, we show that the anytime behavior of IPOP-CMA-ES can be further improved by combining automatic algorithm configuration tools and the hypervolume measure. Moreover, we also report results on tuning IPOP-CMA-ES for a specific termination criterion. The resulting configuration of IPOP-CMA-ES obtains better final quality than the default configuration and the configuration tuned for anytime behavior, but performs substantially worse if interrupted earlier than the specific termination criterion. Therefore, our results indicate that if the specific termination criterion is not known in advance or there is a high chance that IPOP-CMA-ES may be interrupted earlier, then it is better to tune IPOP-CMA-ES according to anytime behavior rather than using the default settings or tuning for a specific termination criterion.

2 Anytime Optimization

Anytime optimization algorithms may be terminated at any moment during their run, and they return a solution that is closer to the optimal the more time they were allowed to run [14]. In fact, most stochastic local search algorithms match this definition. The ideal anytime optimization algorithm would return a solution as close as possible to the optimal at any moment during its run. Hence, algorithms closer to this ideal have better anytime behavior.

One of the goals of adapting parameter settings at run-time is to adapt the exploration and exploitation trade-off to the amount of computation time allowed. Such algorithms converge very quickly to a good solution or local optimum, and then, if more time is allowed, explore more thoroughly the search space to find better solutions. Although algorithms that adapt their parameters, such as IPOP-CMA-ES, purport to remove the need to tune the parameters that are adapted, the adaptation methods introduce parameters that are subject to fine-tuning. In fact, it has been shown that automatically tuning these parameters may considerably improve the final quality obtained by IPOP-CMA-ES on diverse and difficult benchmarks [6, 7, 12]. One may argue, however, that this fine-tuning probably makes the algorithm more dependent on the particular termination criterion used in the tuning, in other words, it worsens its anytime behavior. Hence, it would be desirable to fine-tune the parameters of such algorithms in a way that is not specific to a particular termination criterion.

3 Automatically Improving Anytime Behavior

The anytime behavior of an algorithm may be modeled as a bi-objective optimization problem in terms of Pareto-optimality. In this model, the output of a

run of an algorithm is a set of points in the *time* × *quality* space representing every instant that the algorithm found a solution closer to the optimal. This set of points is by definition mutually nondominated, that is, there is no point in the set that is better than another point in one criterion and not worse in the other. According to this model, we can say that a run of algorithm A has a better anytime behavior than a run of algorithm B, if the output of A is better than the output of B in the Pareto sense, that is, if all points from B are dominated by at least one point from A, and there is no point from A that is dominated by a point from B. In practice, the SQT curves of high-performing algorithms will often cross, and, hence, their outputs are often incomparable in the strict Pareto sense. This is a usual case in multi-objective optimization, and, frequently, unary quality measures are used to compare nondominated sets. Among the quality measures available, the hypervolume is the only one always able to detect whether one nondominated set is not worse than another [15]. When all objectives are minimized, the hypervolume of a nondominated set is the area of the objective space that is bounded below by the set and above by a reference point that should be the same for all sets under comparison. Thus, a larger hypervolume corresponds to a better quality.

Using the above model, López-Ibáñez and Stützle [11] have proposed to integrate the hypervolume into an automatic configuration tool in order to automatically improve the anytime behavior of optimization algorithms. We show here that this technique is able to significantly improve the anytime behavior of IPOP-CMA-ES with respect to its default settings.

4 Experimental Setup

In this paper, we try to automatically improve the anytime behavior of IPOP-CMA-ES. IPOP-CMA-ES is (μ, λ)-evolution strategy that samples a new population of solutions at each iteration from a multi-variate normal distribution. The parameters of this normal distribution are adapted during the run of the algorithm in order to focus the sampling on the most promising region of the search space. IPOP-CMA-ES obtained the best performance in the special session on real parameter optimization of the 2005 IEEE Congress on Evolutionary Computation (CEC'05), and, thus, it is a state-of-the-art algorithm for continuous optimization. In our experiments, we use the C version of IPOP-CMA-ES from Hansen's webpage `http://www.lri.fr/~hansen/cmaesintro.html`. We have modified the code to handle bound constraints by clamping the variable values outside the bounds on the nearest bound value [7].

There are a number of internal parameters of IPOP-CMA-ES that are fixed in the default implementation. These are the initial population size λ_0, the number of parent solutions selected from the population μ, and the initial step-size σ_0 among others. The population size is multiplied by a factor (*ipop*) every time the algorithm is restarted. Restarts are controlled by three additional parameters: *stopTolFunHist*, which is a lower threshold on the range of the best objective function values in recent generations; *stopTolFun*, which is a lower threshold that,

Table 1. Parameters that have been considered for tuning. Given are the default values of the parameters and the continuous range we considered for tuning. The last two columns are the parameter settings obtained for the anytime tuning (tanytime) and the tuning for the final solution quality (tfinal), respectively.

Parameter (tuning)	Internal parameter	Default	Range	Tuned	
				tanytime	tfinal
a	Init pop size: $\lambda_0 = 4 + \lfloor a \ln(D) \rfloor$	3	$[1, 10]$	3.676	9.600
b	Parent size: $\mu = \lfloor \lambda / b \rfloor$	2	$[1, 5]$	1.750	1.452
c	Init step size: $\sigma_0 = c \cdot (B - A)$	0.5	$(0, 1)$	0.325	0.603
d	IPOP factor: *ipop* $= d$	2	$[1, 4]$	1.840	3.292
e	*stopTolFun* $= 10^e$	-12	$[-20, -6]$	-9.653	-8.854
f	*stopTolFunHist* $= 10^f$	-20	$[-20, -6]$	-10.000	-9.683
g	*stopTolX* $= 10^g$	-12	$[-20, -6]$	-9.528	-12.550

in addition to the previous range, also includes all objective function values in the last generation; and *stopTolX*, which is a lower threshold on the standard deviation of the normal distribution.

For tuning IPOP-CMA-ES, we have exposed seven parameters that directly control the internal parameters of IPOP-CMA-ES defined above. These seven parameters are given in Table 1, together with the internal parameter of IPOP-CMA-ES controlled by each of them, their default value and the range considered here for tuning. As tuner we use `irace` [9], a publicly available implementation of the automatic configuration method Iterated F-Race [2]. The budget of each run of `irace` is set to 5 000 runs of IPOP-CMA-ES. The other inputs of `irace` are the parameter ranges given in Table 1 and a set of training instances.

As benchmark instances, we consider the 19 functions from the SOCO benchmark set [4] and the 25 functions from the CEC'05 benchmark set [13]. In order to avoid over-tuning, the training set of instances used for tuning is different from the test sets used for analyzing the results of the tuning. Training instances are a subset of the functions in the SOCO benchmark, with dimension $D \in [5, 40]$. The training functions are then sampled in a random order from all possible such functions [8]. For analyzing the results, we use three test sets: 19 SOCO benchmark functions, but the 10-dimensional (SOCO-10D) and the 100-dimensional (SOCO-100D) versions, and the CEC benchmark functions with dimension 50 (CEC-50D).

We follow the protocols suggested by the authors of the SOCO and CEC benchmarks [4, 13], that is, the maximum number of function evaluations is $5\,000 \cdot D$ for the SOCO functions and $10\,000 \cdot D$ for the CEC functions. Each run of IPOP-CMA-ES is repeated 25 times on each function with different random seed. We report error values defined as $f(\boldsymbol{x}) - f(\boldsymbol{x}^*)$, where $\boldsymbol{x}$ is a candidate solution and $\boldsymbol{x}^*$ is the optimal solution. Following the recommendation of the authors of the CEC benchmark, we use 10^{-8} as the minimum error (zero threshold), and lower values are clamped to this minimum.

5 Experimental Results

We automatically configure the parameters of IPOP-CMA-ES according to anytime behavior. For each run of IPOP-CMA-ES, every time a solution better than the best of the current run is found, we record the number of function evaluations (FEs) performed so far and the quality of the new best solution. In this manner, each run produces a nondominated set of points of quality versus FEs. We restrict the minimum number of FEs to D, that is, we start recording the solution quality after D FEs, in order to avoid the bias of the initial random sampling of IPOP-CMA-ES. All nondominated sets under comparison for the same benchmark function are normalized to the interval $[1, 2]$. Then, we compute the hypervolume of the normalized nondominated sets using $(2.1, 2.1)$ as the reference point. We integrate this procedure into `irace`, and use the hypervolume to evaluate each run of IPOP-CMA-ES. In this manner, we obtain a configuration of IPOP-CMA-ES called henceforth tanytime (Table 1).

Next, we run both tanytime and the default configuration of IPOP-CMA-ES (henceforth, default) on each benchmark function of the three test sets. Each run is repeated 25 times with different random seed. We compute the mean hypervolume of these runs using the same procedure described above. The results reported in Table 2 show that the tuning works, that is, the tanytime configuration obtains better (larger) hypervolume values than default in most functions, even when testing on functions with different dimensionality or from a different benchmark set. Nonetheless, it was not possible to improve the hypervolume on all functions at the same time with a single parameter setting. We performed a two-sided Wilcoxon matched-pairs signed-rank test at the 0.05 α-level, which indicates that the differences in favor of tanytime are significant in each of the three test sets. Therefore, we have found a configuration of IPOP-CMA-ES with better anytime behavior according to the hypervolume.

We assess how much this improvement is visible when evaluating the anytime behavior according to SQT curves, computed as mean error value versus FEs. Figure 1 shows the mean SQT curves, where error values are averaged over 25 runs, on a few test functions of two configurations of IPOP-CMA-ES: default and tanytime. Both axes are in logarithmic scale. Other plots are available as supplementary material [10]. The first observation is that for those functions where the SQT curve of tanytime is clearly better than the one corresponding to default, the hypervolume of tanytime is always higher, which confirms the numerical results. In the few cases where default has a higher hypervolume than tanytime, the SQT curves look like the two plots in the right column of Fig. 1.

Next, we analyze the overall quality reached at a number of termination criteria. We define termination criteria FE_1, FE_2, FE_3, FE_4, and FE_5, which correspond, respectively, to $\{1D, 10D, 100D, 1000D, 5000D\}$ FEs for SOCO functions and $\{2D, 20D, 200D, 2000D, 10000D\}$ FEs for CEC functions. In order to measure the overall quality, we need to summarize error values from different benchmark functions, but the range and distribution of error values varies extremely from function to function. Depending on the scenario, one could assume that the error values are comparable, and compute summary statistics directly

Table 2. Hypervolume values on the SOCO benchmark functions of dimensions 10 (10D) and 100 (100D), and on the CEC'05 benchmark functions of dimensions 50 (50D). Each number is the mean hypervolume over 25 runs.

	SOCO benchmark			
	10D		100D	
Funs	default	tanytime	default	tanytime
f_{soco1}	1.198402	1.199056	1.185412	1.190244
f_{soco2}	1.187217	1.186541	1.195947	1.197192
f_{soco3}	1.209725	1.209742	1.209607	1.209758
f_{soco4}	1.194014	1.19501	1.171771	1.18795
f_{soco5}	1.208867	1.208925	1.204043	1.20496
f_{soco6}	1.129921	1.125299	1.115378	1.115365
f_{soco7}	1.209906	1.209913	1.209999	1.21
f_{soco8}	1.206128	1.206206	1.200541	1.200248
f_{soco9}	1.129397	1.129854	1.091057	1.113938
f_{soco10}	1.206764	1.207165	1.208304	1.208677
f_{soco11}	1.186655	1.201123	1.181395	1.195451
f_{soco12}	1.209611	1.209574	1.209227	1.209338
f_{soco13}	1.20989	1.209898	1.209687	1.209814
f_{soco14}	1.193432	1.193597	1.15417	1.166172
f_{soco15}	1.209983	1.209996	1.209998	1.21
f_{soco16}	1.209613	1.209635	1.209009	1.209123
f_{soco17}	1.209997	1.209987	1.209807	1.209899
f_{soco18}	1.208043	1.20837	1.207676	1.20822
f_{soco19}	1.208811	1.209075	1.209992	1.21
Num of best	4	**15**	2	**17**

	CEC (50D)	
Funs	default	tanytime
f_{CEC1}	1.190977	1.193059
f_{CEC2}	1.199888	1.199974
f_{CEC3}	1.208823	1.208888
f_{CEC4}	1.196578	1.202285
f_{CEC5}	1.198765	1.201426
f_{CEC6}	1.209755	1.209822
f_{CEC7}	1.207348	1.206793
f_{CEC8}	0.545673	0.577091
f_{CEC9}	1.193229	1.196095
f_{CEC10}	1.199903	1.201255
f_{CEC11}	1.180708	1.182876
f_{CEC12}	1.205898	1.207761
f_{CEC13}	1.209852	1.209925
f_{CEC14}	0.610495	0.713652
f_{CEC15}	1.153601	1.154691
f_{CEC16}	1.191556	1.190394
f_{CEC17}	1.132624	1.111859
f_{CEC18}	1.155031	1.15255
f_{CEC19}	1.147348	1.147457
f_{CEC20}	1.181023	1.182817
f_{CEC21}	1.078023	1.120791
f_{CEC22}	1.183588	1.185362
f_{CEC23}	1.050566	1.111973
f_{CEC24}	1.206570	1.206478
f_{CEC25}	1.208004	1.207968
Num of best	6	**19**

on the error values, or analyze how many runs achieve a particular error value. Instead, we consider that the error values of different functions are not directly comparable, and we use a non-parametric approach based on blocking, that is, algorithm runs are ranked per function with respect to the error value, and we compute the mean rank over all test functions in each benchmark set. Fig. 2 shows the mean rank, at each termination criterion, of the default configuration and the tanytime configuration. The other configurations shown will be explained later. The plots show that tanytime configuration ranks better than the default configuration for almost all termination criteria in all benchmark sets.

Anytime Behavior vs. Final Quality. Now we consider the possibility that the default parameters of IPOP-CMA-ES may not be the best for the benchmark sets and the maximum number of function evaluations considered here. Therefore, we tune the parameters of IPOP-CMA-ES according to the final quality

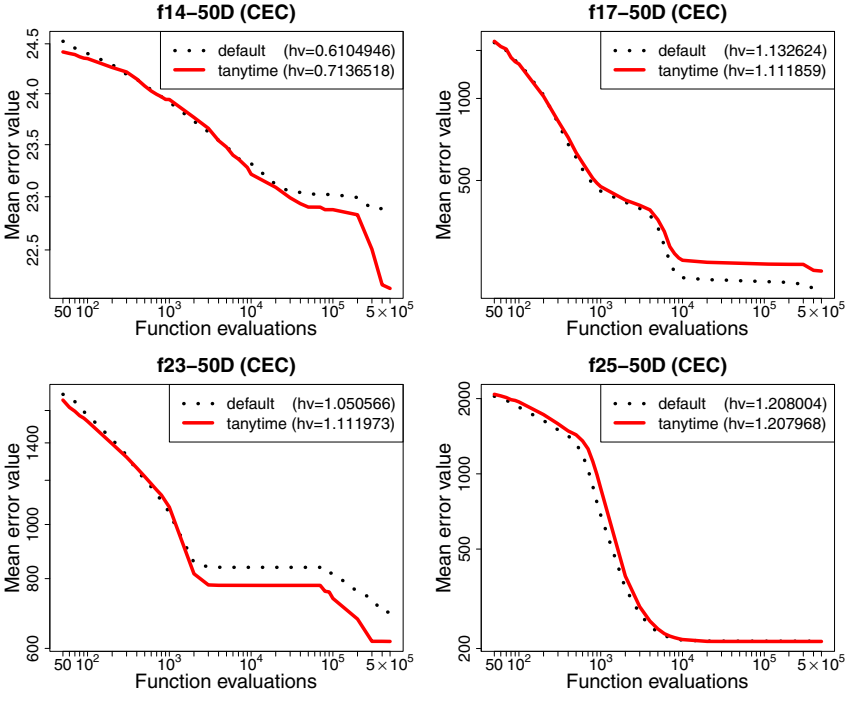


Fig. 1. SQT curves for two configurations of IPOP-CMA-ES. Plots on the left (right) show cases where tanytime obtains a better (worse) mean hypervolume than default.

obtained at the end of the run. In this way, we obtain a configuration that we call tfinal. We run this configuration on the test benchmark sets, and report the results in Fig. 2. The plots show that tfinal is able to obtain better final quality than both tanytime and default, but at the cost of worse anytime behavior. We carry out a Friedman test at each termination criterion to test for the significance of the differences between the best ranked configuration and the other two configurations. Here, we only report the results of the Friedman tests over all benchmark functions (Table 3); results per benchmark set are given as supplementary material [10]. The Friedman tests confirm these observations, that is, tfinal becomes much worse than tanytime and default if stopped earlier (termination criteria FE_1, FE_2, FE_3) than the termination criterion that was used for tuning (FE_5). The other configurations shown in the plots (and in Table 3) are explained in the next paragraph. The fact that there is a strong trade-off between final quality and anytime behavior suggests that there are still opportunities for improving the balance between fast convergence and exploration in IPOP-CMA-ES.

Hypervolume Applied to Logarithmic Transformations. The plots in Fig. 1 use a logarithmic scale in both axes, as usually done when comparing continuous optimizers. Yet, we compute the hypervolume on a linear scale, as

Table 3. Configurations of IPOP-CMA-ES ordered according to the sum of ranks obtained at each termination criterion FE_i. The numbers in parenthesis are the difference of ranks relative to the best configuration. ΔR_α is the minimum significant difference according to the Friedman test at significance level $\alpha = 0.05$. Configurations that are not significantly different from the best one are indicated in bold face.

All 63 functions (SOCO-10D, SOCO-100D, CEC-50D)		
FEs	ΔR_α	**Configurations** (ΔR)
FE1	27.34	**tany-lx-y (0),** tanytime (35.5), tany-lx-ly (38), default (116.5), tany-x-ly (158), tfinal (222)
FE2	20.39	**tany-lx-y (0),** tanytime (72.5), tany-lx-ly (78), default (98.5), tany-x-ly (214.5), tfinal (271.5)
FE3	38.32	**tanytime (0), default (26), tany-lx-ly (32), tany-lx-y (37),** tany-x-ly (57.5), tfinal (120.5)
FE4	32.28	**tfinal (0), tany-x-ly (12),** tanytime (67.5), tany-lx-ly (75.5), tany-lx-y (115.5), default (125.5)
FE5	30.51	**tfinal (0), tany-x-ly (21),** tanytime (38.5), tany-lx-ly (77.5), tany-lx-y (86), default (101)

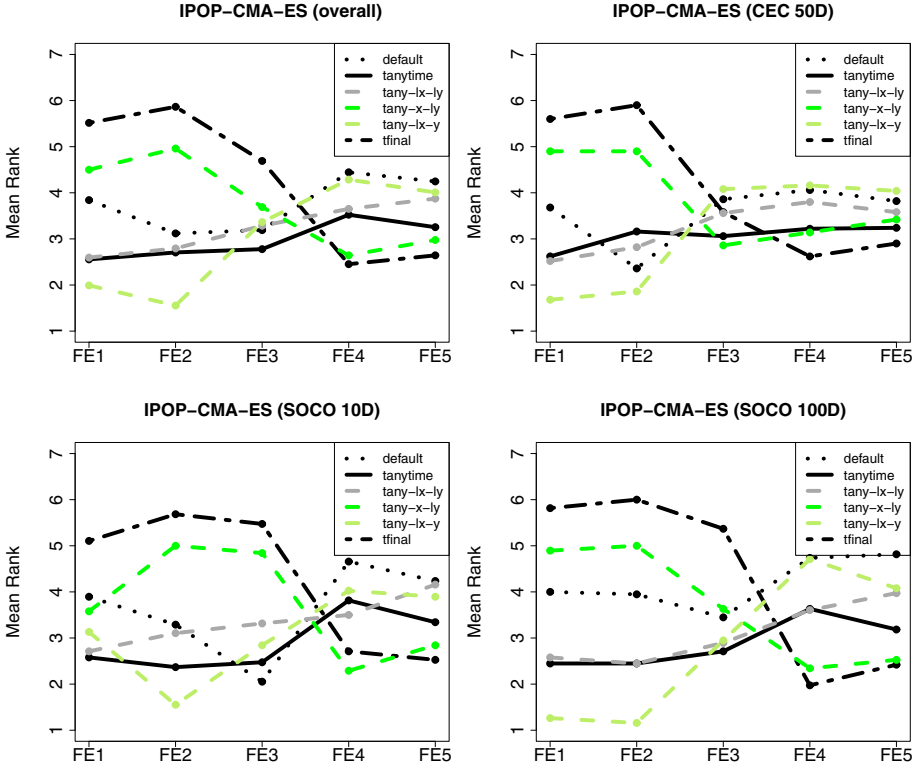


Fig. 2. Mean ranks obtained by configurations default, tanytime and tfinal at each termination criterion (FE_1 to FE_5)

commonly done in multi-objective optimization. Nonetheless, we can also apply a logarithmic scale for FEs, error values or both, before computing the hypervolume. Such transformations define a particular preference among otherwise incomparable nondominated sets, and, hence, lead to different anytime behaviors.

Fig. 2 provides an overall comparison of these alternatives (individual SQT plots are available as supplementary material [10]). Three additional configurations of IPOP-CMA-ES were obtained by tuning as described above, but using a modified hypervolume where either the number of FEs (tany-lx-y), the error values (tany-x-ly), or both (tany-lx-ly) were converted to a logarithmic scale. The plot shows that the configuration tany-lx-y (log. FE) performs better for short termination criteria, whereas the tany-x-ly (log. error values) obtains better results when running for longer FEs. Interestingly, there is no much difference between applying a logarithmic transformation to both objectives or to none of them. Our conclusion is that logarithmic transformations of only one objective (either quality or computational effort) introduce a strong bias, which should be taken into account to not defeat the purpose of tuning for anytime behavior.

6 Conclusions

In this paper, we have investigated whether the anytime behavior of IPOP-CMA-ES can be improved by automatically tuning its parameters. We have applied a recently proposed technique that integrates the hypervolume quality measure into an automatic configuration method (`irace`). Our results have shown that the anytime behavior of the default parameters of IPOP-CMA-ES can be substantially improved. Moreover, we have also shown that simply tuning IPOP-CMA-ES according to the quality achieved at a large termination criterion does improve the results at that particular termination criterion; however, it compromises the results for shorter termination criteria, becoming even worse than the default configuration of IPOP-CMA-ES. Therefore, if the specific termination criterion is not known in advance, it is better to tune IPOP-CMA-ES according to anytime behavior than for a very large termination criterion.

Our results also suggest that, despite the adaptation of the population size and the restart step in IPOP-CMA-ES, its results are not ideal in terms of anytime behavior. Therefore, we plan to investigate in the future whether the anytime behavior of IPOP-CMA-ES can be further improved by adapting other parameters or making some parameters time-varying.

Acknowledgments. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Manuel López-Ibáñez and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a postdoctoral researcher and a research associate, respectively. Tianjun Liao acknowledges a fellowship from the China Scholarship Council. The authors also acknowledge support from the FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*". The authors have contributed equally to this work.

References

1. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. In: Proc. of CEC 2005, pp. 1769–1776. IEEE Press, Piscataway (2005)
2. Balaprakash, P., Birattari, M., Stützle, T.: Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement. In: Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HM 2007. LNCS, vol. 4771, pp. 108–122. Springer, Heidelberg (2007)
3. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
4. Herrera, F., Lozano, M., Molina, D.: Test suite for the special issue of Soft Computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems (2010), http://sci2s.ugr.es/eamhco/
5. Hoos, H.H., Stützle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann Publishers, San Francisco (2005)
6. Hutter, F., Hoos, H.H., Leyton-Brown, K., Murphy, K.P.: An experimental investigation of model-based parameter optimisation: SPO and beyond. In: Rothlauf, F. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2009, pp. 271–278. ACM Press, New York (2009)
7. Liao, T., Montes de Oca, M.A., Stützle, T.: Computational results for an automatically tuned IPOP-CMA-ES on the CEC'05 benchmark set. Tech. Rep. TR/IRIDIA/2011-022, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
8. Liao, T., Montes de Oca, M.A., Stützle, T.: Tuning parameters across mixed dimensional instances: A performance scalability study of Sep-G-CMA-ES. In: Krasnogor, N., Lanzi, P.L. (eds.) GECCO (Companion), pp. 703–706. ACM Press, New York (2011)
9. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
10. López-Ibáñez, M., Liao, T., Stützle, T.: On the anytime behavior of IPOP-CMA-ES: Supplementary material (2012), http://iridia.ulb.ac.be/supp/IridiaSupp2012-010/
11. López-Ibáñez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. Tech. Rep. TR/IRIDIA/2012-012, IRIDIA, Université Libre de Bruxelles, Belgium (2012)
12. Smit, S.K., Eiben, A.E.: Beating the 'world champion' evolutionary algorithm via REVAC tuning. In: Ishibuchi, H., et al. (eds.) Proc. of CEC 2010, pp. 1–8. IEEE Press, Piscataway (2010)
13. Suganthan, P.N., Hansen, N., Liang, J., Deb, K., Chen, Y.P., Auger, A., Tiwari, S.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Tech. rep., Nanyang Technological University, Singapore (2005)
14. Zilberstein, S.: Using anytime algorithms in intelligent systems. AI Magazine 17(3), 73–83 (1996)
15. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Grunert da Fonseca, V.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation 7(2), 117–132 (2003)

HappyCat – A Simple Function Class Where Well-Known Direct Search Algorithms Do Fail

Hans-Georg Beyer and Steffen Finck

Research Center Process and Product Engineering
Department of Computer Science
Vorarlberg University of Applied Sciences
Achstr. 1, A-6850 Dornbirn, Austria
{Hans-Georg.Beyer,Steffen.Finck}@fhv.at

Abstract. A new class of simple and scalable test functions for unconstrained real-parameter optimization will be proposed. Even though these functions have only one minimizer, they yet appear difficult to be optimized using standard state-of-the-art EAs such as CMA-ES, PSO, and DE. The test functions share properties observed when evolving at the edge of feasibility of constraint problems: while the step-sizes (or mutation strength) drops down exponentially fast, the EA is still far way from the minimizer giving rise to premature convergence. The design principles for this new function class, called HappyCat, will be explained. Furthermore, an idea for a new type of evolution strategy, the Ray-ES, will be outlined that might be able to tackle such problems.

1 Introduction

The design of direct search methods for optimization problems in $\mathbb{R}^N$ is still a vivid area of research and publications. Reviewing various journals and conferences, one finds a plethora of proposals for new or improved algorithms. The superiority of which is usually validated by empirical investigations. Such investigations compare the performance of the new algorithm with a collection of other algorithms on a "well-crafted" set of artificial test functions. An alternative would be – of course – a performance comparison based on real-world applications (RWAs) or on toy problems derived from such RWAs. However, such kinds of comparisons are hard to find and/or difficult to perform (e.g., problem size scaling investigations are often excluded due to expensive goal function evaluations). This may be the main reason why one resorts to artificial test beds. The currently most-advanced endeavor in this direction is the COCO (COmparing Continuous Optimizers) initiative (URL: http://coco.gforge.inria.fr/) and the related Black-Box Optimization Benchmarking (BBOB) workshops at GECCO 2009, 2010, and 2012. This workshop series focuses on unconstrained optimization. However, in practice one often encounters constraints (not only box constraints) that restrict the feasible solutions in non-trivial manner. While there is also a series on benchmark competitions in constrained evolutionary optimization (see e.g. the CEC 2010 workshop [1]), it is interesting to notice that the most competitive strategies found at BBOB are not in the winner portfolio of the CEC constrained benchmarking competition. There might be different reasons for that observations and we do not want to speculate too

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 367–376, 2012.

much as to why this is the case. However, from our own attempts using CMA-ES [2] for a constrained optimization problem with linear inequality constraints, we have made the observation that this strategy can exhibit premature convergence if the optimizer is located in one of the vertices of the simplex. A similar behavior has been observed and analyzed theoretically by Arnold [3]. The premature convergence behavior is due to a failure of step-length control. When approaching the edge of feasibility the mutation strength decreases exponentially fast such that the CMA-ES is not able to learn the covariance matrix.

At first sight this premature convergence behavior might come as a big surprise given the fact that CMA-ES performs so well on the BBOB test bed. However, the problem lurks already in the BBOB test bed. It is this plain *sharp ridge* test function that carries already parts of the problem. The fact that one does not observe premature convergence for this function when using *standard implementations* of CMA-ES is simply due to a tiny implementation detail: There is always a test built-in that checks for a minimal step-size in the search space. One can find this kludge already in early CMA versions, see e.g. [4, p. 180]. While the CMA designers explained this implementation detail as a means to prevent numerical precision problem, we will provide a class of simple (unconstrained) test functions where CMA-ES fails to locate the optimizer with sufficient precision. This failure appears even though (or just because) these test functions share local similarities with the sharp ridge.

The rest of the paper is organized as follows. First we will describe the construction of a simple scalable test function class, called *HappyCat* with tuneable *"CMA-ES hardness."* Then we will provide empirical performance evaluations including not only CMA-ES, but also generic differential evolution (DE) and particle swarm optimization (PSO) algorithms to show that the problem is not only restricted to CMA-ES. In a next section we will outline a new ES, the so-called *Ray-ES* that can exhibit improved performance on this test function. Finally, we will give an outlook providing additionally a somewhat more complicated test function that should be subject for further research.

2 Bending the Ridge – HappyCat

The motivation for developing a new test function class was triggered by the behavior of ES on the ridge function class. Ridge functions can be expressed in terms of

$$f(\boldsymbol{x}) := x_1 + d\left(\sum_{i=2}^{N} x_i^2\right)^{\alpha}. \tag{1}$$

If $\alpha = 1/2$ we get a V-shaped ridge, the sharp ridge. A first systematic investigation of ES performance on ridge functions has been done in the PhD thesis of Oyman [5] during the late 1990s. He was the first to interpret the evolutionary minimization on ridge functions as a process of both approaching the ridge axis in an $N-1$-dimensional sub-space and decreasing the linear x_1 component in (1). If d is sufficiently large, $f(\boldsymbol{x})$ is dominated by an $N-1$-dimensional sphere model and the linear x_1 part is rather a (noisy) perturbation. While for $\alpha > 1/2$ the sphere model influence reduces when approaching the ridge axis, the opposite holds for $\alpha < 1/2$, and $\alpha = 1/2$ is the limit case. Evolution on the $\alpha = 1/2$ case is a race between sphere model minimization

and linear x_1 decrease (minimization!). If the sphere model is dominating, the mutation adaptation process decreases the mutation strength σ continuously (exponentially fast). As a result one observes premature convergence. This also holds for CMA-ES. In that case, the flow of covariance information obtained from the successful mutations into the covariance matrix is continuously reduced. Learning the covariance matrix has a complexity of $O(N^2)$ (measured in function evaluations), however, the shrinking of the $(N-1)$-dimensional sphere proceeds with $O(N)$. As a result, CMA-ES must necessarily fail for sufficiently large d. A way to circumvent this shrinking is by keeping the mutation strength σ at a reasonable level. Thus, the CMA can learn the ridge direction. And this approach (or similar ones) has been implemented in standard CMA-ES.

Learning the ridge direction solves the adaptation problem for CMA-ES on the sharp ridge. After having adapted the covariance matrix, the ES has only to follow a straight path. However, what happens if the path is not a straight line? To get an answer to this question, we first have to construct a simple test function with such a property. In order to keep things simple, a spherical path will be constructed. To this end, note that $\|\boldsymbol{x}\|^2 - N = 0$ describes a sphere with radius $\sqrt{N}$. That is, the function $(\|\boldsymbol{x}\|^2 - N)^2$ measures the deviation of an arbitrary $\boldsymbol{x}$ vector from the radius $\sqrt{N}$ sphere. Thus, one obtains a function with a degenerated zero minimum the optimizer $\boldsymbol{x}^*$ of which are all points on that sphere. Now we break the rotational symmetry by adding a simple unimodal quadratic function $f_q(\boldsymbol{x})$. Demanding the minimizer of $f_q(\boldsymbol{x})$ at $\boldsymbol{x}^* = (-1, \ldots, -1)^{\mathrm{T}}$ and for sake of simplicity $f_q(\boldsymbol{x}^*) = 0$, one obtains

$$f_q(\boldsymbol{x}) := \tfrac{1}{N}\left(\tfrac{1}{2}\|\boldsymbol{x}\|^2 + \textstyle\sum_{i=1}^{N} x_i\right) + \tfrac{1}{2}. \tag{2}$$

This can be easily checked by calculus. Putting things together, one obtains the HappyCat function the minimizer of which is $\boldsymbol{x}^* = (-1, \ldots, -1)^{\mathrm{T}}$ and $f_{HC}(\boldsymbol{x}^*) = 0$

$$\boxed{f_{HC}(\boldsymbol{x}) := [(\|\boldsymbol{x}\|^2 - N)^2]^{\alpha} + \tfrac{1}{N}\left(\tfrac{1}{2}\|\boldsymbol{x}\|^2 + \textstyle\sum_{i=1}^{N} x_i\right) + \tfrac{1}{2}.} \tag{3}$$

The case $N = 2$ is displayed in Fig. 1.

As one can see, the α-part in Eq. (3) produces an attracting groove for path-oriented search strategies. If $\alpha = 1/2$ the groove is V-shaped. For $\alpha < 1/2$ the groove shape resembles the geometry of a black hole. Actually, it turns out that getting closer to the groove results in an increasing descent gradient towards the bottom of the groove. Its absolute value goes to infinity. That is why, it is difficult to escape from this "black groove". Since the shape of the groove is tuneable by the α exponent, one can continuously control the problem hardness.

In Fig. 2 the performance of DE, PSO, and CMA on HappyCat with $N = 10$ and $\alpha = 1/8$ is shown. All strategies were used in a form close to their default version. DE is a *Rand 3* type strategy [6], which is almost identical to the (common) DE/rand/1/bin strategy. It uses a population size of $NP = 20$, crossover parameter $CR = 0.5$, and mutation parameter $F = 0.9$. PSO is a local best variant with a swarm of 20 particles, parameter $\varphi = 2.07$ (see [7]), and 3 informants per particle. The information links between the particles are randomly chosen at the start of each iteration and a particle will always inform itself. For CMA-ES the population parameters are $\lambda = 10$

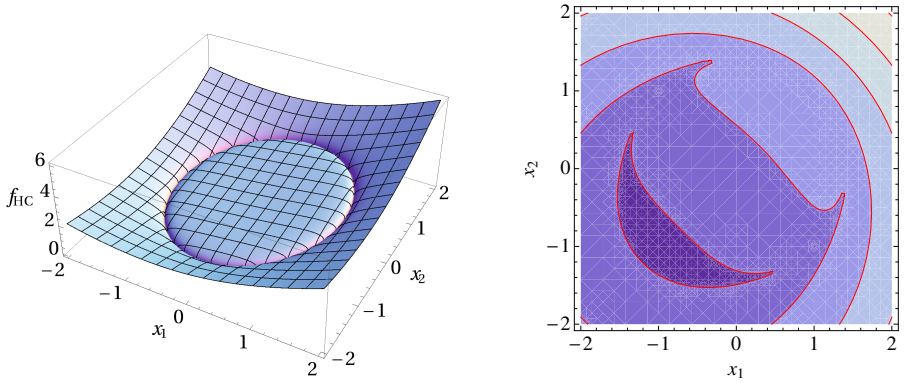


Fig. 1. HappyCat in two dimensions with $\alpha = 1/8$ as 3D-plot (left) and contour plot (right). The latter gave rise to the funny naming of this function.

(offspring) and $\mu = 5$ (parents). The remaining learning and cumulation parameters are identical to the default ones used in CMA-ES version `3.55.beta` obtained from URL: `http://www.lri.fr/~hansen/cmaes_inmatlab.html`. Additionally, the minimal coordinate axis deviation is set to $\forall j : \sigma\sqrt{C_{jj}} \geq 10^{-7}$ with C_{jj} being an entry of the diagonal of the covariance matrix.

The left plot of Fig. 2 shows the dynamics of the function value w.r.t. the number of function evaluations in a log-log format. In case of DE and PSO it represents the best function value in the current population, while for CMA it is the function value of the parent individual. The dynamics of the 3 strategies differ. CMA achieves fast progress before stagnating. PSO initially is comparable to CMA but enters the stagnation phase earlier. In contrast to CMA, the particles are able to find a region of improved fitness in later iterations (without restarting). On the other hand, DE shows a step-like characteristic where phases of stagnation are followed by small "improvement jumps". Overall, DE is slower compared with CMA and PSO. Inspecting the final state of the population in DE and PSO reveals that they are not converged (for $N = 10$). For PSO, the mean distance between the particles is slightly reduced compared to the initial mean distance and a similar observation is made for the particles' velocities. This indicates that there is "kinetic energy" left in the swarm, however, it is difficult to find improved solutions. Considering the positions of the personal best solutions, one finds that PSO tracks the groove very quickly. From that point on, progress can be made by either reducing the distance to the groove bottom or by moving toward the optimizer. Since reducing the distance to the groove bottom is much more rewarding, the personal best positions will not converge toward a single point but rather be distributed along the groove bottom. This in turn prevents a reduction in the velocities (except for the global best point) and impairs the local search behavior. For DE the situation is somewhat different. In small dimensions $N \leq 5$ convergence in the experiments performed ($NP = 20$) is observed. There the expected population variance [8] is less than 10^{-14}, however, DE converges to non-optimal points. For larger search space dimensionalities, the diversity in the population remains large and similar to PSO, the points are distributed along the groove. Since DE employs a greedy selection scheme, new population members are

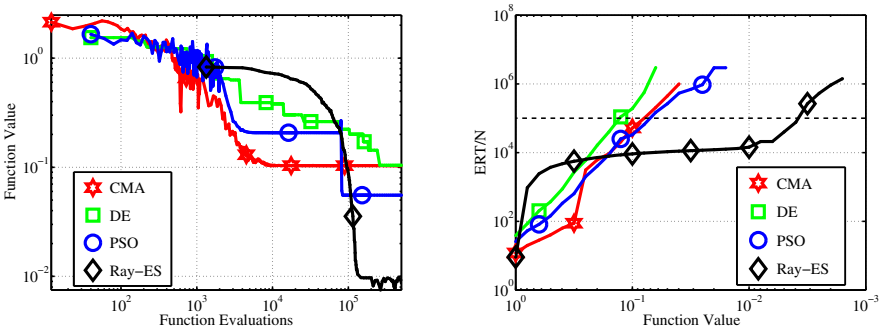


Fig. 2. Dynamic behavior of different strategies on the HappyCat function with $N = 10$ and $\alpha = 1/8$. In the left figure single run dynamics are shown, while in the right one the curves are based on 30 samples for each strategy. The term ERT refers to expected running time, expressed in number of function evaluations. The horizontal line in the right plot indicates the budget of function evaluations for each sample. Note, the vertical axis is normalized by the search space dimensionality N and the horizontal axis is reversed in direction. As for the fourth strategy, the Ray-ES, see Section 3.

only accepted if the distance to the groove bottom and/or the distance to the optimizer is reduced. However, the newly created individuals depend on the distances between the population members, hence only slow progress is made.

In CMA, the mutation step generated by $\sigma\mathcal{N}(\mathbf{0}, \boldsymbol{C})$, with $\boldsymbol{C}$ as covariance matrix, decreases quickly. Once the mutation step is too small, the progress of CMA stops. The performance of CMA can be improved by using larger population sizes than the default one. While being slower in the early iterations a larger population size comes closer to the optimizer and has a better performance at some point. For DE and PSO no such improvement with regard to the population size is observed.

Considering more than just a single run, yields the right-hand plot in Fig. 2. For all experimental runs, the necessary individual(s) for each strategy are initialized by uniformly drawing a vector from the range $[-2, 2]^N$. The budget is set to $10^5 N$ function evaluations and 30 samples are performed for each strategy. Restarts of the strategies are allowed as long as the budget is not exhausted. In the plot the expected running time (ERT) [9] is shown as function of the best-so-far function value of all evaluated points. ERT represents the expected behavior in terms of solution quality and necessary budget. The horizontal dashed line indicates the available function evaluation budget. Points above this line indicate function values which were not achieved in all samples. In such a case the success probability is less than 1 and its inverse becomes a factor in the calculation of ERT. Therefore these data points are based on an extrapolation of the available experimental data. To achieve these performances (to a certain extent) one must increase the function evaluation budget and provide better restart criteria. Considering the trend of the ERT-curves, one observes for PSO and DE that each curves could be approximated by a straight line. This indicates a power law relation between function value and function evaluation budget. For CMA there exists a jump in the curve

indicating (probably) the phase where the covariance matrix is adapted. Before and after this jump a power law relation approximates the relation between function value and number of function evaluation.

However, the best curve is the one for Ray-ES. This strategy can achieve an order of magnitude better solution quality (see also the left-hand plot) and is competitive with CMA and PSO in terms of function evaluations for $f \leq 10^{-1}$. In the next section we will describe Ray-ES.

3 Ray-ES

In the following we propose a concept for treating the HappyCat function. Note, this is a conceptual algorithm and not a fully developed strategy. Starting from a fixed point in the domain the idea is to find the ray direction which contains the optimizer. To this end, the strategy evolves ray directions and performs (simple) line searches along these rays to evaluate their quality. The ray evolution itself is based on the blueprint of a $(\mu/\mu_l, \lambda)$-σSA-ES [10], hence the name Ray-ES.

Algorithm 1. Ray - ES

1: **repeat**
2: **for** $l \leftarrow 1$ to λ **do**
3: $\tilde{\sigma}_l \leftarrow \sigma \mathrm{e}^{\tau \mathcal{N}(0,1)}$
4: $\tilde{\boldsymbol{y}}_l \leftarrow \boldsymbol{y} + \tilde{\sigma}_l \boldsymbol{\mathcal{N}}(\mathbf{0}, \boldsymbol{I})$
5: $\boldsymbol{r}_l \leftarrow \dfrac{\tilde{\boldsymbol{y}}_l}{\|\tilde{\boldsymbol{y}}_l\|}$
6: $[\tilde{\boldsymbol{x}}_l \ \tilde{f}_l] \leftarrow \mathrm{LineSearch}\,(\boldsymbol{r}_l)$
7: **end for**
8: $\boldsymbol{y} \leftarrow \langle \tilde{\boldsymbol{y}} \rangle$ ▷ new ray direction
9: $\sigma \leftarrow \langle \tilde{\sigma} \rangle$ ▷ new mutation strength
10: **until** termination criterion satisfied

In Alg. 1 the pseudocode for the basic version of Ray-ES is shown. Due to the underlying design principles, one must specify values for the population sizes λ and μ, and the learning parameter τ. The parental mutation strength σ and the parental ray $\boldsymbol{y} \in \mathbb{R}^N$ must be initialized. From line 2 to line 7 in Alg. 1 λ new rays are created by mutation (line 4) and evaluated (line 6). The mutation operator follows the self-adaptation scheme [10], i.e. each ray has its own mutation strength $\tilde{\sigma}_l$ which itself is a mutant of the parental σ (line 3). Since one is only interested in the direction of the ray, it is normalized (line 5) before being evaluated. The evaluation is performed by the function LineSearch which is given in Alg. 2. It returns the best point found $\tilde{\boldsymbol{x}}_l$ and its corresponding function value $\tilde{f}_l$ which serves as measures for the ray quality. In lines 8 and 9 the variables for the parental mutation strength and parental ray are updated by means of intermediate recombination where the μ best of the λ offspring are used. The rule is

$$\langle x \rangle = \frac{1}{\mu} \sum_{m=1}^{\mu} x_{m;\lambda}, \tag{4}$$

where $x_{m;\lambda}$ is the mth best of the λ values. The ranking is done for all parameters w.r.t. the function value. If no termination criterion is satisfied the evolutionary process continues. Typical termination criteria are based on solution quality, budget of function evaluations, and/or measures for the stagnation of the evolutionary process.

Algorithm 2. LineSearch

1: **function** LINESEARCH($\boldsymbol{r}$)
2: set $L, k, \boldsymbol{o}, \epsilon$
3: $\boldsymbol{x}_0 \leftarrow \boldsymbol{o}$
4: $\Delta r \leftarrow \frac{L}{k}$
5: **while** $\Delta r > \epsilon$ **do**
6: **for** $p \leftarrow 1$ to $2k+1$ **do**
7: $\boldsymbol{x}_p \leftarrow \boldsymbol{x}_0 + \Delta r(p-k-1)\boldsymbol{r}$
8: $f_p \leftarrow F(\boldsymbol{x}_p)$
9: **end for**
10: $\Delta r \leftarrow \begin{cases} 2\Delta r, & \text{if } (\boldsymbol{x}_{1;2k+1} = \boldsymbol{x}_1) \vee (\boldsymbol{x}_{1;2k+1} = \boldsymbol{x}_{2k+1}) \\ \Delta r/k, & \text{otherwise} \end{cases}$
11: $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_{1;2k+1}$
12: **end while**
13: **return** $\boldsymbol{x}_{1;2k+1}, f_{1;2k+1}$
14: **end function**

The evaluation of a ray is a line search for the minimizer on the ray. The procedure is stated in Alg. 2. It requires the ray direction $\boldsymbol{r}$, an initial search length $L \in \mathbb{R}^+$, the number of subdivisions $k \in \mathbb{Z}^+$, the ray origin $\boldsymbol{o} \in \mathbb{R}^N$, and the minimal division length $\epsilon \in \mathbb{R}$ acting as precision measure. Except for the ray direction all these parameters are held globally constant. In line 4, the length Δr of the k sections is initialized. The line search (lines 5–12) is then performed as long as Δr is greater than ϵ. At first, k equidistant points in positive and negative ray direction from the start point $\boldsymbol{x}_0$ are created (line 7) and evaluated (line 8). The start point itself is also evaluated, resulting in $2k+1$ function evaluations. The best of these points, $\boldsymbol{x}_{1;2k+1}$, is set as new start point (line 11). To find a better approximation of the minimizer, the length Δr is reduced by factor k iff $\boldsymbol{x}_{1;2k+1}$ is not at the ends of the ray considered. However, if the current minimum is at the ends of the ray, the section length is doubled (line 10). It can be shown that the number of function evaluations for each line search can be estimated as (provided that the strategy does not leave the initial search interval $[-L, L]$)

$$\mathrm{FEs}_{LS} \simeq \frac{2k}{\ln k} \ln \frac{L}{\epsilon}. \tag{5}$$

In the actual implementation of Ray-ES we also memorized the best-so-far solution and evaluated the center of gravity of the points returned by LineSearch. In some situations this recombinant achieved an improved solution quality. Throughout this text the following parameter setting is used for Ray-ES: $\lambda = 10, \mu = 3, \tau = 1/\sqrt{N}, k = 3, L = 2, \boldsymbol{o} = (0, \ldots, 0)^N$, and $\epsilon = 10^{-8}$.

The single run dynamics and the expected performance for Ray-ES are shown in Fig. 2. The single run curve is based on the best value returned by LineSearch and

aforementioned recombinant. Ray-ES is initially slower than the other strategies considered and needs more function evaluations per iteration. This is due to the nearly constant line search effort given by (5).[1] However, at some point it is competitive with the other strategies and later achieves a solution quality not realized by the other strategies (for the parameters considered). The steep rise of the slope at the end is due to the decrease in the success probability for the function values considered.

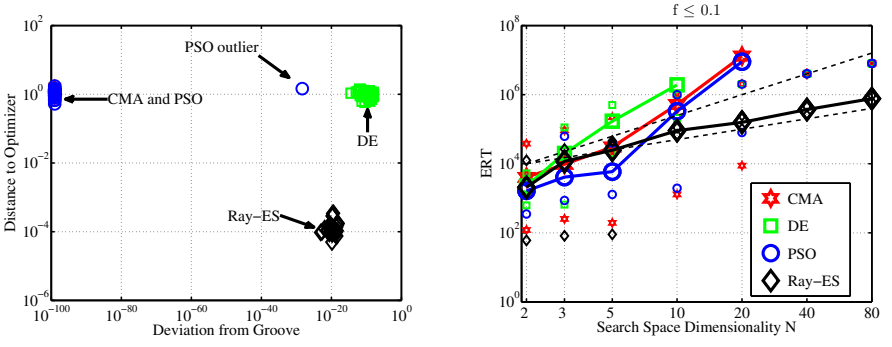


Fig. 3. The left-hand plot shows the distribution of the best solutions found in 30 samples in terms of distance to the optimizer $(-1,\ldots,-1)$ and the deviation from the groove for $N = 10$. The right-hand plot shows the scaling of the expected running time ($\mathrm{ERT}(f \leq 10^{-1})$) for the strategies as function of the search space dimensionality N. The dashed lines represent linear and quadratic scaling, the small markers indicate the best and worst observed number of function evaluations.

In Fig. 3 additional performance plots are shown for all strategies. In the left-hand plot the distribution of the best point found in each of the 30 samples ($N = 10$) is shown. While PSO and CMA-ES are located at the groove bottom (horizontal axis in Fig. 3 left)[2], Ray-ES is able to achieve a much smaller distance to the optimizer (see vertical axis, there is a factor of about 10^{-4}) while still being considerably close to the groove bottom. That is, the final solutions obtained by CMA, DE, and PSO are rather poor when evaluated in the search space (i.e., w.r.t. distance to the optimizer).

In the right-hand plot of Fig. 3 the scaling of ERT w.r.t. the search space dimensionality is presented. The curves represent the expected running time to find a point with $f \leq 10^{-1}$ for the first time. Ray-ES shows a scaling behavior between linear and quadratic, while the other strategies have a greater than quadratic scaling behavior.

In Fig. 4 the performance of Ray-ES (left) and CMA-ES (right) for various problem hardnesses is shown. In case of Ray-ES, the curve with $\alpha = 0.1$ is an outlier which is due the choice of the minimal division length ϵ. Decreasing ϵ improves the performance. For CMA-ES the performances are not in order with α, i.e. $\alpha = 0.1$ is easier than $\alpha = 0.2$. Investigations into this behavior showed that it might be due to the frequency of restarts triggered. For small α values much more restarts occurred than for larger

[1] One may think of an adaptive line search to reduce FEs_{LS} in the initial phase, however, this is beyond the scope of this paper.

[2] For visualization reasons 10^{-99} was added to the distance from the groove bottom.

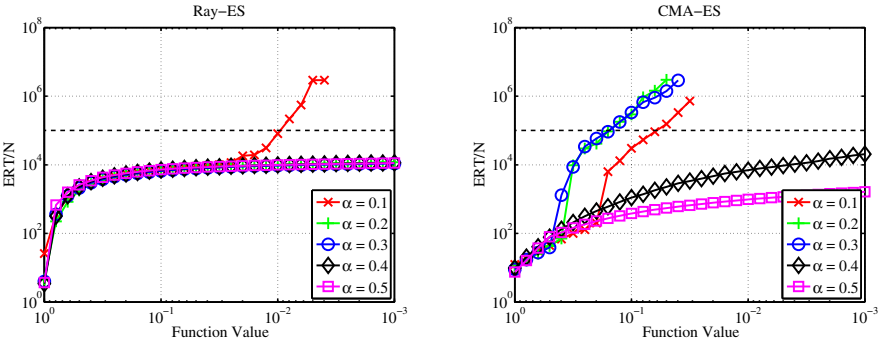


Fig. 4. Scaling of ERT with respect to the problem hardness α for $N = 10$ and the default parameter settings

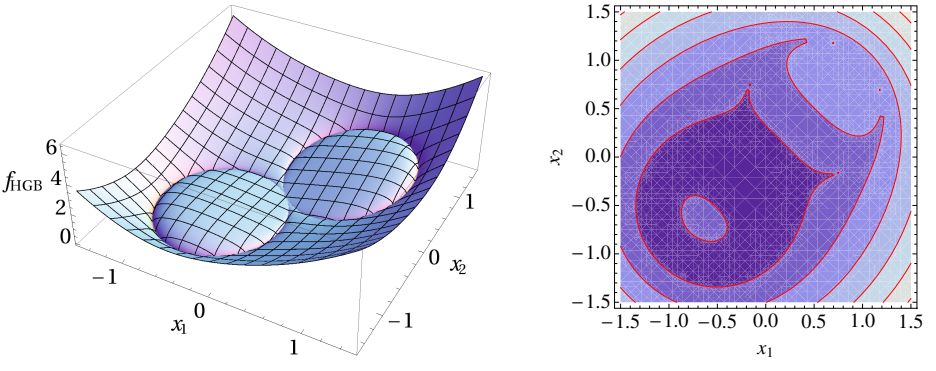


Fig. 5. HGBat in two dimensions with $\alpha = 1/4$ as 3D-plot (left) and contour plot (right). The latter gave rise to the funny naming of this function resembling the silhouette of Batman's head.

values. Note, we did not use stagnation as a possible restart criterion. For DE and PSO the performance improves with increasing values of α (decreasing problem hardness).

4 Conclusions and Outlook

In this paper, we have proposed a new scalable test function for EAs in real-coded search spaces that allows for a continuous tuning of the problem hardness via α in Eq. (3). We have shown empirically that standard state-of-the-art EAs such as CMA-ES, DE, and PSO do fail on such topologies even in the case of small search space dimensionalities if α is chosen below a certain critical value. We also provided a proof of concept for a new class of evolution strategies, the Ray-ES, that might cope with this kind of function topologies. However, the investigations concerning this new strategy type are still in the beginning and the performance of the strategy depends on the choice of the ray origin. Yet it is a new strategy that might be worth further investigations in the future.

The main purpose of this paper is test function design. From this aspect, the design principle behind HappyCat can also be used to construct more complex functions. "Complex" is meant here in the sense that the path defined by the groove can assume more complex forms than the spherical one. As an example, HGBat shall be mentioned here

$$f_{HGB}(\boldsymbol{x}) := \left[\left(\|\boldsymbol{x}\|^4 - \left(\sum_{i=1}^N x_i\right)^2\right)^2\right]^\alpha + \frac{1}{N}\left(\frac{1}{2}\|\boldsymbol{x}\|^2 + \sum_{i=1}^N x_i\right) + \frac{1}{2}. \quad (6)$$

Comparing with HappyCat (3), one sees that the quadratic symmetry-breaking part defined in (2) remains the same. The only difference is due to the first term where the expression in the bracket is a degree 8 polynomial instead of a degree 4 polynomial in Eq. (3). The 2D shape of (6) is shown in Fig. 5.

Investigations concerning this function and even more complex forms remain to be done in the future. Furthermore, it is our hope that this kind of test functions, modeling certain aspects of search and (co-) variance adaptation in constrained optimization problems, will be incorporated in the commonly used testbeds of black box optimization [9].

Acknowledgments. This work was supported by the Austrian Science Fund (FWF) under grant P22649-N23.

References

1. Mallipeddi, R., Suganthan, P.N.: Problem Definitions and Evaluation Criteria for the CEC 2010 Competition on Constrained Real-Parameter Optimization. Technical report, Nanyang Technological University (2010)
2. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). Evolutionary Computation 11(1), 1–18 (2003)
3. Arnold, D.V.: Analysis of a Repair Mechanism for the $(1, \lambda)$-ES Applied to a simple constrained problem. In: Krasnogor, J., et al. (eds.) GECCO-2011: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 853–860. ACM, New York (2011)
4. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation 9(2), 159–195 (2001)
5. Oyman, A.I.: Convergence Behavior of Evolution Strategies on Ridge Functions. Ph.D. Thesis, University of Dortmund, Department of Computer Science (1999)
6. Feoktistov, V.: Differential Evolution: In Search of Solutions. Springer-Verlag New York, Inc., Secaucus (2006)
7. Clerc, M.: Particle Swarm Optimization. ISTE Ltd., London, UK (2006)
8. Beyer, H.-G., Deb, K.: On Self-Adaptive Features in Real-Parameter Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation 5(3), 250–270 (2001)
9. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report RR-7215, INRIA (2010)
10. Beyer, H.-G., Schwefel, H.-P.: Evolution Strategies: A Comprehensive Introduction. Natural Computing 1(1), 3–52 (2002)

Differential Gene Expression with Tree-Adjunct Grammars

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

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

Abstract. A novel extension of an existing artificial Gene Regulatory Network model is introduced, combining the dynamic adaptive nature of this model with the generative power of grammars. The use of grammars enables the model to produce more varied phenotypes, allowing its application to a wider range of problems. The performance and generalisation ability of the model on the inverted-pendulum problem, using a range of different grammars, is compared against the existing model.

1 Introduction

Recently in the field of Evolutionary Computation there has been an increase of interest in developmental biology and how it affects the evolutionary models currently in use. It is interesting to investigate whether these developmental systems can be applied to improve existing evolutionary approaches, as developmental processes play such an important role in nature.

To date, much of the work that has been done with developmental systems in the field of Genetic Programming (GP) has been ontogenetic in nature. That is to say, concerning with the growth or morphogenesis of individuals, by means of interaction with the environment [12, 5, 4]. These developmental systems derive the phenotype from the genotype, evaluating this phenotype, before undergoing some morphogenesis defined within the phenotype itself. This process continues, creating more complex phenotypes. The organism morphologies in nature on which these systems are modeled on, however, are themselves the result of an underlying regulatory process, known as gene regulatory networks (GRNs).

GRNs, which are at the core of developmental biology allow for *"differential gene expression from the same nuclear repertoire"* [3]. This is partly achieved by including feedback loops and being directly influenced by the environment. An outline of a new genetic representation for GP using GRNs was given by Banzhaf [1], exploring the dynamics of such a system. This new representation has since been extended, providing methods for encoding the state of the environment into the GRN, as well as extracting a signal from the GRN [10]. However, even in its extended form, the phenotype produced is a series of signals between 0.0 and 1.0, making this representation difficult to apply to many different problems.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 377–386, 2012.

This study is concerned with applying the generative power of grammars to the model, and investigating the effects of this complexification of the mapping process. Making use of grammars to generate phenotypes enables the model to be applied to many different problems. Tree-Adjunct Grammatical Evolution (TAGE) [9], a variant of the popular grammar-based form of GP, Grammatical Evolution (GE), is extended to include the GRN model. TAGE is ideal due to a unique property of the grammar type it employs, Tree-Adjoining Grammars (TAGs) [6], which produce valid individuals at every stage of derivation.

The following section gives introductions to TAGE and the GRN model. Section 3 presents the extension of the GRN model into the TAGE algorithm, followed by a description of the pole-balancing problem in section 4. The experiments performed and results obtained are presented with some discussion in section 5. The study concludes in section 6, outlining some future work.

2 Background

2.1 Tree-Adjunct Grammatical Evolution

TAGE is a grammar-based form of GP, combining aspects of Darwinian natural selection, genetics and molecular biology with the representational power of grammar formalisms [11, 9]. TAGE uses a representation consisting of a TAG and a chromosome. 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 non-terminal (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*; and A is a finite set of finite trees called *auxiliary trees*.

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`. 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 make use of codons to join elementary trees together. When using TAGs, at each stage of derivation, before and after each adjunction operation, valid phenotypes can be extracted. Due to space constraints, a full description of TAGE has been omitted, but can be found in [9]. In addition, the grammars used throughout this study are presented in Fig. 4 as context-free grammars (CFG) in an effort to save space. TAGE can transform between CFGs and equivalent TAGs.

2.2 Artificial Gene Regulatory Networks

The GRN model [1] used in this study, which has been extended to enable input and output [10], consists of three components: a genome, genes, and proteins. The model mimics the biological interaction of proteins with a cell's genes. By binding at *regulatory sites*, certain proteins can regulate the expression of genes, and hence, the production of additional proteins. Proteins are assigned concentrations in the model, with a total concentration of 1.0 for each type of protein.

The term *concentration*, with regards to this model, represents the proportion of a particular protein to the rest of the proteins of the same type.

Gene. A gene consists of four sections, as shown in Fig. 1. The first two are two 32 bit regulatory sites, the *enhancer* and *inhibitor* sites. These two sites affect a gene's expression positively or negatively, respectively. A 32 bit *promoter site*, which follows these regulatory sites, is of the form `XYZ01010101` where `XYZ` is an arbitrary 24 bit sequence and the final eight bits define the gene's type or signature. The specific eight bit signature of a promoter is used to identify genes along the genome. Following the promoter is a 160 bit region, which encodes the protein. These 160 bits are subdivided into five 32 bit sections. A majority vote is performed at each bit position across the five sections to determine the 32 bits making up the protein signature, as shown in Fig. 2.

Gene/Protein Types and Input/Output. Genes are split into two groups in this model, *Transcription Factors* (TF-genes) and *Products* (P-genes). A gene's type is dependent upon its promoter's signature, in this case `XYZ00000000` is used to identify TF-genes and `XYZ11111111` to identify P-genes. TF-genes produce TF-proteins which can bind to regulatory sites and affect gene expression. P-genes produce P-proteins which are are used solely to extract output from the model, and as such, are prevented from binding and affecting regulation. The sum of concentrations for each protein type must add to 1.0. Input into the model is achieved by injecting specific concentrations of *free* TF-proteins into the model. Input values are encoded into the proteins' concentration levels. The concentration of free proteins remains static unless new inputs are injected.

Regulation. How much each protein affects a gene's expression by binding at that gene's regulatory sites (enhancer and inhibitor) is calculated by taking the XOR of the protein's signature with that of the regulatory site. The number of bits set is the degree of match between the two, i.e., the number of complementary bits between them. The enhancing, e_i, and inhibiting, h_i, signals for the expression of gene g_i are calculated as follows:

$$e_i, h_i = \frac{1}{N} \sum_{j=1}^{N} c_j e^{\beta(u_j - u_{max}))} \ , \tag{1}$$

where N is the total number of TF-proteins, c_j is the concentration of protein j, u_j is the number of complementary bits between the regulatory site and protein j, u_{max} is the maximum number of complementary bits observed in the system, and β is a positive scaling factor. A value of 1.0 was used for scaling factors β and δ across all experiments.

The expression rate of g_i (the production of p_i) at time $t + 1$ is given as:

$$\frac{dc_i}{dt} = \delta(e_i - h_i)c_i \ , \tag{2}$$

where δ is a scaling factor. Once all produced TF-protein concentrations have been updated, these concentrations are scaled such that when summed with the free TF-protein concentrations they add to 1.0.

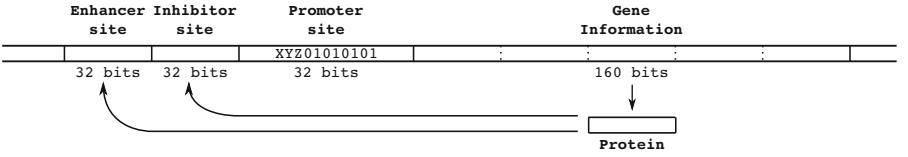


Fig. 1. A gene on the genome split into its different sections

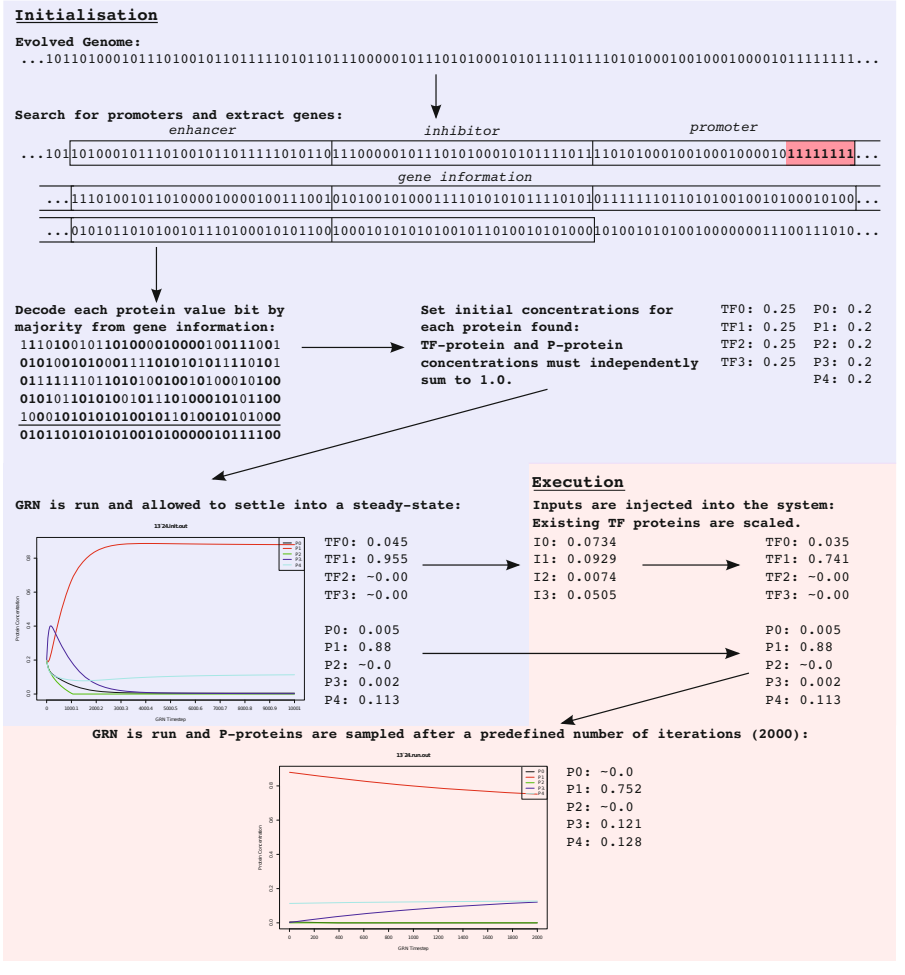


Fig. 2. An illustrative example of the GRN model. The genome is scanned for promoter sites and genes are extracted. Each gene's protein signature is calculated and each protein is given an initial concentration. The system is run for a number of iterations to become stable. The environment state is then read, encoded as TF-protein concentrations and injected into the system, scaling the existing TF-protein concentrations. The system is run for a number of iterations before extracting P-proteins to be used for mapping. This process of encoding and injecting inputs, iterating and extracting outputs can continue indefinitely.

A similar formula is used for the expression rate of P-genes. P-protein concentrations are normalised separately from the TF-protein concentrations to sum to 1.0.

$$\frac{dc_i}{dt} = \delta(e_i - h_i) \ , \tag{3}$$

3 Combining Artificial GRNs and Grammars

In order to enable the use of GRNs within the TAGE algorithm, the TAGE pipeline must be extended. The GRN model is embedded at the start of the mapping process. This enables easy access to the evolved genome in order to search for TF-genes and P-genes to construct the GRN, as well as providing access to the traditional mapping process to remap the phenotype each time the fitness function/environment changes. In order to allow for this, a feedback loop is required from the fitness function to the mapper.

To evaluate an individual, its genome must first be mapped. The mapping call initiates a search of the individual's genome for genes and a GRN is constructed. This GRN is allowed to run, without free TF-proteins (inputs), for 10000 iterations in an effort to allow the network to settle into a steady state. This may stop prematurely if a steady state within the GRN is detected earlier.

When the fitness function is called to evaluate the individual, the initial state of the environment, encoded as concentrations of free TF-proteins, is passed to the mapper and injected into the GRN, scaling the existing produced TF-proteins as necessary. The GRN is then run for a predefined number of iterations before the P-proteins and their concentrations are used to create a string of integer codon values. These codons are then used by the traditional mapping process to produce a valid phenotype. One of the major advantages of using TAGs is that regardless of how many codon values are available to the mapper, a valid phenotype can always be produced.

3.1 Using Product-Proteins as Mapping Inputs

There are many different methods of interpreting P-proteins and their concentration levels to produce codon values for mapping. In this study, four such methods are examined, two for use with a binary grammar and two for use with grammars of any arity. The first two methods are chosen to simulate the approaches taken in [10] while still retaining the use of a direct mapping grammar. These two methods make use of a single P-protein to produce a codon value, whereas the remaining two methods use all available P-proteins to produce a chromosome of codon values, enabling mapping with grammars of much higher complexity.

Concentration Value. The P-protein's concentration is examined. If it is found to be greater than 0.5, a codon value of `1` is used. Otherwise, a codon value of `0` is used. Using Fig. 2 as an example, After the model has iterated, `P0` has a concentration of ~ 0.0 producing a codon value of `0`. This results in the tree at index `0` from the grammar being chosen.

Concentration Tendency. The change in concentration of a P-protein from input injection until the final iteration. If the magnitude of this change is greater than some *threshold*, a codon value of 0 or 1 is used depending on the sign of that change, positive or negative respectively. Otherwise, the codon value chosen previously will be reused. Taking Fig. 2 as an example, after the inputs have been injected, `P0` has concentration of 0.05, and a concentration of ~ 0.0 after iteration `2000`. As $|-0.05|$ is both greater than the *threshold* of 1^{-10} and is negative, a codon value of `1` is used, choosing the initial tree from index `1` in the grammar.

Sort by Concentration. The set of P-proteins is sorted in descending order by concentration level, as was suggested in [1]. The P-protein's 32 bit signatures are then used to represent integer codon values, with the most concentrated P-protein being the left-most codon. In Fig. 2, the P-proteins when sorted by concentration are `P1, P4, P3, P0, P2`, using their signatures as codons in this order, an initial tree is chosen from the grammar (one codon) and two adjunction operations are performed (two codons each).

Sort by Concentration Tendency. Similar to the *Sort by Concentrated* method above, however the P-proteins are sorted by means of the signed magnitude of the tendency of their concentration values. In this case, the P-proteins are sorted in the order `P3, P4, P2, P0, P1` as `P3` has the largest increase of 0.119 and `P1` with the smallest of -0.128. The protein signatures are used as codon values in this order with the grammar to create a phenotype.

4 The Inverted Pendulum (Pole-Balancing) Problem

The problem examined throughout the course of this study is the pole-balancing problem [2, 13], a classic dynamic control problem that has recently been examined in the study of applying GRNs to evolutionary computation [10, 8]. The description below is based on the problem setup used by Nicolau et al. [10].

The problem consists of simulating a cart which can move along a finite two dimensional track. A rigid pole is hinged to the centre of the cart. Forces may be applied to the cart in either direction, causing the cart and the pole to accelerate either left or right. The aim of the problem is to prevent the angle created between the pole and the vertical from becoming greater than some threshold, and keeping the cart within the bounds of the track. The problem model consists of four state variables:

$x \in [-2.4, +2.4]m$ the cart position, relative to the centre of the track;
$\dot{x} \in [-1.0, 1.0]m/s$ the velocity of the cart;
$\theta \in [-12, 12]^\circ$ the angle of the pole with the vertical;
$\dot{\theta} \in [-1.5, 1.5]^\circ/s$ the angular velocity of the pole.

The (friction-less) physical simulation of the cart-pole model is governed by the following non-linear differential equations:

$$\ddot{\theta}_t = \frac{g \sin\theta_t - \cos\theta_t \left[\frac{F_t + ml\dot{\theta}_t^2 \sin\theta_t}{m_c+m}\right]}{l\left[\frac{4}{3} - \frac{m\cos^2\theta_t}{m_c+m}\right]} \qquad \ddot{x}_t = \frac{F_t + ml\left[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t\right]}{m_c + m}, \quad (4)$$

where $g = -9.8m/s^2$, acceleration due to gravity, $m_c = 1,0kg$, the mass of cart, $m = 0.1kg$, the mass of pole, $l = 0.5m$, the half-pole length, $F_t = \alpha \cdot 10N$ where $\alpha \in [-1.0, 1.0]$, the force applied to the cart's center of mass at time t.

These variables are encoded as free TF-proteins for the GRN model by assigning a unique signature to each variable, with each variable's value normalised in the range $[0.0, 0.1]$ taking up a max concentration of 0.4 in the GRN model. The signatures of each variable are:

x: `00000000000000000000000000000000` θ: `11111111111111111111111111111111`
$\dot{x}$: `11111111111111110000000000000000` $\dot{\theta}$: `00000000000000001111111111111111`

These free TF-proteins are injected into the stabilised GRN and the existing produced TF-proteins are normalised as required. The GRN is then iterated 2000 times, corresponding to $0.2s$ of simulated time for the cart-pole model. After which, the P-proteins are mapped and the phenotype is evaluated resulting in a scaling co-efficient, α, for the force. The value of α is clamped between -1.0 and 1.0 inclusive. The scaled force is applied to the equations of motion and the state variables are updated, re-encoded, and fed back into the GRN. This process continues until the maximum number of time steps is reached, or the cart-pole model enters a failure state, i.e., $-2.4 > x > 2.4$ or $-12° > \theta > 12°$. Fitness is calculated by:

$$F(x) = \frac{120000}{successful\ time\ steps} - 1\ . \tag{5}$$

5 Experiments

This study is concerned with the application of grammars to the artificial GRN model. Several experiments are presented to help to examine this effect. The grammars used throughout this study are listed in Fig. 4, with the general evolutionary parameters of the system used in Tab. 1. For each run, the pole-cart model is initialised with a random state at the start of each generation.

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

Parameter	Value
Generations	50
Population Size	250
Initialisation	Random
Chromosome Size	128 (4096 bits)
Replacement Strategy	Generational
Elitism	25 Individuals
Selection Operation	Tournament
Tournament Size	3
Bit Mutation Prob	0.005

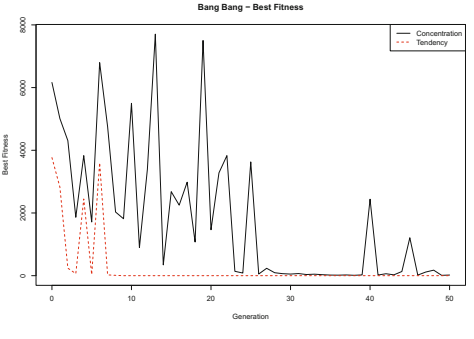


Fig. 3. Mean best fitness plot

```
<power>  ::= 1.0 0.0 - | 1.0
```

(a) Direct mapping grammar

```
<power>  ::= <const> 0.0 <op>
<op>     ::= + | -
<const>  ::= 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
             0.6 | 0.7 | 0.8 | 0.9 | 1.0
```

(b) Discrete digits grammar

```
<power>  ::= <const> 0.0 <op>
<op>     ::= + | -
<const>  ::= 0.<digits>
<digits> ::= <digit><digits> | <digit>
<digit>  ::= 0 | 1 | 2 | 3 | 4 | 5 |
             6 | 7 | 8 | 9
```

(c) Continuous digits grammar

```
<power> ::= <expr> <expr> <op>
<expr>  ::= <expr> <expr> <op> | <const>
<op>    ::= + | - | * | /
<const> ::= 0.<digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 |
            6 | 7 | 8 | 9
```

(d) Symbolic regression grammar

Fig. 4. The grammars used, in reverse polish notation. Note that (c) generates the range $(-0.9, 0.9)$ and the values of (d) are clamped to $[-1.0, 1.0]$ as previously.

In order to determine whether the system used by this study is comparable to the work done previously by Nicolau et al. [10], 50 independent runs of the simple grammar using the first and second output methods (see Section 3.1) are performed, using only a single codon value, and producing either `-1` or `+1`. Following this, assessing the effect of grammar complexity on the system, three different grammars of increasing complexity are examined over 50 runs using the remaining two P-protein mapping approaches.

5.1 Analysis of Results

The mean best fitness plots of the direct mapping grammar experiments are presented in Fig. 3. From the plot, it appears that the P-protein concentration tendency approach performs better, managing to find solutions in far fewer generations than using the protein's concentration value. The concentration tendency approach also manages to find more solutions overall than the value approach.

While these plots are similar in trend to those presented in [10], showing a similarity in performance, Fig. 3 has a higher variance. This variance can be accounted for by the choice of selection methods, generational vs steady-state [10]. Similar plots are presented for the three other grammars in Fig. 5. From these plots, it appears that using the concentration tendency approach is better than using the protein concentration itself.

Generalisation Results. The generalisation test proposed by Whitley et al. [13] is used in this study to examine robustness. After each run, the best individual is tested for 1000 time steps on 625 test cases. For each of the problem's four state variables, a set of five values is calculated, with these values normalised to 0.05, 0.275, 0.725 and 0.95 of the each variable's range. The test cases are obtained from combining these values. However, only `457` cases are solvable [10].

Given how quickly solutions are found by the tendency approach, as seen in Fig. 5, it is not surprising that this approach does not perform as well in terms of generalisation. Tab. 2 shows that while finding fewer successful solutions, the concentration value approach performs better in the generalisation tests.

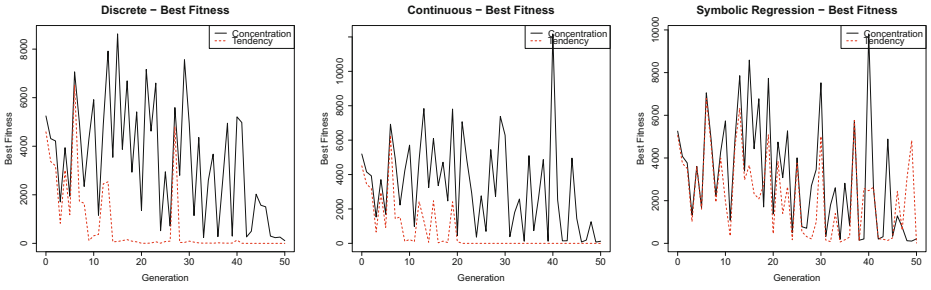


Fig. 5. Mean best fitness plots for the three more complex grammars

Table 2. Generalisation test results: 1000 time steps for 625 test cases

Approach	Best	Worst	Median	Mean	Std. Dev.	Suc. (50)
Best Product - Concentration	406	0	203	203.18	116.05	47
Best Single Output - Tendency	137	0	52	57.52	34.43	50
Discrete - Concentration	355	0	110	120.7	111.76	36
Discrete - Tendency	240	20	87	88.68	45.13	50
Continuous - Concentration	390	0	162	155.48	118.46	40
Continuous - Tendency	200	10	51	61.62	36.35	50
Sym. Reg. - Concentration	356	0	111	128.44	110.21	39
Sym. Reg. - Tendency	208	0	69	78.82	51.75	43

This could be due to the fact that perfect individuals which use this approach appear much later in the run, with the population having being exposed to more instances of the problem than those using the tendency approach. Allow evolution to continue for the whole run rather than stopping once a solution to any instance of the problem is found might help counteract this.

6 Conclusions

The objective of this study was to examine the effect of integrating grammars and an artificial GRN model, the motivation for which is two fold. Firstly, advances in developmental biology have increased our understanding of developmental processes and how they affect the natural evolution; this knowledge is slowly filtering into EC fields such as GP. Secondly, while artificial GRNs have been applied to the field of GP previously [10], the phenotypes have been limited to simple signals. By exploiting the generative power of grammars, these systems can be extended to have more varied phenotypic products, allowing their application to a broader range of problem domains.

Addressing this, an artificial GRN model was adapted into the TAGE algorithm. Different methods of extracting output from the GRN model for mapping using a grammar were examined. The results obtained show that the inclusion of a grammar is not detrimental to performance of the GRN in the simplest case, reproducing results previously published in the literature [10], with more complex grammars highlighting the system's ability to produce and operate effectively with a more complex genotype-phenotype mapping.

Future work includes the further study of the model, in particular the time complexity of the algorithm, i.e., the computational cost of initialising the GRN and processing the output proteins for mapping. The method will also be compared against other approaches to the pole balancing problem, such as [7], as well as harder instances of the problem. In addition, the system will be applied to to other dynamic control problems as the uniquely inherent dynamism provided by the GRN could prove to be beneficial.

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

References

[1] Banzhaf, W.: Artificial regulatory networks and genetic programming. In: Genetic Programming Theory and Practice, ch.4, pp. 43–62. Kluwer (2003)
[2] Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, & Cybernetics, 834–846 (September- October 1983)
[3] Gilbert, S.F.: Developmental Biology, 8th edn. Sinauer Associates Inc. (2006)
[4] Harding, S., Miller, J.F., Banzhaf, W.: SMCGP2: self modifying cartesian genetic programming in two dimensions. In: GECCO 2011: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, July 12-16, pp. 1491–1498. ACM, Dublin (2011)
[5] Hoang, T.H., Essam, D., McKay, R.I.B., Hoai, N.X.: Developmental evaluation in genetic programming: The TAG-based frame work. International Journal of Knowledge-Based and Intelligent Engineering Systems 12(1), 69–82 (2008)
[6] Joshi, A., Schabes, Y.: Tree-Adjoining Grammars. Handbook of Formal Languages, Beyond Words 3, 69–123 (1997)
[7] Khan, M.M., Khan, G.M., Miller, J.F.: Evolution of neural networks using cartesian genetic programming. In: IEEE Congress on Evolutionary Computation (CEC 2010), July 18-23. IEEE Press, Barcelona (2010)
[8] Lopes, R.L., Costa, E.: ReNCoDe: A Regulatory Network Computational Device. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 142–153. Springer, Heidelberg (2011)
[9] 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 Press, Barcelona (2010)
[10] Nicolau, M., Schoenauer, M., Banzhaf, W.: Evolving Genes to Balance a Pole. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 196–207. Springer, Heidelberg (2010)
[11] O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, Genetic programming, vol. 4. Kluwer Academic Publishers (2003)
[12] Spector, L., Stoffel, K.: Ontogenetic programming. In: Genetic Programming 1996: Proceedings of the First Annual Conference, pp. 394–399. MIT Press, USA (1996)
[13] Whitley, D., Dominic, S., Das, R., Anderson, C.W.: Genetic reinforcement learning for neurocontrol problems. Machine Learning 13, 259–284 (1993)

Analysing the Effects of Diverse Operators in a Genetic Programming System

MinHyeok Kim[1], Bob (RI) McKay[1], Kangil Kim[1], and Nguyen Xuan Hoai[2]

[1] Seoul National University, Korea
[2] Hanoi University, Vietnam
{rniritz,rimsnucse,kangil.kim.01,nxhoai}@gmail.com
http://sc.snu.ac.kr

Abstract. Some Genetic Programming (GP) systems have fewer structural constraints than expression tree GP, permitting a wider range of operators. Using one such system, TAG3P, we compared the effects of such new operators with more standard ones on individual fitness, size and depth, comparing them on a number of symbolic regression and tree structuring problems. The operator effects were diverse, as the originators had claimed. The results confirm the overall primacy of crossover, but strongly suggest that new operators can usefully supplement, or even replace, subtree mutation. They give a better understanding of the features of each operator, and the contexts where it is likely to be useful. They illuminate the diverse effects of different operators, and provide justification for adaptive use of a range of operators.

Keywords: Evolutionary Operator, Tree Adjoining Grammar, Genetic Programming, TAG3P, Fitness, Tree Size, Tree Depth

1 Introduction

Many diverse genetic operators are used in evolutionary computation. In classical Genetic Programming (GP) [11], and most other tree-based GP systems (e.g. Context Free Grammar (CFG) GP [19]) there is less flexibility: constraints on tree structures restrict the available operators, and most of those since defined (e.g. [12,17]) are sub-operators of Koza's. However a number of GP systems do permit more varied operators: for example, linear GP representations such as Grammatical Evolution (GE) and Gene Expression Programming (GEP), which has led to claims that these more varied operators support better search, with some level of supporting analysis [8,3]. Similarly, Tree Adjoining Grammar-Guided GP (TAG3P) [6] permits greater structural flexibility than other tree representations. The resulting range of operators were claimed in [6] to be both diverse and beneficial, but beyond some tailoring of operators to specific problems, there has been little analysis. One is entitled to scepticism. Perhaps the new operators merely overlap each other in functionality, without greatly changing the search. We aim to characterise the effects of these operators, determining the extent of their problem specificity. This forms part of a wider stream of work,

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 387–396, 2012.

investigating the combination of a diverse range of operators with operator rate self-adaptation. It focuses on TAG3P [10] as an example of a much wider range of such algorithms.

There is already extensive research in Genetic Algorithms (GA) into how genetic operators affect individuals and move them in the solution space [14]. Related research in GP has been more limited [5], and restricted to a fairly narrow range of operators. Thus the (potentially) more diverse operators of TAG3P form an interesting field of study in its own right. In GA, the issues to consider are relatively limited – because the complexity of individuals does not change, the main interest lies in how fitness distributions change under the effect of operators. While this is also important in GP, there are other important dimensions of change, notably the change in complexity [13], as measured for example by individual size and/or depth. Our objective is to study these effects.

In section 2 we provide additional background on previous studies of operator effects in GP, and also on genetic operators supported by TAG3P. Section 3 details the methods we used, including the experimental regime and parameters. Section 4 presents the results of the experiments. In section 5 we discuss their implications, concluding in section 6 with the assumptions and limitations of our study, a summary of the general conclusions, and directions for further work.

2 Background

2.1 Genetic Operators and the Solution Space

Evolutionary algorithms operate on a search space, moving individuals toward optima by applying genetic operators such as crossover and mutation. GA researchers have analysed the effect of operators in the search space (e.g. [20,14]). The flexible chromosome structure makes this more complex in GP; nevertheless a schema theory has been derived (e.g. [18,16] and others [5] have presented a new operator-based distance measure for GP, implicitly analysing the effects of genetic operators on individuals in the search space.

2.2 Genetic Operators in Tree Adjoining Grammar Guided GP

Tree Adjoining Grammar-Guided GP (TAG3P) is a grammar-guided GP [6], based on Tree Adjoining Grammars (TAG) [9]. It is based on the adjunction operator, which models the way elements such as adjectives ('big', 'black') and phrases ('preening its fur') – β trees in TAG terminology – may be inserted into basic sentences ('The cat sat on the mat') – α trees – so as to generate new, more complex sentences ('The big black cat sat on the mat preening its fur'). TAG3P's key property is flexibility: the representation is less constrained than other tree-based GP systems.[1] Thus it is possible to extend typical GP operators to TAG3P,

[1] Specifically, it is always possible to delete any subtree from a TAG3P tree while retaining its feasibility, and it is always possible to adjoin to any unoccupied adjunction site; this property is not shared by other GP tree representations.

but also to define a range of further genetic operators, including some based more directly on classical GA operators, and others modelling features of biological genetic phenomena. In this respect, it resembles linear GP representations more than it does other tree-based GP systems.

Typical TAG3P Genetic Operators include:

1. Size-Fair Crossover (X) is directly analogous to that in other tree-based GP, permitting exchange of random subtrees with roots having matching nonterminals, and (in size-fair form [12]) of the same size.
2. Subtree Mutation (M) selects a random point in the tree, deletes the subtree below it, and replaces it with a subtree built with the initialisation algorithm.
3. Duplication and Truncation (D/T) randomly choose a node. In duplication, the subtree below is copied to a matching location in the individual; in truncation, it is removed. These operators have opposite size and depth biases, so they are used paired: Duplication or truncation is randomly chosen with 0.5 probability. They are useful for coarse adjustment.
4. (Point) Insertion and Deletion (I/D) randomly choose a node. Insertion selects an open node (a location that has not been adjoined), randomly chooses a matching β tree, and adjoins it. Deletion chooses a closed node and deletes its child. As with the duplication and truncation operators, they are used paired. They are useful for fine-tuning the size of an individual.
5. Relocation (R) disconnects a random subtree from the tree, and randomly re-adjoins it at another open location with the same label. By design, it is a deterministically size-fair operator.
6. Replication copies a parent to its child, preserving it for the next generation.

3 Methods

This work aims to characterise the effect on fitness, size or depth of the various evolutionary operators. The change depends on the state of the system, hence we wanted to see how that change itself varied over the course of an evolutionary run. We did this by conducting typical GP runs. At each generation, in addition to the normally-created children which were actually used in the evolutionary run, we generated extra children simply to evaluate the effects of the different operators, but not otherwise used in the run.

In each generation, we took 200 additional samples for each operator (in addition to those used for evolution) – of the same order as the number of real trials of each operator in a generation. We selected the parents for these trials using the selection mechanism. Thus we were examining the children actually reachable after selection.

3.1 Test Problems

We used a family of symbolic regression problems [11] and *Lid* problems [4,15,1]. In the symbolic regression problems, the target was a polynomial $F_n = \Sigma_{i=0}^{n} x^i$,

$n \in \{3, 6, 9, 12, 15\}$. We evaluated candidates f on 20 random points $X \subset [-1, 1]$; f is a *hit* if, for some predefined $\epsilon, \forall x \in X : |F_n(x) - f(x)| < \epsilon$. The objective was, using $\{+, -, \times, \div, \sin, \cos, \exp, \log\}$, to construct a hit, with fitness $(f) = \sum_{x \in X} |F_n(x) - f(x)|$. Among *Lid* problems, we used the Majority and Order problems. The target for Majority is a tree in which the number of nodes P_i is larger than of nodes N_i for all i; the target for Order is a tree in which there is a P_i before each N_i, in preorder traversal, for all i. All trees in these problems are binary trees, using only one function $\{JOIN\}$ and $2n$ terminals $\{P_i, N_i : i = 1 \ldots n\}$. The fitness function is {the number of i satisfying the condition}. We used $n \in \{25, 30\}$ for both problems.

3.2 Experimental Settings

Figure 1 shows the elementary trees defining the TAG grammar used by TAG3P [7,6]; we ran 100 trials for each problem. Table 1 shows the detailed parameter settings. Typical GP systems use high rates of crossover and lower rates of other operators for best performance. But our aim was to examine the behaviour of the system; a high crossover rate would imply low rates for other operators. We used a compromise rate of 0.5 for crossover, other operators 0.1 in creating the 'normal' children that were actually used in evolution.

Table 1. Experimental Settings (Left: Symbolic Regression; Right: Lid)

Target Function	$F_3, F_6, F_9, F_{12}, F_{15}$	$M_{25}, M_{30}, O_{25}, O_{30}$
Fitness Cases	20 Random Points from $[-1, 1]$	
Fitness Function	Sum of MAE of cases	# of satisfied i
Success Predicate	Error <0.01 on all fitness cases	n fitness value
Function Set	$+, -, \times, \div, \sin, \cos, \exp, \log$	JOIN
Terminal Set	X	$P_1, ...P_n, N_1, ...N_n$

Generations	50	Population Size	500	Tournament Size	3

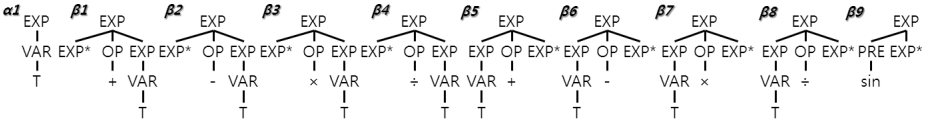


Fig. 1. TAG ElementaryTrees for Experiments for Symbolic Regression

Table 2. Overall Problem Performance

Problem	Success	Hit Time	Problem	Success	Hit Time	Problem	Success	Hit Time
F_3	100%	1.45	F_{12}	7%	48.84	M_{30}	12%	48.15
F_6	57%	30.96	F_{15}	6%	48.72	O_{25}	97%	23.11
F_9	23%	45.03	M_{25}	28%	42.98	O_{30}	80%	33.72

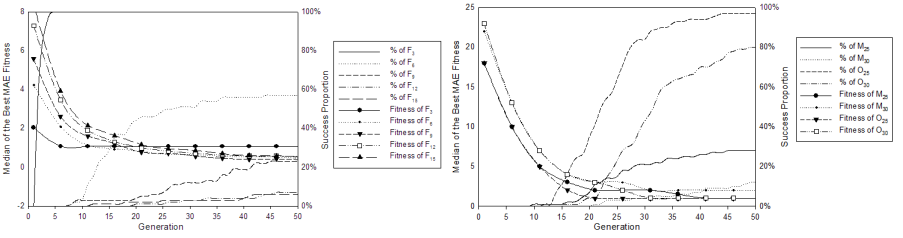


Fig. 2. General Performance

4 Results

4.1 Overall Performance

Before analysing operator effects, we first present an overview of the system performance. Figures 2 and Table 2 illustrate the performance (hit time is the first generation that a hit occurs). Figures 2 shows both the median (over all runs) of the best fitness (left axis), and the cumulative success rate (right axis). System performance on all problems decreases as n increases: the problems become tougher with n, but at a decreasing rate. The fitness curves are typical for evolutionary processes – an initial steep fall to about generation 10, then gradual convergence. The cumulative success curves are also typical, with little success at first, an increasing rate in the middle region, and a final tailing off. Based on this, to condense the immense amount of data generated, we divided the generations into three stages: begin (1-10), middle (11-25), end (26-50).

4.2 Detailed Analyses

We conducted detailed analyses on all experiments, but can only show F_9 and O_{30} due to space. F_9 is intermediate in difficulty and typical of both extremes, while O and M problems behaved similarly to each other. F_9 and O_{30} are sufficient to summarise the general trends, though we will mention some more detailed observations when appropriate. For brevity, we denote a plot for function X_m calculated from the fittest $n\%$ of children as $X_m^{n\%}$, with $n \in \{10, 30, 50, 70, 90\}$ and $X \in \{F, M, O\}$. The figures show how the genetic operators change the properties of individuals in each learning stage. The horizontal bars indicate means over 100 runs, while the vertical lines show their standard deviations.

All plots show how each operator changes the specific property for individuals (the difference between child and parent values – for fitness, negative values indicate improvement). Replication is omitted because it deterministically has no effect.

Fitness Analysis: The results in Fig. 3 overall reflect our understanding of evolutionary behaviour: the operators have a larger range of effect in early search (they are more exploratory), whereas later on, elite children resemble their parents much more.

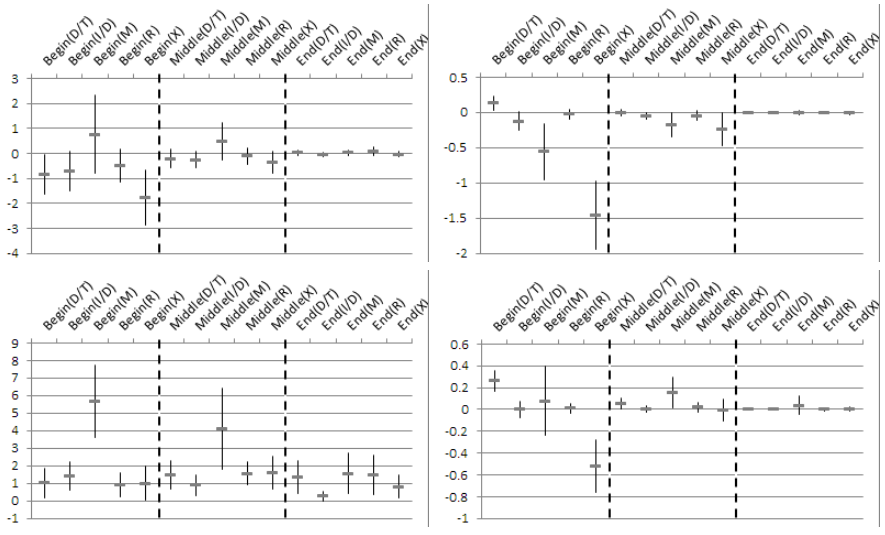


Fig. 3. Fitness Change for Selected Parents
Top: 30% Elite; Bottom: 70% Elite; Left: F_9; Right: O_{30}

The most notable differential effect in Fig. 3 is the much larger range of effect of the traditional M and X operators: the new TAG3P operators have a much smaller overall range of effect, suggesting that they are much less exploratory. In the early stages, X is on average much more beneficial than mutation – for F_9, most of the 30% elite children are an improvement on their parents, while much fewer M children are; any benefit from M comes from rarer positive mutations. While M is overall constructive for problem O_{30}, it is still substantially less so than X. However the effect of X rapidly diminishes, especially for O_{30}; M remains effective longer.

I/D are generally beneficial in early stages (the 30% elite see some worthwhile improvement on their parents. I/D retains small but very slightly beneficial effect until the end stages, befitting its proposed role as a fine-tuning operator.

D/T behave similarly to I/D on F_9, though any beneficial effect disappears by the end stages. Their effect on O_{30} is rather different, being slightly damaging in the early stages of search, very slightly beneficial in the mid stages, and losing all effect at the end.

R throughout has a relatively small effect, disappearing almost entirely by the end stages (deterministically, it had no effect in the majority problem, since it cannot change the fitness).

Size Analysis. While we saw different trends between 30% and 70% elite children in the fitness plots, there was no such difference for size – size effects were independent of child fitness; we display the results for the 50% elite. R and X do not change size at all, so we omit them from discussion.

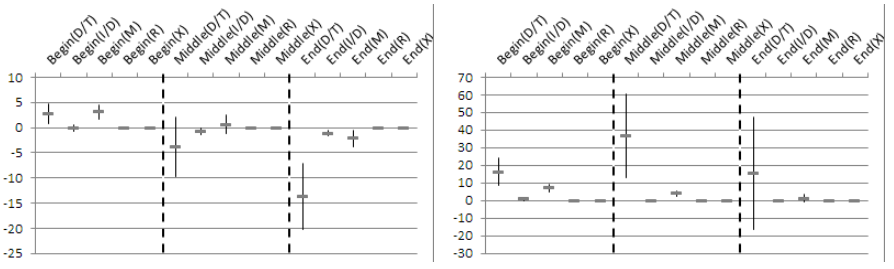


Fig. 4. Size Change, Left: $F_9^{50\%}$; Right: $O_{30}^{50\%}$

D/T generally causes a size change over the run (Fig. 4), with the scale increasing gradually. However the effect is reversed between the problems: D/T decreases size for F_9 but increases it for O_{30} (similar, but less pronounced, effects were seen with other operators). The difference may be because most individuals were near the size bound in F_9, so that many larger duplications would fail, while most truncations would succeed, introducing a bias.

M began by slightly increasing the size of individuals, but the scale decreased to zero for O_{30}, and M eventually became reducing for F_9. I/D (by design) made only very small size changes throughout.

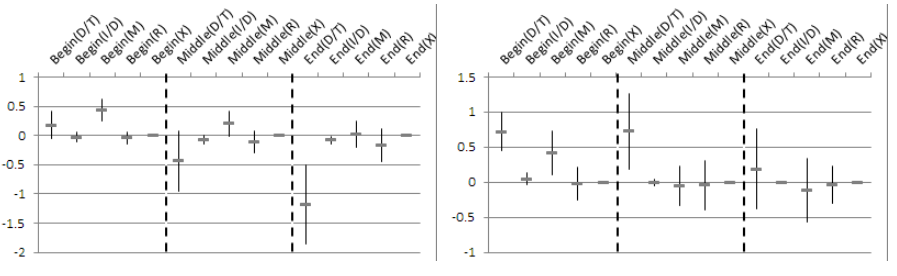


Fig. 5. Depth Change, Left: $F_9^{50\%}$; Right: $O_{30}^{50\%}$

Depth Analysis. We omit analysis of X because, as with size, most operator applications result in no change in depth, so there is little to see.

The general trends are similar to size (Fig. 5), but on a reduced scale (because of the logarithmic relationship between depth and size). The shapes of the plots are generally very similar. The only exception is with operator R, which shows a slight bias toward depth reduction, increasing in scale over time.

5 Discussion

From the perspective of fitness change, crossover appears to be the most effective operator at the start of a run. However, insertion/deletion may be preferable at

the end because of its ability to fine-tune results. On the other hand, subtree mutation is not particularly effective when we consider fitness change. It has very low probability of improving individuals; but it does cause the biggest changes in fitness. It appears from these results that other operators (not available in most tree-based GP systems) may be more effective. Duplication/truncation does theoretically similar work to insertion/deletion: it adds or deletes a sub-tree in the individual. Because insertion/deletion uses just one elementary tree, but duplication/truncation uses a sub-tree, duplication/truncation seems to work similarly but with a larger step size. On the other hand, there are no obviously strong points for relocation in terms of fitness: It may not have much effect on learning, at least in these problems.

Size and depth effects appear to be largely independent of fitness.This may have some implications for theories of the cause of GP bloat. There is little difference between the operators in their effects on size and depth (except for operators specifically designed not to affect them). At first, all operators that are free to do so increase size and depth. At the end of a run, however, the reverse occurs, and the operators decrease size and depth in the symbolic regression problem. Our hypothesis at this point is, in interacting with the size bound, the genetic operators and selection reach an equilibrium – selection increasing size and depth, with the genetic operators decreasing them.

While mutation caused the greatest change in fitness, duplication/truncation led to the biggest changes in both size and depth, while relocation was able to change depth without affecting size. Thus when a problem requires structural change, duplication/truncation and relocation may be useful, but they can be correspondingly wasteful on problems such as Order.

6 Conclusions

6.1 Summary

We investigated the roles genetic operators play and what they are useful for. We confirmed that crossover is an effective operator in the early stages of GP, but it is not effective throughout a run. Subtree mutation, another well known operator, causes large changes in fitness, even in the middle of a run, but the changes are generally negative. Insertion/deletion may be a useful alternative, leading to smoother fitness search – It is effective for fine-tuning, but at the risk of getting stuck in local optima. Duplication/truncation and relocation may be useful when structural change is needed, but can also have negative effects on poorly-matched problems.

More generally, we may conclude that there is value in having a diverse range of operators: they really do perform different tasks, either in different problems, or at different times in the evolution of solutions for the same problem. Since we will not, in general, have a priori knowledge of which operator is most suitable a any specific time, this motivates and justifies research into operator adaptation in evolutionary algorithms in general, and in GP in particular.

6.2 Assumptions and Limitations

Our study was limited to TAG3P; conclusions are likely to extend to other more flexible GP representations, but have limited relevance to standard expression tree or CFG-based GP. While the problems considered are very different, yet generally yielded similar results, they may extend to other problem domains, but a wider sample would be desirable in future. Although we saw something of the gross structural effect of operators (on size and depth), we did not investigate their effect on tree shape in detail.

6.3 Future Directions

Our extensions will have two main directions. We will look more closely at the shapes of solutions, to better understand operator effects, including working with structure-only problems such as Daida's *Lid* problem [2]. We also aim to extend our analysis to a range of other GP test problems.

Acknowledgements. 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. 2012-004841), and the BK21-IT program of MEST. The fourth author was partly funded by The Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant 102.01-2011.08 for doing this work.

References

1. Daida, J., Li, H., Tang, R., Hilss, A.: What Makes a Problem GP-hard? Validating a Hypothesis of Structural Causes. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2724, pp. 1665–1677. Springer, Heidelberg (2003)
2. Daida, J.M., Bertram, R.R., Stanhope, S.A., Khoo, J.C., Chaudhary, S.A., Chaudhri, O.A., Polito, J.A.I.: What makes a problem gp-hard? analysis of a tunably difficult problem in genetic programming. Genetic Programming and Evolvable Machines 2, 165–191 (2001), doi:10.1023/A:1011504414730
3. Ferreira, C.: Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence. Springer (2002)
4. Goldberg, D.E., O'Reilly, U.-M.: Where Does the Good Stuff Go, and Why? How Contextual Semantics Influences Program Structure in Simple Genetic Programming. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 16–36. Springer, Heidelberg (1998)
5. Gustafson, S., Vanneschi, L.: Operator-Based Distance for Genetic Programming: Subtree Crossover Distance. In: Keijzer, M., Tettamanzi, A.G.B., Collet, P., van Hemert, J., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 178–189. Springer, Heidelberg (2005)

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, Australia (2004)
7. Hoai, N., McKay, R.: A framework for tree adjunct grammar guided genetic programming. In: Proc. Postgraduate Conf. on Computer Science, Canberra, Australia, pp. 93–99 (2001)
8. Hugosson, J., Hemberg, E., Brabazon, A., O'Neill, M.: Genotype representations in grammatical evolution. Applied Soft Computing 10(1), 36–43 (2010)
9. Joshi, A.K., Levy, L.S., Takahashi, M.: Tree adjunct grammars. Journal of Computer and System Sciences 10(1), 136–163 (1975)
10. Kim, M., McKay, R.I.B., Kim, D.K., Nguyen, X.H.: Evolutionary Operator Self-adaptation with Diverse Operators. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 230–241. Springer, Heidelberg (2012)
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
12. Langdon, W.B.: Size fair and homologous tree crossovers for tree genetic programming. Genetic Programming and Evolvable Machines 1(1-2), 95–119 (2000)
13. Langdon, W., Soule, T., Poli, R., Foster, J.: The evolution of size and shape. In: Advances in Genetic Programming, vol. 3, ch.8, pp. 163–190. The MIT Press, Cambridge (1999)
14. Moraglio, A., Poli, R.: Topological Interpretation of Crossover. In: Deb, K., Tari, Z. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 1377–1388. Springer, Heidelberg (2004)
15. O' Reilly, U., Goldberg, D.: How fitness structure affects subsolution acquisition in genetic programming. Genetic Programming 98, 269–277 (1998)
16. Poli, R., McPhee, N.: General schema theory for genetic programming with subtree-swapping crossover: Part i. Evolutionary Computation 11(1), 53–66 (2003)
17. Tackett, W., Carmi, A.: The unique implications of brood selection for genetic programming. In: World Congress on Computational Intelligence, Orlando, FL, USA, vol. 1, pp. 160–165 (June 1994)
18. Whigham, P.A.: A schema theorem for context-free grammars. In: IEEE Conf. on Evolutionary Computation, vol. 1, pp. 178–181. IEEE Press (November 1995)
19. Whigham, P., et al.: Grammatically-based genetic programming. In: Proc. Workshop on Genetic Programming: From Theory to Real-World Applications, July 9, pp. 33–41. Ueniversity of Rochester (1995)
20. Wineberg, M., Oppacher, F.: Distance between Populations. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 1481–1492. Springer, Heidelberg (2003)

Quantitative Analysis of Locally Geometric Semantic Crossover

Krzysztof Krawiec and Tomasz Pawlak

Institute of Computing Science, Poznan University of Technology, Poznań, Poland
{kkrawiec,tpawlak}@cs.put.poznan.pl

Abstract. We investigate the properties of locally geometric semantic crossover (LGX), a genetic programming search operator that is approximately semantically geometric on the level of homologous code fragments. For a pair of corresponding loci in the parents, LGX finds a semantically intermediate procedure from a library prepared prior to evolutionary run, and creates an offspring by using such procedure as replacement code. LGX proves superior when compared to standard subtree crossover and other control methods in terms of search convergence, test-set performance, and time required to find a high-quality solution. This paper focuses in particular the impact of homology and program semantic on LGX performance.

Keywords: genetic programming, semantic crossover, homology.

1 Introduction

In a broad sense, a program is a sequence of symbols (instructions), where each symbol has a particular, given a priori, *semantics*. The semantics of an instruction determines its *effect*: how its output should be determined (computed) from a given input. A human programmer familiar with programming language knows that semantics and can use it to anticipate combined behavior of two concatenated instructions or substituting one instruction with another. However, the algorithms considered in genetic programming (GP) have no access to such information. From their perspective, a program is a purely symbolic structure, where opcodes associated with particular instructions have no particular *meaning*.

On one hand, this is consistent with the evolutionary aspect of GP: in the end, natural evolution does not 'know' the phenotypic expression of genes. On the other hand, the knowledge of semantics is definitely one of factors that makes human programming so effective. Therefore, equipping GP algorithms with some semantic extensions can lead to substantial progress in automated programming, and this opportunity attracted notable interest in recent GP research [4,8,11].

In [3] and [5] we proposed methods that make GP alert to certain semantic aspects of programs. The locally geometric semantic crossover (LGX, [5]) finds a semantic approximation of an intermediate ('medial') procedure for a pair of procedures (subtrees) located in parent programs and uses it as a replacement for the parent programs. In this follow-up study, we investigate the properties of this method, in particular the impact of homology on its performance.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 397–406, 2012.

2 The Method

The proposed approach exploits the *compositional* character of programs, by which we mean that not only complete programs, but also parts of programs and program conglomerates have valid interpretation in many programming languages. We also assume that the fitness function captures the divergence between program output and some known desired output. This is consistent with GP standards, where individuals are usually tested on a set of fitness cases, and fitness is some form of error built upon the outcome of these tests. Formally, a *metric* $||\cdot||$ calculating such error is given. The consequence of this assumption is a *convex surface* of fitness landscape, spanned over the space of vectors holding program outputs. Convexity allows designing recombination operators that are likely to yield offspring of good quality [10,3]. This is easy to demonstrate for Euclidean metric: given a point x corresponding to the desired output of a program and a pair of points x_1, x_2 representing parent solutions, any point on the segment between x_1 and x_2 cannot be further from x than $\max(||x_1 - x||, ||x_2 - x||)$.

The proposed method exploits this property *locally*, i.e., on level of program fragments, rather than entire programs. It operates in two major phases. Prior to evolutionary run, it creates a library of short programs, calculates their semantics and builds upon them an index for fast access. Then, during an evolution, the library is used by the crossover operator to modify the fragments of parents' programs in a semantically-aware way.

Building the Library of Procedures. The input to the method is a set of instructions I, each of them being an operator of arbitrary arity. The first step consists in creating from I a library L of short programs, called *procedures* in following discussion. The library is purposed to provide semantically diverse code fragments for the crossover operator. The choice of procedures in L can be done along different criteria, but here, for simplicity, L contains all trees of height at most h.

Next, the *semantics* $s(p)$ of every procedure $p \in L$ is calculated. Throughout this paper, by semantics we mean a vector of d outcomes produced by a program for all d inputs (fitness cases). Any two procedures p_1, p_2 that have the same semantics ($||s(p_1), s(p_2)|| = 0$) do the same thing, which is redundant from the viewpoint of the method. Therefore, we discard from L procedures that duplicate the semantics of other procedures, leaving only the shortest ones.

Indexing the Library. The semantics $s(p)$ of a procedure p is a point in a d-dimensional space. Distances between such points reflect the semantic differences between procedures. Semantically similar procedures are located close to each other, while the very different ones occupy distant positions.

To efficiently search this space for procedures that are as close as possible to an arbitrarily selected point, we employ *spatial index*, a data structure designed for geographic databases. As in this study we limit our interest to symbolic regression, the space of consideration is Euclidean and the semantic distance $||\cdot||$ becomes a norm, which allows us to employ the *R-trees* [1].

Locally Geometric Semantic Crossover. After the library is built and equipped with an R-tree index, a GP run is launched. It proceeds as regular GP,

except for employing a homologous crossover operator, termed *locally geometric crossover* (LGX). Given two parent programs p_1, p_2, LGX first identifies the *structurally common region* for them, which is defined as in one-point crossover by Poli and Langdon [13], i.e., a set of node locations (loci) that occur in both parents. The common region can be considered as an intersection of the parents, where the opcodes are ignored – only the tree structure matters (taking the opcodes into account would often render the common region almost empty). The subtree can embrace at most all locations in both parents, but typically it is smaller.

Next, LGX selects a random location (locus) in the common subtree, intended to serve as crossover point. This choice follows the same rules as in the canonic Koza-style GP [2]: an internal node is selected with probability 0.9 and a leaf with probability of only 0.1, to reduce bloat. Subsequently, LGX identifies the subtrees p'_1 and p'_2 rooted in the selected location in p_1 and p_2, respectively. As they are independent executable programs, their semantics $s(p'_1)$ and $s(p'_2)$, are known (technically: cached during individuals' evaluation), which allows us to determine the midpoint between them in the semantic space:

$$s_m = \frac{s(p'_1) + s(p'_2)}{2} \tag{1}$$

This point represents the semantics of a hypothetical procedure $p : s_m = s(p)$, which, when inserted into parents at the appointed location, would make the resulting offspring programs semantically intermediate at the point of crossover (cf. [3]). However, finding p in general requires solving an inverse problem $p = s^{-1}(s_m)$, which is a separate program induction problem in itself. Also, as s_m is a combination of semantics of two, potentially big trees, it may not be represented by a program available within the assumed program space.

This is where the library comes at help. Instead of looking for a procedure whose semantics is exactly s_m, we find in L the procedure that is semantically *most similar* to s_m, i.e.:

$$p = \arg\min_{p' \in L} ||s(p') - s_m|| \tag{2}$$

Finding p is facilitated using the R-tree index. The procedure p replaces then the subtrees p'_1 and p'_2 in the parent solutions, which so become the two offspring. This step concludes the crossover act.

Properties of the Approach. An important property of the proposed approach is *completeness*. As the library contains representatives of *all* semantic equivalence classes obtainable from given set of procedures, LGX can produce any tree. The semantic search space is not constrained.

The computational overhead compared to the standard GP approach is the sum of the time required to prepare the library (generation of procedures, calculation of semantics, elimination of semantic duplicates, and construction of an R-tree) and the time of querying the R-tree in LGX. According to [12], the worst-case complexity of the latter component is linear w.r.t. the number of objects (here: library size $|L|$), but usually the query time is significantly lower. This

cost depends also on the number of fitness cases and the number of procedures to be stored in the library, which in our case is a function of h. For large h, it can be substantial, thus, to keep the computational cost at bay, we use $h \in \{3, 4\}$.

Related Research. The past contributions that have something common with the approach presented here can be grouped according to two features: the use of a library and the semantically-aware modification of solutions. Concerning the former, LGX can be likened to run transferable libraries [14], which are repositories of program fragments intended to be used across multiple GP runs applied to different problem instances. However, [14] does not involve semantics. Concerning the latter, McPhee *et al.* were probably the first to study the impact of crossover on program semantics and so-called semantic building blocks [8]. In [9], 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.* [11] swaps a pair of subtrees in parent solutions that have similar, yet not too similar, semantics.

In the context of these contributions, LGX remains unique in combining three elements: the choice of program fragments w.r.t. their semantic properties, homologous character of crossover, and the use of a library of procedures.

3 The Experiment

The experiment is aimed at verification whether the semantic properties of LGX influence the efficiency of GP search. The experimental framework is symbolic regression, with instructions $\{+, -, \times, /\}$ and a terminal representing independent variable x. Semantics is defined as a vector of values returned by a program for 20 fitness cases distributed equidistantly in the interval $[-1, 1]$.

We consider two libraries, for the maximum procedure height $h \in \{3, 4\}$. For $h = 3$, there are 81 procedures, but only 38 of them are semantically distinct, so $|L| = 38$. For $h = 4$, these figures amount to 21385 and 1697, respectively.

We examine LGX with two types of control setups. The first of them is standard Koza-style GP [2], which involves conventional tree-swapping crossover that uses the same probability distribution as LGX for node selection (0.1 for leafs and 0.9 for internal nodes). Like other considered operators, it never replaces the root node. The latter, called RX (random crossover), is intended to verify if the observed results are due to the geometric character of crossing over performed by LGX. To certain extent, RX operates as LGX (Section 2), however its choice of procedure from L is purely random. Thus, RX is similar to LGX in terms of mode of operation, but it is completely blind to the structure of semantic space.

To sum up, there are 5 setups in total: canonical GP, RX and LGX for $h \in \{3, 4\}$, further referred as GP, RX_3, RX_4, LGX_3 and LGX_4.

We solve 6 univariate symbolic regression problems shown in Table 1: 3 polynomials and 3 rational functions taken from [6]. For each configuration, 150 runs are carried out, each starting from different initial population of size 1024 and lasting for 250 generations. Fitness is minimized and defined as the absolute error of the output produced byof an individual w.r.t. the desired output, summed for

Table 1. Test problems

Problem	*Definition (formula)*	*Problem*	*Definition (formula)*
Sextic	$x^6 - 2x^4 + x^2$	*R1*	$(x+1)^3/(x^2 - x + 1)$
Septic	$x^7 - 2x^6 + x^5 - x^4 + x^3 - 2x^2 + x$	*R2*	$(x^5 - 3x^3 + 1)/(x^2 + 1)$
Nonic	$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	*R3*	$(x^6 + x^5)/(x^4 + x^3 + x^2 + x + 1)$

Table 2. The absolute error with 0.95 confidence interval, committed by the best-of-run individuals on *training set* (mean of 150 runs)

	Sextic	*Septic*	*Nonic*	*R1*	*R2*	*R3*
GP	0.002 ±0.001	0.159 ±0.040	0.122 ±0.041	0.441 ±0.102	0.231 ±0.040	0.184 ±0.026
RX_3	0.002 ±0.001	0.130 ±0.039	0.128 ±0.040	0.175 ±0.039	0.130 ±0.028	0.164 ±0.027
LGX_3	0.003 ±0.001	0.109 ±0.034	0.114 ±0.041	0.170 ±0.033	0.086 ±0.020	0.179 ±0.024
NHX_3	0.001 ±0.001	0.138 ±0.047	0.085 ±0.039	0.292 ±0.073	0.145 ±0.033	0.141 ±0.030
RX_4	0.005 ±0.002	0.102 ±0.024	0.130 ±0.035	0.140 ±0.035	0.101 ±0.019	0.076 ±0.014
LGX_4	0.001 ±0.001	0.044 ±0.011	0.043 ±0.009	0.061 ±0.014	0.041 ±0.013	0.028 ±0.007
NHX_4	0.002 ±0.001	0.084 ±0.022	0.063 ±0.014	0.102 ±0.022	0.060 ±0.014	0.045 ±0.010
GP_{time}	0.002 ±0.001	0.102 ±0.031	0.070 ±0.024	0.210 ±0.041	0.137 ±0.028	0.085 ±0.015

the 20 fitness cases. The selection method is tournament of size 7, crossover likelihood is 0.9 and reproduction likelihood is 0.1. There is no mutation involved. Other parameters are set to defaults used in the ECJ package [7], which served as experimental environment.

Search Progress. Figure 1 presents the fitness of best-of-generation individuals averaged over 150 runs, along with 0.95-confidence intervals shown as shading. LGX_4 is an unquestionable winner in terms of speed of convergence, while LGX_3 makes much slower progress. This may be explained by the fact that the library it uses is almost two orders of magnitude smaller than that of LGX_4 (38 vs. 1697 procedures). As a consequence, the semantic diversity of the procedures inserted into offspring (the number of unique semantic) is here much lower, which deteriorates the algorithm's ability to perform effective exploration.

The fact that LGX outperforms RX is the main result of this study. It demonstrates that introducing 'medial' tendency in crossover makes the search process converge faster towards good solutions. It is particularly remarkable when we recall that LGX never affects the root node. Therefore, the effects of geometric-aware changes introduced into deeper tree nodes must propagate to its root, and, on average, improve the fitness of offspring more than for the other methods. This confirms the conclusion of our former study that dealt with a problem of more discrete nature [3].

Last but not least, the confidence intervals for LGX are much narrower than those for the other methods. The behavior of this method is thus much more predictable, and, in convenient circumstances, it should be possible to estimate the expected number of generations required to attain an assumed fitness level.

Importance of Homology. LGX adds two elements to standard subtree crossover: homology and the semantically geometric choice of procedures. Its

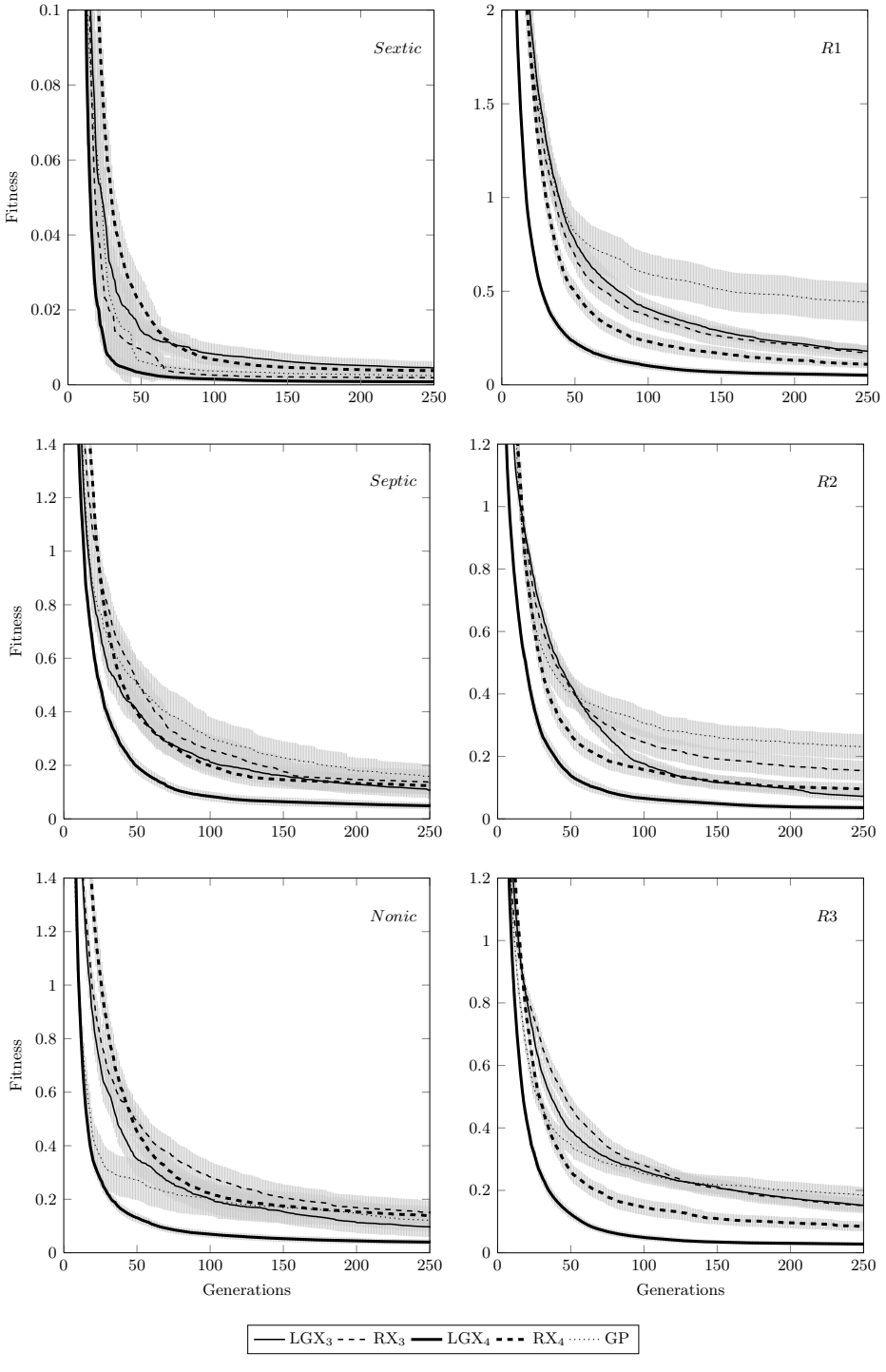


Fig. 1. Best-of-generation fitness graphs averaged over 150 evolutionary runs

superiority of LGX to RX demonstrates that the latter is essential. However, would LGX perform equally well if it was not homologous? To settle this issue, we prepared an additional control setup that uses a non-homologous but locally geometric crossover operator (NHX). NHX mimics LGX except for the choice of loci to be affected, where it works as the standard tree-swap crossover, i.e., selects them in both parents randomly and independently. Then it finds the semantically most medial procedure in the library and inserts it into both parents.

Table 2 compares the final (end-of-run) fitness of best-of run individuals evolved by NHX with the other methods. For $h = 3$ (small library), the duel between NHX and LGX is inconclusive, methods win or lose depending on the problem. However, for $h = 4$ the conclusion is clear: LGX yields lower error rate and not worse variance than NHX. Homology is then an essential component for this operator that significantly contributes to its performance.

Impact on Tree Size. By being homologous, LGX can be expected to affect also tree size. Figure 2 depicts the mean number of nodes per individual, calculated over all individuals in populations and averaged over 150 runs. The results are very similar across all benchmark problems. The methods using large library ($h = 4$) suffer from substantial bloat that is more severe than for the small library ($h = 3$). This can be easily explained. The mean tree depth of procedures in the library is greater for $h = 4$ than for $h = 3$. On average then, every act of crossover brings more genetic material to the population in the former case.

Another observation following from Fig. 2 is that RX suffers from bloat more than LGX. This suggests that the semantically close-to-geometric procedures inserted by LGX are on average shorter than the procedures selected from the library at random by RX. Our explanation for this phenomenon pertains to the relation between lengths of procedures and their location in the semantic space. Typically, short procedures will have semantics of small magnitudes, as it is unlikely to produce large values using arithmetic instructions that operate on numbers form interval $[-1, 1]$ (with obvious exception of the division operator). Such semantics will crowd closely around the origin of semantic space. On the contrary, longer procedures are capable of producing larger output values, which correspond to semantics that are distant from the origin. Also, every long procedure that is semantically equivalent to a shorter procedure is discarded when the library is being built (see Sec. 2). LGX, which looks for procedures that are semantically medial with respect to parents' subtrees (cf. s_m in Eq. (2)), is more likely to generate s_m that is close to the origin of semantic space. As a consequence, it selects shorter procedures more frequently.

Test-Set Performance. To assess the generalization capability of the considered methods, we employed a test set composed of 20 cases drawn randomly from the interval $[-1, 1]$, with uniform distribution. The best-of-run individual for each run is executed on these cases, and its generalization capability is expressed in the same terms as for the training process – the absolute error.

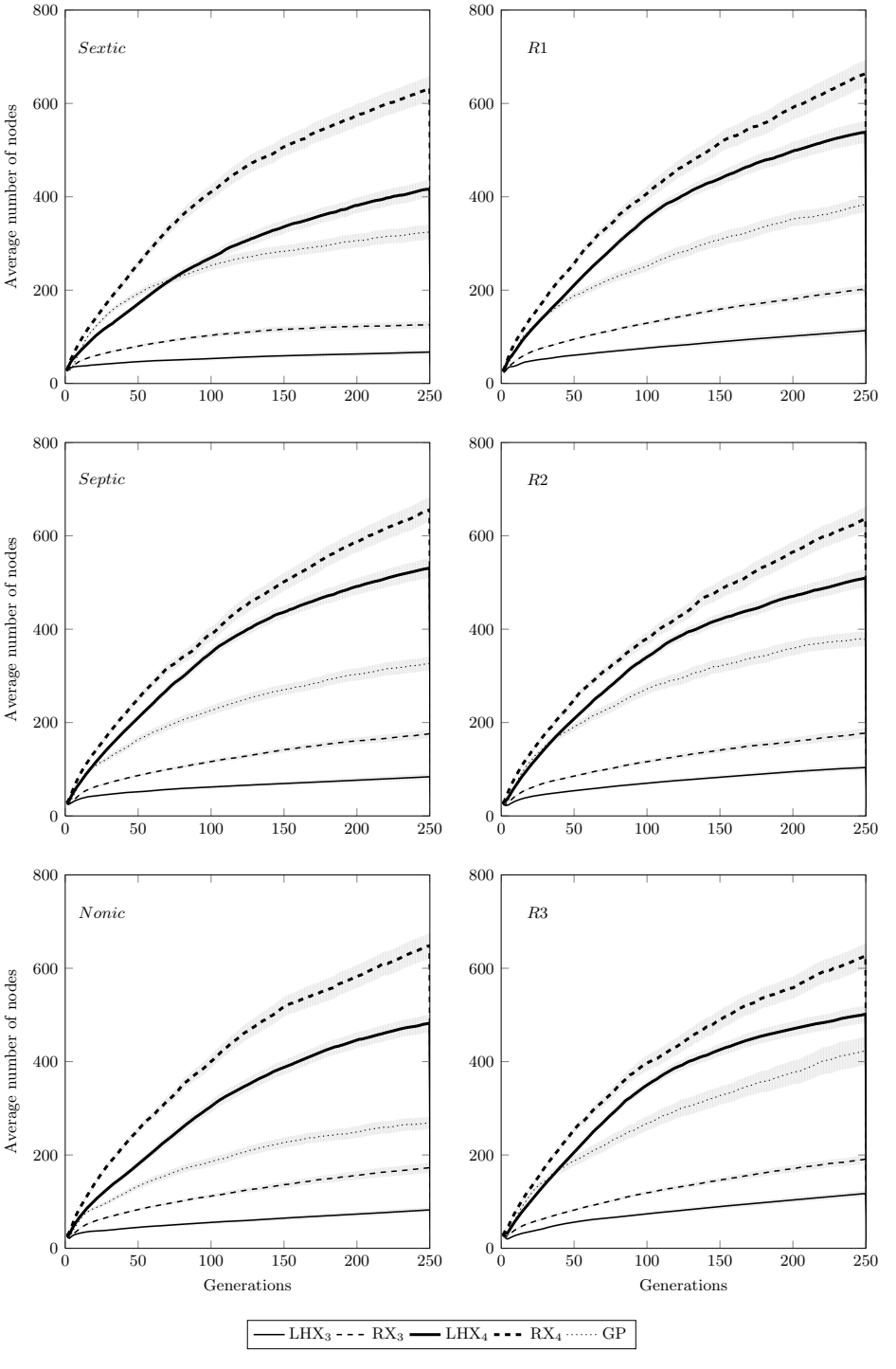


Fig. 2. Number of nodes (population mean) averaged over 150 evolutionary runs

Table 3. The absolute error with 0.95 confidence interval, committed by the best-of-run individuals on *test set* (mean of 150 runs)

	Sextic	*Septic*	*Nonic*	*R1*	*R2*	*R3*
GP	0.009 ±0.007	0.233 ±0.068	0.182 ±0.070	0.483 ±0.120	0.302 ±0.109	0.230 ±0.040
RX_3	0.004 ±0.002	0.140 ±0.040	0.146 ±0.047	0.191 ±0.044	0.129 ±0.028	0.167 ±0.037
LGX_3	0.005 ±0.002	0.122 ±0.038	0.117 ±0.045	0.226 ±0.069	0.086 ±0.020	0.163 ±0.020
RX_4	0.029 ±0.031	23.443 ±42.628	0.262 ±0.102	8.025 ±14.853	0.196 ±0.055	0.316 ±0.149
LGX_4	0.003 ±0.002	0.116 ±0.032	0.099 ±0.026	0.102 ±0.027	0.077 ±0.032	0.103 ±0.029

Table 3 presents the test-set errors of the best-of-run individuals averaged over all runs. Comparison of these figures with fitness values achieved on the training set (Table 2) leads to conclusion that all methods suffer from overfitting. In terms of the ratio of test-set error to training-set error, GP is superior. However, in absolute terms, LGX_4 attains the lowest error on the test set for all problems and has also the lowest variance. This is particularly interesting, because LGX_4 yields substantially bigger trees than GP (Fig. 2).

Time Complexity. The benefits of LGX come at extra computational cost of creating the library and searching for the semantically similar procedures. While the former turns out to be low (1–2 seconds compared to 100–150 seconds of the cost of entire run), the latter cannot be ignored. The roughly $250 \times (1024/2) \times 0.9 = 115,200$ R-tree queries per run make LGX substantially slower. In effect, its overall runtime is on average 2.8 times longer than GP's. This, together with the curves in Fig. 1, urges us to ask the question: will LGX maintain its superiority to GP with same time allocated to both methods?

To verify this possibility, we conducted an additional experiment, which consisted in giving GP the same amount of time as corresponding LGX runs took. By comparing the results of these runs, presented in the last row of Table 2 (GP_{time}) with the final fitness values of LGX, we conclude that GP, despite having more time, cannot catch up LGX_4, although it manages to reach the performance level of LGX_3 for some problems. Therefore, LGX can be considered attractive not only from theoretical viewpoint, but also in practical perspective.

4 Conclusion

The main conclusion of this study is that search operators that are at the same time homologous and semantically medial can improve the efficiency of GP search and cause the evolved programs generalize better. The experimental analysis suggests that both these properties are essential. It can be hypothesized, though remains to be verified that, with time of evolution, LGX causes emergence of a common semantic blueprint in the population, with the subprograms located at particular loci specializing at solving certain subproblems of the original problem. This hypothesis sounds very attractive, as it implies a capability for discovering semantic modules in the structure of the problem, which in turn could provide the possibility of problem decomposition.

LGX has been presented and verified here in the context of symbolic regression, but it has wider applicability. Any domain for which semantics are computable and a semantic metric is available, can be subject to this approach. In particular, if the metric $||\cdot||$ is not a norm, there are alternative ways in which a crossover can be made semantically medial [3].

Acknowledgment. Work supported by grant no. DEC-2011/01/B/ST6/07318.

References

1. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. ACM SIGMOD Conf., p. 47, Boston, MA (June 1984); Reprinted in Stonebraker, M.: Readings in Database Sys. Morgan Kaufmann, San Mateo, CA (1988)
2. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
3. Krawiec, K.: Medial Crossovers for Genetic Programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 61–72. Springer, Heidelberg (2012)
4. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Raidl, G., et al. (eds.) GECCO 2009: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, Montreal, July 8-12, pp. 987–994. ACM (2009)
5. Krawiec, K., Pawlak, T.: Locally geometric semantic crossover. In: GECCO 2012: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, Philadelphia, PA, USA. ACM Press (accepted, July 2012)
6. Krawiec, K., Wieloch, B.: Automatic generation and exploitation of related problems in genetic programming. In: IEEE Congress on Evolutionary Computation (CEC 2010), July 18-23. IEEE Press, Barcelona (2010)
7. Luke, S.: The ECJ Owner's Manual – A User Manual for the ECJ Evolutionary Computation Library, zeroth edition, online version 0.2 edition (October 2010)
8. 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)
9. Moraglio, A., Krawiec, K., Johnson, C.: Geometric semantic genetic programming. In: Igel, C., et al. (eds.) The 5th Workshop on Theory of Randomized Search Heuristics, ThRaSH 2011, Copenhagen, Denmark, July 8-9 (2011)
10. Moraglio, A., Poli, R.: Topological Interpretation of Crossover. In: Deb, K., Tari, Z. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 1377–1388. Springer, Heidelberg (2004)
11. 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)
12. Papadopoulos, A., Manolopoulos, Y.: Performance of Nearest Neighbor Queries in R-Trees. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 394–408. Springer, Heidelberg (1996)
13. Poli, R., Langdon, W.B.: Schema theory for genetic programming with one-point crossover and point mutation. Evolutionary Computation 6(3), 231–252 (1998)
14. Ryan, C., Keijzer, M., Cattolico, M.: Favorable biasing of function sets using run transferable libraries. In: O'Reilly, U.-M., et al. (eds.) Genetic Programming Theory and Practice II, May 13-15, ch.7, pp. 103–120. Springer, Ann Arbor (2004)

Length Scale for Characterising Continuous Optimization Problems

Rachael Morgan and Marcus Gallagher

School of Information Technology and Electrical Engineering,
University of Queensland, Brisbane 4072, Australia
r.morgan4@uq.edu.au, marcusg@itee.uq.edu.au

Abstract. In metaheuristic optimization, understanding the relationship between problems and algorithms is important but non-trivial. There has been a growing interest in the literature on techniques for analysing problems, however previous work has mainly been developed for discrete problems. In this paper, we develop a novel framework for characterising continuous optimization problems based on the concept of *length scale*. We argue that length scale is an important property for the characterisation of continuous problems that is not captured by existing techniques. Intuitively, length scale measures the ratio of changes in the objective function value to steps between points in the search space. The concept is simple, makes few assumptions and can be calculated or estimated based only on the information available in black-box optimization (objective function values and search points). Some fundamental properties of length scale and its distribution are described. Experimental results show the potential use of length scale and directions to develop the framework further are discussed.

Keywords: Continuous optimization, Problem properties, Problem characterisation, Fitness landscape analysis.

1 Introduction

A continuous optimization problem with a simple symmetric boundary constraint is to find a solution vector $\mathbf{x}^*$ such that:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{S} \tag{1}$$

where $\mathcal{S} = [b_l, b_u]^n \subseteq I\!R^n$. Given a metaheuristic algorithm, a standard question is how well will the algorithm perform at solving a given problem (in other words, how well-suited is the algorithm to the problem)?. For some types of problems, it is possible to answer these questions rigorously (e.g. if f is convex, smooth and differentiable, then Newton-based algorithms converge rapidly towards the optimum). However if little can be assumed (e.g. f is a 'black-box' problem), then the questions are much more difficult to answer. Metaheuristics utilise multiple heuristics, complex models and randomness, while problems may be high-dimensional, noisy, or have features such as many local optima or

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 407–416, 2012.

other complex structures. The relationship between problems and algorithms in practice is the result of interactions between these factors.

Metaheuristics research has been dominated by the development of algorithms, but a recent focus has been to better understand both the relationship between algorithms and problems, and the nature of the problems themselves. For example, fitness landscape analysis has produced a theoretical framework and techniques for studying problems. However, this work has mainly been developed for discrete or combinatorial problems.

In this paper, we develop a framework for characterising continuous optimization problems based on the concept of *length scale*. While previous techniques from fitness landscape analysis can be applied, we argue that length scale is a critical concept for continuous problems that is not captured by these techniques. Sec. 2 reviews previous work on problem analysis with a focus on applicability in the continuous case. In Secs. 3 and 4 we define and develop the notion of problem length scale and its distribution. Illustrative experiments are provided in Sec. 5, and discussion is given in Sec. 6.

2 Discrete and Continuous Fitness Landscape Analysis

The notion of f as an (n-dimensional) 'fitness' landscape defined over $\mathcal{S}$ has been widely used as a model in evolutionary biology and computation. A fitness landscape is defined using f and a graph, G representing $\mathcal{S}$ (i.e $\mathcal{S}$ is discrete). Edges in G can be defined by a move operator and induce a neighbourhood in $\mathcal{S}$. Properties of the landscape can be defined in this framework, e.g. a (strict) local optimum is a point $\mathbf{x}'$ where all neighbours have a fitness worse than $f(\mathbf{x}')$. For a discrete $\mathcal{S}$, it is possible to determine whether or not $\mathbf{x}'$ is a local optimum by exhaustive evaluation of its neighbours. For all but very small problem sizes, enumeration of the landscape is impractical. Fitness landscape analysis typically uses random, statistical or other sampling methods to obtain points of interest (and/or their fitness values) from a landscape. Examples include the distribution of f values (density of states), fitness distance correlation (FDC) between the sample and a point (typically the global optimum), autocorrelation and correlation length statistics of random walks in $\mathcal{S}$. Information content aims at quantifying landscape ruggedness based on transitions observed in f values [15,14]. Previous research has also focussed on problem-specific techniques to characterise properties of combinatorial problems such as the travelling salesman problem. Comprehensive reviews of fitness landscape and problem analysis techniques can be found in [9,10,13].

If $\mathcal{S}$ is continuous, landscape features conceptually similar to the discrete case can be defined mathematically (as suggested in [9]), but evaluating them in practice is problematic. Each solution has an infinite number of neighbours in theory, yet a finite but extremely large number in practice due to finite-precision floating-point representation. Another significant difference between discrete and continuous landscapes is tied to the *distance* between points in $\mathcal{S}$ (using some metric). For a discrete landscape, the minimum possible distance will occur between a point and one of its neighbours, with a finite set of possible distance

values between all points in G. For a continuous landscape, the minimum distance between points can be made arbitrarily small (in practice until the limit of precision is reached) and the number of possible distance values is infinite.

The reason for these difficulties lies in the difference between continuous and discrete problems. Consider a combinatorial problem with binary representation, $S = [0, 1]^n$. To solve the problem is to determine whether each variable x^i in the solution vector should take the value 0 or 1. A metric (e.g. Hamming distance) can be defined, but there is no notion of the *scale* of x^i. For a continuous problem however, finding an appropriate scale for each x^i is critical (e.g. does the objective function vary in a significant way with changes in x^i of order 10^3? 10^{-3}? 10^{-30}?). Fitness landscape techniques originating from the assumption of a discrete $\mathcal{S}$ do not capture such information because it is not relevant for the discrete case.

Despite these issues, there have been some adaptations of landscape analysis to continuous problems. Gallagher calculated FDC for the training problem in multi-layer perceptron neural networks [4]. For the learning tasks considered (student-teacher model), the global optimum is known, however this would not normally be the case for such a problem. Points were sampled from within a specified range around the global optimum. Wang and Li calculate FDC in the context of a continuous NK-landscape model and on some standard test functions [16]. Müller and Sbalzarini [7] analyse the CEC 2005 benchmark function set using FDC on points uniformly sampled from $\mathcal{S}$. While these results show interesting structure and differences between problems, the limitations of FDC noted for discrete problems remain (e.g. [7] concludes that FDC alone is not sufficient for problem design or measuring difficulty).

Dispersion is a recently-proposed problem metric [6] which measures the average distance between pairs of high quality solutions. Quality is determined by sampling points and retaining a percentage with the best fitnesses (according to a specified threshold). Dispersion is shown to be a useful metric in studying the performance of CMA-ES on a number of functions. Dispersion makes only limited use of the f values of points via the threshold used to produce the sample. Pairwise distances between solutions have also been analysed in samples of apparent local minima for multi-layer perceptron training [4].

In summary, there are some important limitations of existing techniques for the analysis of continuous problems, stemming from the adaptation of techniques developed for discrete problems and/or a limited use of the available information from sampling solution and their fitness values.

3 Length Scale in Optimization

We aim to develop a framework to study the topological/structural characteristics of a problem landscape independent of any particular algorithm. Importantly, the framework should utilise all information available in the black-box optimization setting, be estimated easily from data and be amenable to statistical and information theoretic analysis.

Definition 1. *Let* $\mathbf{x}^i$ *and* $\mathbf{x}^j$ *be two distinct solutions in the search space* ($\mathbf{x}^i \neq \mathbf{x}^j$) *with corresponding objective function values* $f(\mathbf{x}^i)$ *and* $f(\mathbf{x}^j)$. *The* ***length scale****,* r*, is defined as:*

$$r : [0, \infty) = \frac{|f(\mathbf{x}^i) - f(\mathbf{x}^j)|}{\|\mathbf{x}^i - \mathbf{x}^j\|} \tag{2}$$

The length scale intuitively measures how much the objective function value changes with respect to a step between two points in the search space. In this paper, we use Euclidean distance, however any appropriate metric can be used. Length scale is defined simply as a magnitude over a finite interval in the search space: directional information about a step from $\mathbf{x}^i$ to/from $\mathbf{x}^j$ is not considered.

Our definition of r is related to the *difference quotient* (also known as *Newton's quotient* and is a generalisation of *finite difference* techniques) from calculus and numerical analysis. The difference quotient is defined as $\frac{f(x+h)-f(x)}{h}$, and can be used to estimate the gradient at a point x, as $h \to 0$ [8]. Implementations of gradient-based algorithms utilise approximations of this form if the gradient of f is not available. Finite difference methods are widely used in the solution of differential equations, but are not directly related to this paper. Length scale is also related to the *Lipschitz constant*, defined as a constant, $L \geq 0$, where $|f(\mathbf{x}^i) - f(\mathbf{x}^j)| \leq L\|\mathbf{x}^i - \mathbf{x}^j\|, \forall \mathbf{x}^i, \mathbf{x}^j$ [17]. However, r does not assume that f is continuous and captures information about *all* rates of change of f over $\mathbf{x}$.

In some cases, it is possible to derive a simple expression for the length scale of a problem, as illustrated by the following examples.

Example 1. *1-D linear objective function*

Given $f = ax$ *where* $(x, a \in \mathbb{R})$, *the length scale between* $\mathbf{x}^i$ *and* $\mathbf{x}^j$ *is:*

$$\begin{aligned} r &= \frac{|f(\mathbf{x}^i) - f(\mathbf{x}^j)|}{\|\mathbf{x}^i - \mathbf{x}^j\|} \\ &= \frac{|ax^i - ax^j|}{|x^i - x^j|} \\ &= |a| \end{aligned}$$

For this function, r captures the intuition that any step in $\mathcal{S}$ will be accompanied by a proportional change in f. The length scale of any finite set of samples from the search space (e.g. the points visited by an optimization algorithm) is invariant to the location(s) in $\mathcal{S}$ or the order in which the points were taken. The length scale of a (n-D) neutral or flat landscape is also a special case of this.

For most continuous problems, r will not be a constant over $\mathcal{S}$. In different regions of the space, the length scale value will depend on the local topology of the fitness landscape (varying slope, basins of attraction, ridges, saddle points, etc.).

Example 2. *1-D quadratic objective function*

Given $f = ax^2$ *where* $(x, a \in \mathbb{R})$*, the length scale between* $\mathbf{x}^i$ *and* $\mathbf{x}^j$ *is:*

$$\begin{aligned} r &= \frac{|f(\mathbf{x}^i) - f(\mathbf{x}^j)|}{\|\mathbf{x}^i - \mathbf{x}^j\|} \\ &= \frac{\|ax^{i2} - ax^{j2}\|}{\|x^i - x^j\|} \\ &= \frac{|a|\|(x^i - x^j)(x^i + x^j)\|}{\|x^i - x^j\|} \\ &= |a|\|x^i + x^j\| \end{aligned}$$

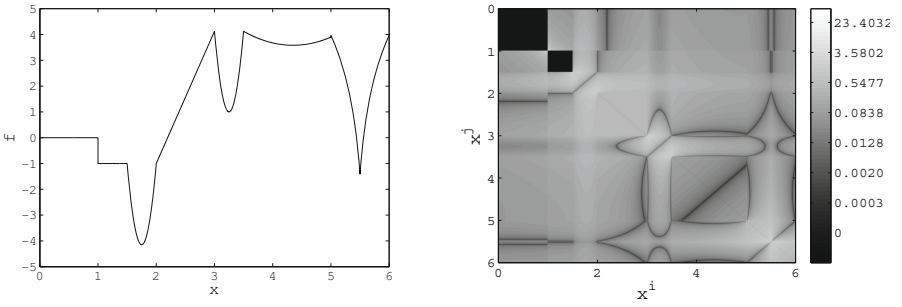
Here, steps between points that are relatively close to the optimum result in relatively small length scales compared to the same-sized steps further from the optimum. This suggests that an algorithm needs to reduce the size of the steps it makes to successfully approach the optimum of this function (e.g. gradient descent). To illustrate the richness of length scale information we construct an artificial 1-D function with a variety of different topological features.

Example 3. *1-D 'mixed-structure' function defined as follows and shown in Fig. 1(a).*

$$f(x) = \begin{cases} -1 \text{ if } 1 \leq x < 1.5 \\ 50(x - 1.75)^2 - 4.15 \text{ if } 1.5 \leq x < 2 \\ 5.125x - 11.25 \text{ if } 2 \leq x < 3 \\ 50(x - 3.25)^2 + 1 \text{ if } 3 \leq x < 3.5 \\ 0.75(x - 4.35)^2 + 3.583 \text{ if } 3.5 \leq x < 5 \\ 3\log(|x - 5.6|) + 5.5 \text{ if } 5 \leq x < 5.5 \\ 3\log(|x - 5.4|) + 5.5 \text{ if } 5.5 \leq x < 6 \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

Since f is a 1-D problem $(x \in [0, 6])$ it is possible to enumerate length scales over the entire search space to a certain level of numerical precision. Fig. 1(b) shows the length scales calculated between pairs of points, x^i, x^j, at increments of 10^{-3} across $\mathcal{S}$. We have coloured the values using a logarithmic scale to better visualise magnitudes of change. The plot is symmetric across the diagonal, which follows from the definition of r. The thin black line along the diagonal is approximately a zero-length step $(x^i = x^j)$. The two flat regions of the function produce black squares where $r = 0$. The dark lines and curves in the plot show steps in the space where $f(x^i) \approx f(x^j)$, e.g. moving from a point on one side of a basin or funnel to a point on the other side of the minimum at the same height. Within the plot, it can be seen that components of the function combine to produce different patterns and gradients of r values.

Overall, it is clear that r reflects the structure of f: if f has complex structure then this will also be captured in r. In addition, Fig. 1(b) gives an indication of how much variety is contained in the search points and f values that an algorithm encounters as it attempts to search a landscape effectively.



(a) (b)

Fig. 1. (a) 1-D 'mixed-structure' function. (b) Enumeration of length scales in the 1-D 'mixed-structure' function.

4 Length Scale Distribution

Length scale values produce information about problem structure. While it is possible to enumerate r over a large set of values for a 1-D problem, this is clearly infeasible for higher dimensions. One possibility is to summarise the values of r that occur over a given landscape. A good summary of r may be usable to predict the values of r we will see if further exploration of the landscape is conducted (particularly if the sampling technique is the same).

Definition 2. *Consider r as a continuous random variable. Then, let the* ***length scale distribution*** *be defined as the probability density function $p(r)$.*

Consider again Example 1. Since $r = |a|$, $p(r)$ is a Dirac delta function:

$$p(r) = \begin{cases} 1 \text{ if } r = |a| \\ 0 \text{ otherwise} \end{cases}$$

It follows that the n-D flat function also results in a Dirac delta function with a spike at $r = 0$.

Now reconsider Example 2. Let Z be the sum of two independent, continuous uniform random variables bounded by $[b_l, b_u]$. This produces a triangular distribution [5]:

$$p_Z(z) = \begin{cases} \frac{z-2b_l}{(b_u-b_l)^2} \text{ if } 2b_l \leq z \leq (b_l + b_u) \\ \\ \frac{2b_u-z}{(b_u-b_l)^2} \text{ if } (b_l + b_u) < z \leq 2b_u \\ \\ 0 \text{ otherwise} \end{cases}$$

The length scale distribution for Example 2 is the absolute value of $p_Z(z)$:

$$p(r) = |p_Z(r)| = p_Z(r) + p_Z(-r), \forall r \geq 0$$

Therefore, $p(r)$ of the 1-D quadratic function is a 'folded' triangular distribution.

While $p(r)$ can be derived for some functions, in general it can be approximated using probability density estimation based on r values sampled from the landscape (see Sec. 5). The length scale distribution is not unique for a problem but will vary depending on the structure present in that problem.

We can utilise concepts from information theory to compare landscapes via their length scale distributions. Shannon entropy is used as a measure of the uncertainty of a random variable [2]. Entropy measures the expected amount of information needed to describe the random variable. The entropy of $p(r)$ is:

$$h(r) = -\int_0^\infty p(r)\log_2\big(p(r)\big)dr \tag{4}$$

We conjecture that problems with structure of similar complexities should yield a similar $h(r)$, and hence, $h(r)$ is potentially very useful for categorising problems. The Dirac delta function has the smallest entropy of all density functions, meaning the n-D flat and 1-D linear functions minimize $h(r)$. The uniform density function (in a bounded region) has the largest entropy of any other density function bounded within the same region. To obtain a uniform $p(r)$, there must be length scales of uniformly varying size, e.g. random noise functions. Therefore, random noise functions maximize $h(r)$. This results in two extreme values of $h(r)$ that any given landscape is within.

5 Length Scales of the BBOB'10 Test Functions

In this section we examine length scales of the Black-Box Optimization Benchmarking 2010 (BBOB'10) test functions [3]. We aim to investigate whether or not there is a relationship between the 'difficulty' of functions (as measured by the best performing algorithms in BBOB'10) and length scale. Problems with largely varying length scales may contain a richer, more complex structure, and may be more difficult to solve. The methodology used in these experiments is general and can be easily applied to other black-box problems. Source code used is available at http://www.itee.uq.edu.au/∼uqrmorg4/length-scale-bbob.html.

We use a random Levy walk to sample $\mathcal{S}$ and corresponding f values. Levy walks generally yield good coverage of the search space at varying magnitudes of step sizes [12]. The Levy distribution pertaining to step size is parameterised by scale (γ) and location (δ) parameters, here both set to 0.001. This type of walk has frequent small steps (with 0.001 being the minimum), and infrequent large steps.

A Levy walk of 10^5 steps was conducted for each of the BBOB'10 functions. Walks were bounded by $[-5, 5]^{10}$, with proposed steps outside the boundary rejected. Length scales were calculated for each pair of solutions, producing 50005000 values of r. Kernel density estimation was then used to estimate $p(r)$. The kernel bandwidth was calculated using the 'solve-the-equation plug-in'

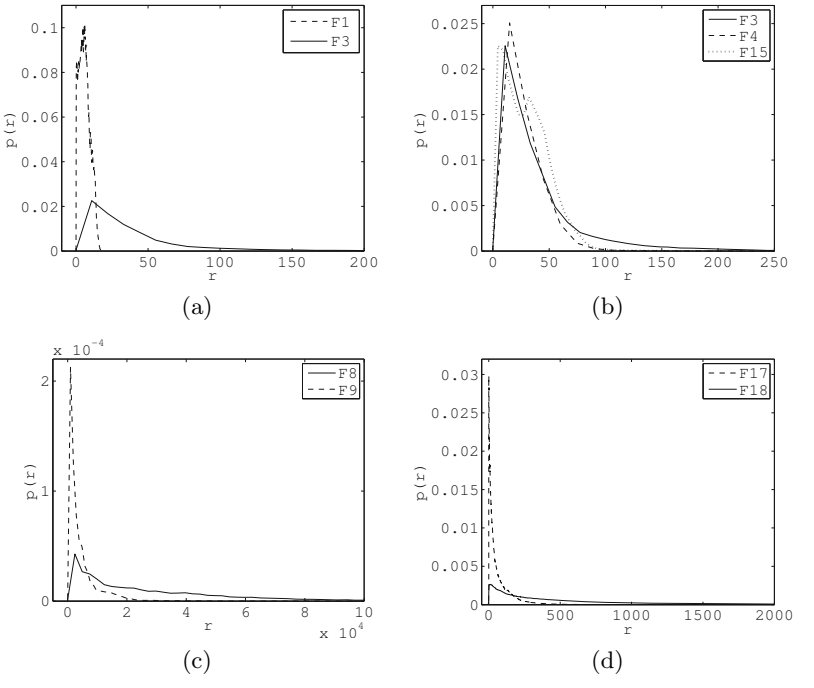


Fig. 2. Examples of similar length scale distributions estimated on BBOB'10 functions

method [11]. The resulting length scale distributions were quite varied across the problems, however there were a few notable similarities, shown in Fig. 2.

Fig. 2(a) shows $p(r)$ for Sphere (F1) and Rastrigin (F3). We do not expect these distributions to be identical, however since the global structure of F3 is F1, we observe some similarity. In Fig. 2(b), we see almost identical length scale distributions, resulting from sampling Rastrigin-like functions. The length scale distributions for Rosenbrock (F8) and Rosenbrock Rotated (F9) can be seen in Fig. 2(c). The larger peak in the F9 distribution indicates that there are more low-valued length scales. Both functions are variations of Rosenbrock and we observe similar changes in fitness, and hence similar ranges of length scales. This observation is also true for the Schaffer F7 (F17) and Shaffer F7 Moderately Ill-Conditioned (F18) distributions (Fig. 2(d)). It is clear that problems with similar structure have similar length scale distributions, while problems with vastly different structure have different length scale distributions.

To examine the relationship between r and problem difficulty, we use the expected running time (ERT) for the best-performing algorithm in the BBOB'10 results [1] as a proxy for problem difficulty. We use the results within a precision of 10^{-8} of the global optimum (e.g. the BFGS algorithm performs best on F1 with an ERT of 23, and so we use '23' to indicate problem difficulty). Given the kernel density estimate of each $p(r)$, we can estimate $h(r)$. Fig. 3 shows $h(r)$ vs ERT for the BBOB'10 functions. There is an interesting relationship

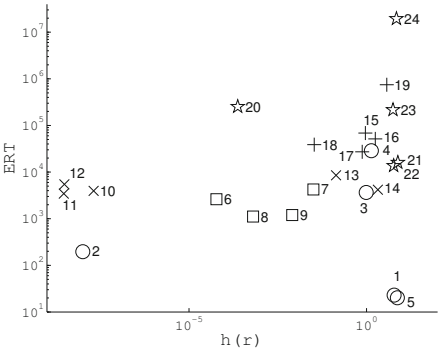


Fig. 3. Length scale distribution entropy vs ERT for BBOB'10 problems. F1 to F5 are ∘, F6 to F9 are □, F10 to F14 are ×, F15 to F19 are + and F20 to F24 are ⋆.

between $h(r)$, ERT and the function type. Separable functions are denoted by ∘; low to moderately conditioned functions by □; high and uni-modal functions by ×; multi-modal with adequate global structure by +; and multi-modal with weak global structure by ⋆. Fig. 3 clearly shows clustering of problems within categories, e.g. F10-12, F6-7 and F15-19. In fact, for problems F6-12, ERT is incapable of distinguishing the categories, while $h(p(r))$ can. This illustrates that length scales capture valuable information about problem structure.

In Fig. 3, there is a great distinction between the uni-modal (∘, □ and ×) and multi-modal functions (+ and ⋆). In general, multi-modal functions are more difficult, and so we expect ERT to separate uni-modal functions from multi-modal functions. This is observed, however we can additionally see that $h(r)$ is capable of characterising uni-modal and multi-modal functions.

6 Summary and Conclusions

We have proposed a framework for characterising continuous optimization problems using the notion of length scale and its distribution. The framework is based on utilising all available information in black-box optimization and is readily calculated using points from $\mathcal{S}$ and their f values. This paper has discussed some properties of length scale, presented examples and experimental results using the BBOB'10 competition results.

We believe that there is considerable scope for future work. It should be possible to explore the relationship between features such as landscape modality or ruggedness and the shape of $p(r)$. Entropy was used to summarise the distribution, but other ideas from statistics and information theory deserve investigation. Our experimental results assume that the sampling methodology used produces a representative sample of the search space. This requires quantification. It would be interesting to analyse different real-world and benchmark continuous problems using length scale. The set of points that an algorithm evaluates during a run could also be analysed to examine the length scales visited by the algorithm.

Acknowledgments. We thank Mike Preuss and Olaf Mersmann for making available pre-processed results data from BBOB'10.

References

1. Auger, A., Finck, S., Hansen, N., Ros, R.: BBOB 2010: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RT-388, INRIA (2010), http://hal.inria.fr/inria-00516689
2. Cover, T.M., Thomas, J.A.: Elements of information theory. Wiley Interscience, New York (1991)
3. Finck, S., Hansen, N., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Tech. Rep. 2009/20, Research Center PPE (2009)
4. Gallagher, M.: Multi-layer perceptron error surfaces: visualization, structure and modelling. Ph.D. thesis, Dept. Computer Science and Electrical Engineering, University of Queensland (2000)
5. Killmann, F., von Collani, E.: A note on the convolution of the uniform and related distributions and their use in quality control. Economic Quality Control 16(1), 17–41 (2001)
6. Lunacek, M., Whitley, D.: The dispersion metric and the CMA evolution strategy. In: GECCO 2006, pp. 477–484. ACM, New York (2006)
7. Müller, C.L., Sbalzarini, I.F.: Global Characterization of the CEC 2005 Fitness Landscapes Using Fitness-Distance Analysis. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcázar, A.I., Merelo, J.J., Neri, F., Preuss, M., Richter, H., Togelius, J., Yannakakis, G.N. (eds.) EvoApplications 2011, Part I. LNCS, vol. 6624, pp. 294–303. Springer, Heidelberg (2011)
8. Overton, M.: Numerical Computing with IEEE Floating Point Arithmetic. Cambridge University Press (2001)
9. Pitzer, E., Affenzeller, M.: A Comprehensive Survey on Fitness Landscape Analysis. In: Fodor, J., et al. (eds.) Recent Advances in Intelligent Engineering Systems. SCI, vol. 378, pp. 161–191. Springer, Heidelberg (2012)
10. Reidys, C., Stadler, P.: Combinatorial landscapes. SIAM Rev. 44(1), 3–54 (2002)
11. Sheather, S., Jones, M.: A reliable data-based bandwidth selection method for kernel density estimation. Journal of the Royal Statistical Society. Series B (Methodological) 683–690 (1991)
12. Shlesinger, M.F., West, B.J., Klafter, J.: Lévy dynamics of enhanced diffusion: Application to turbulence. Physical Review Letters 58, 1100–1103 (1987)
13. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Computers and Operations Research 39(5), 875–889 (2011)
14. Steer, K., Wirth, A., Halgamuge, S.: Information Theoretic Classification of Problems for Metaheuristics. In: Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H.A., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K.C., Branke, J., Shi, Y. (eds.) SEAL 2008. LNCS, vol. 5361, pp. 319–328. Springer, Heidelberg (2008)
15. Vassilev, V.K., Fogarty, T.C., Miller, J.F.: Information characteristics and the structure of landscapes. Evol. Comput. 8, 31–60 (2000)
16. Wang, Y., Li, B.: Understand behavior and performance of real coded optimization algorithms via NK-linkage model. In: CEC 2008, pp. 801–808. IEEE (2008)
17. Wood, G.R., Zhang, B.P.: Estimation of the Lipschitz constant of a function. Journal of Global Optimization 8, 91–103 (1996)

Analyzing the Behaviour of Population-Based Algorithms Using Rayleigh Distribution

Gabriel Luque and Enrique Alba

Universidad de Málaga, Spain,
Dpto. de Lenguajes y Ciencias de la Computación
E.T.S.I. Informática
Campus Teatinos 29071, Málaga, Spain
{gabriel,eat}@lcc.uma.es

Abstract. This paper presents a new mathematical approach to study the behaviour of population-based methods. The calculation of the takeover time and the dynamical growth curves is a common analytical approach to measure the selection pressure of an EA and any algorithm which manipulates a set of solutions. In this work, we propose a new and more accurate model to calculate these values. This new model also includes other very interesting features, such as the characterization of the complete behaviour of the methods using a single value, the Rayleigh distribution parameter. We also extend the study to consider the effect of the mutation (or in general, any neighborhood exploration operator) and we show several advanced uses of this models such as building self-adaptive techniques or comparing algorithms.

Keywords: Growth Curves, Takeover Time, Rayleigh Distribution.

1 Introduction

Optimizing (or searching or learning) is a commonly practiced sport in designing a new metaheuristic that beats others on a given problem or set of problems. This kind of experimental research finishes by establishing the superiority of a given technique over others. In this scenario, researchers should not be limited to establishing *that* one metaheuristic is better than another in some way, but also to investigate *why*, i.e., they must understand how the algorithms work.

These last studies usually are developed using mathematical tools (run-time analysis, takeover time study or landscapes analysis). In this work, we focus on the takeover time and the growth curves. This approach measures the converge time (time in which all the population is only composed by the best individual) under several assumptions (the best possible individual is in the initial population and only selection operators are employed) This approach was successfully used to analyze GA [1], and other types of population-based algorithms [2,3].

In this work, we propose a new model to calculate the growth curves and takeover time. This new model is more accurate than existing ones, and it has another important feature, it allows to characterize the behaviour of the method

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 417–427, 2012.

using a single value (the Rayleigh parameter). Later, we will also extend this model to consider a more realistic case, in which we incorporate the mutation to the method. We study how the new parameter can give us some information of the effect of mutating solutions. Finally, we will show some advanced uses of this mathematical model which can be useful for any researcher. In particular, we show how we can build a new self-adaptive technique and how we can use this model to classify and compare algorithms.

This paper is organized as follows. Section 2 is an introduction containing some preliminary background about some basic concepts, previous models for takeover time, and our proposed approach. In Section 3, we analyze the predicted takeover times provided by the models. Section 4 studies how our model can capture the effect of the mutation. Some advanced uses of the proposed model are given in Section 5. In the last section we summarize the conclusions and give some hints on the future work.

2 Growth Curves and Takeover Times

In this section, we first give a brief definition of growth curves and takeover times (Subsection 2.1). Later, in Subsection 2.2 we describe some existing models for the calculation of these vales and we finish this section analyzing our new approach and its possible advantages.

2.1 Definitions

A common analytical approach to study the selection pressure of an EA is to characterize its takeover time [1], i.e., the number of generations it takes for the best individual in the initial population to fill the entire population under selection only. The growth curves are another important issue to analyze the dynamics of the population-based methods. These growth curves are functions that associate the number of generations of the algorithm with the proportion of the best individual in the whole population. In Fig. 1 we show an example of these two concepts.

Now, we give a mathematical definition of these concepts. Let us start by formally defining what the growth curve is, since it is the basis to define the takeover time.

Definition. Given a population-based algorithm, under selection only, with an initial population containing exactly one individual of the best fitness class; the function $P_{sel} : \mathbb{N} \rightarrow [0, 1]$ that maps the proportion of copies of the best individual in the population to each generation step, is termed as the *growth curve*.

As result of applying selection only in an population-based method (without variation operators), at every generation the number of copies of the best individual potentially grows up. The number of generations it takes for the algorithm to completely fill the population is what is called the takeover time, formally defined as follows:

Definition. Let P_{sel} be the function defining the growth curve induced by a selection method, the value $t_{sel} = min\{t : P_{sel}(t) = 1\}$ is called the *takeover time* associated to the given selection method.

Several models have been proposed to estimate the takeover time for most common selection methods. We can classify the selection techniques in two categories: (a) methods which only depend on the order among the individuals in the population (order-based), and (b) techniques which take the fitness of each individuals into account (fitness-based). In this work, we focus on tournament and proportional selection, which are representative of these two cases, respectively. In next subsection, we discuss some existing mathematical models to estimate the takeover time for these two selection methods, and later, we present our proposal.

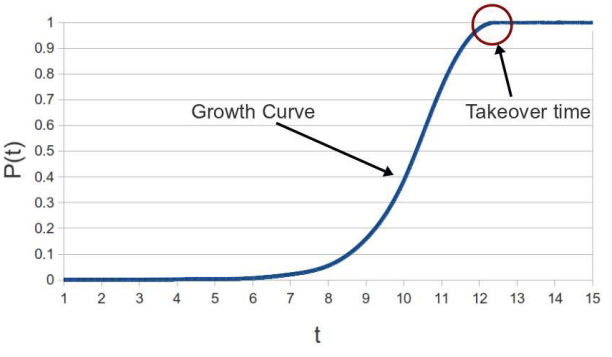


Fig. 1. Growth curve and takeover time for a population-based method using binary tournament selection method

2.2 Existing Models

G&D model. We start analyzing the theoretical models proposed in the original work of Goldberg and Deb [1]. They propose different models for each selection method. In tournament selection, two or more individuals are chosen at random for a fitness-based competition and the best of them is selected with a high probability for breeding. Their model for calculating the takeover time is:

$$t_{tour}^{G\&D} = \frac{1}{\ln s}[\ln \mu + \ln(\ln \mu)] \tag{1}$$

where μ is the population size and s stands for the *tournament size*, i.e., the number of individuals chosen for competition. You can notice that this function only depends of the population size and tournament size, which are both parameters of the algorithm.

In proportionate selection individuals are chosen according to their fitness values, so that the fittest members have a higher chance of being selected. The takeover time in this case is defined to be:

$$t_{prop}^{G\&D} = \frac{1}{c}\mu \log \mu \tag{2}$$

where c is a constant value, which is a function of some fitness-based feature of the initial population.

Logistic model. Let us continue by discussing the work of Sarma and De Jong [2]. In that work, they proposed a simple quantitative model for cellular EAs (which can also be used in panmictic scenarios) based in the logistic family of curve. In summary, the proposed equation for calculating growth curve is (3):

$$P^{LOG}(t) = \frac{1}{1 + b \cdot e^{-at}} \quad . \tag{3}$$

where a and b are growth coefficients. This approach uses the same model for any kind of selection method, since the effect of the selection technique is incorporated in the adjustable values a and b. To calculate the takeover time using this model, we can make use of the definition of takeover time of the previous subsection, and iterate this models (starting with $P(0) = 1/\mu$, being μ the population size), searching the lowest t value, which makes $P(t) \approx 1$.

Hypergraph model. Sprave [4] has proposed a unified description for any non-panmictic population structured EA, that could even end in an accurate model for panmictic populations (since they can be considered as fully connected structured populations). He modelled the population structure by means of *hypergraphs*. A hypergraph is an extension of a canonical graph. The basic idea of a hypergraph is the generalization of edges from pairs of vertexes to arbitrary subsets of vertices's.

He developed a method to estimate growth curves and takeover times. This method is based on the calculation of the diameter of the actual population structure and on the probability distribution induced by the selection operator. In fact, Chakraborty *et al.* [5] calculated the success probabilities for the most common selection operators (p_{select}), what represents an interesting complement for putting hypergraphs to work in practice. A complete description of the hypergraph model can be found in [4].

Topology model. Alba and Luque [3] proposed an accurate model for distributed evolutionary algorithms but it could be used for panmictic populations considering this case as special one of a distributed method using an appropriate migration policy. Their proposed model is the following:

$$P^{TOP}(t) = \sum_{i=1}^{i=d(T)} \frac{1/N}{1 + a \cdot e^{-b \cdot (t - per \cdot (i-1))}} + \frac{N - d(T)/N}{1 + a \cdot e^{-b \cdot (t - per \cdot d(T))}} \tag{4}$$

where $d(T)$ is the diameter of the topology, per is the migration frequency, N is the number of islands, and a and b are growth coefficients . This expression is a combination of the logistic model plus our previous model. In fact, in the

panmictic case [1] ($d(T) = 0, per = 0$, and $N = 1$), this equation is the same as the logistic one, and therefore it will not be used in the rest of the paper.

Other models. Some other models were proposed in the literature [6,7,8,9]. These models usually are very accurate ones, but they are linked to specialized algorithms or specific parameter settings. Therefore, in the rest of the paper, we only compare our proposed model against logistic, hypergraph, and Goldberg & Deb models.

2.3 Our Proposed Model

In this paper, we propose a new model for calculating the takeover time (and the growth curves) based on the cumulative function of Rayleigh distribution [10]. The Rayleigh probability density function is:

$$f(x;\sigma) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2}, \quad x \geq 0, \tag{5}$$

for parameter $\sigma > 0$, and cumulative distribution function:

$$F(x) = 1 - e^{-x^2/2\sigma^2} \tag{6}$$

for $x \in [0, \infty)$.

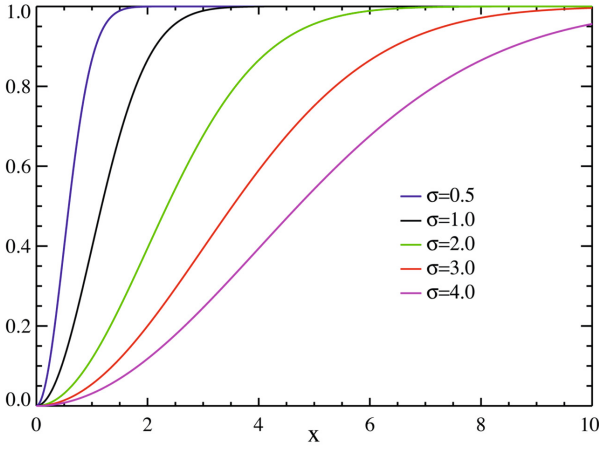


Fig. 2. Cumulative distribution function of Rayleigh distribution

If we observe the cumulative function of this distribution (see Fig. 2), we can notice that it is very similar to the growth curve shape presented in the previous section (Fig. 1) and it is controlled by a single parameter σ. This was one of the reason why we select this model. In additional, this distribution have been deeply studied and some interesting properties have been defined. Maybe, one of the most important properties (from our work point of view) is that it is possible

calculate an accurate estimation of the parameter σ with some points (N) of the Rayleigh distribution, using the next equation:

$$\hat{\sigma} \approx \sqrt{\frac{1}{2N}\sum_{i=1}^{N} x_i^2}. \tag{7}$$

Summarizing, the proposed model for the calculation of the growth curves is:

$$P^{RAY}(t) = 1 - e^{-t^2/2\sigma^2} \tag{8}$$

And the expected advantages of this models are:

- Simplicity: A single model that can be used to describe the behaviour of any population-based algorithm under any selection method (and maybe some variations operator as we will showed in Section 4).
- A single control parameter: We can describe the complete behavior of the method using a single real value.
- Ability to predict the control parameter: We do not execute the whole algorithm to calculate the σ parameter, but this value can be estimated with a relative low number of steps of the technique, maintaining a quite accurate result.

3 Accuracy of Models for Takeover Times

In this section we study the precision of the different models to predict the real values of the takeover time for binary tournament and proportional selections. We divide this section into three parts: in the first one, we describe the methodology followed to perform the experiments and the comparisons; later, we compare the accuracy of the existing models and our new approach; finally, we study the ability of our model to calculate the Rayleigh parameter (and therefore, the final takeover time) using only a small number of generations instead of all ones.

3.1 Methodology

We have performed experiments with binary tournament and proportional selection. In the experiments, we use a population of 4096 individuals ($\mu = 4096$), with $(\mu + \mu)$ strategy and the initial population is randomly generated with individual fitness between 0 and $\mu - 1$ and then we introduce a single best individual (fitness $= \mu$). In hypergraphs we have used an expected level of accuracy of $\varepsilon = 2.5 \cdot 10^{-4}$. For the actual curves we have performed 100 independent runs.

In order to compare the accuracy of the models we proceeded to calculate the mean square error (9) between the actual values and the theoretically predicted ones (where k is the number of points of the predicted curve).

$$MSE(model) = \frac{1}{k}\sqrt{\sum_{i=1}^{k}(model_i - experimental_i)^2}\ . \tag{9}$$

Using this metric, as we said in the previous section, we compare the accuracy of our proposed model against some existing one. In concrete, the comparison will be performed against the Goldberg, logistic, and hypergraph models.

3.2 Analysis of the Results

Now, we analyze the mean square error for the different models. Fig. 3 contains the error of all the models analyzed in this work for binary tournament and a proportional selection method.

Several conclusions can be obtained from this figure. First, we can notice that the error of the models predicting the takeover time value for the proportional selection is slightly larger than the mean error for tournament selection, but in both cases, the behaviour of the models is quite similar. It is clear that the hypergraph model is not able to capture the dynamics of a panmictic algorithms. This is not a surprising result since this model was proposed for structured populations, and although some configurations makes the behaviour of a non-panmictic algorithm be closer to a panmictic one, there exist still some differences as it is proved by the error that produces this model. The logistic model and the model of Goldberg and Deb obtain quite similar and accurate results, but they are significantly worse than the proposed one.

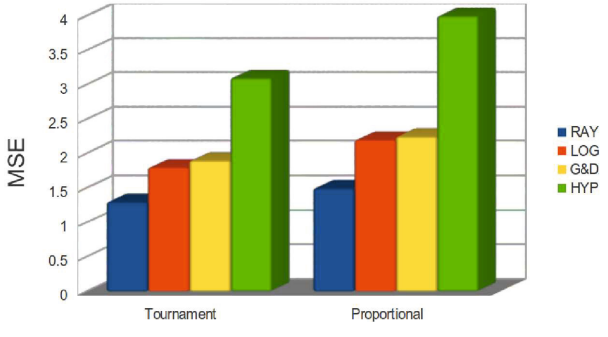


Fig. 3. Mean square error for binary tournament and proportionate selection methods

The first advantage that we expect of our model (the accuracy) is fulfilled, and therefore, in the next subsection, we analyze the second one. The second important expected advantage is the ability to estimate the σ parameter using only the results of a few generations of the method instead of using the results of the complete execution of the technique (as we do in this subsection).

3.3 Accuracy of the σ Estimation

In this section, we use the Equation 7 to estimate the σ parameter. To do it, we calculate the approximation of σ using different amount of generations (measured as a different percentage of the global execution). In Fig. 4, we show the error of our method for the different estimation.

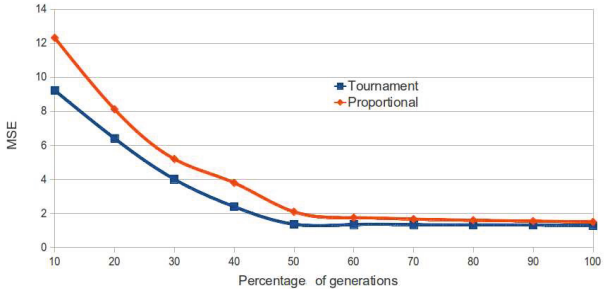


Fig. 4. Mean square error when the σ value is estimated using a different amount of points

As it was expected, the larger the number of generation used for the prediction, the more accurate the results are. We can notice that using about 50-60% of the global execution (depending of the selection method) allows to obtain a very accurate prediction, and adding new data from more generations only gets insignificant improvements. Then, for a good estimation of the σ parameters in this case we only need to run half of the expected total execution. That percentage is quite high (we expected a lower number) but it is due to the takeover time for the tested selection method is quite low (around 25-40 generations), and even a small number of generation implies a high percentage. Anyway, we have observed that this model allows to estimate the parameter controlling its behaviour without the need of completing the execution.

4 Analyzing the Effect of the Mutation

In this section, we tackle a more realistic scenario in which the mutation operator is used. In this preliminary work, we use a simple mutation operator. In concrete, we analyze the bit-flip operator with different intensity (considering this parameter as the number of bits changed in the solution by the operator). Since the calculation of the growth curves only considers increasing successions, in the experiments, we will only take into account the applications of the mutation operator in which the fitness of the resulting solution will be equal or better than the original individual. As test problem for the experiments, we will use the academic OneMax problem with a bitstring of 10000 elements.

In Fig. 5a we show the MSE of our model. We can observe that the error for all the intensities are quite small and similar to the obtained in the previous section (without mutation). Therefore, our model is able to capture successfully the mutation effect without including additional terms in the equation.

Another interesting topic is analyzing how the σ parameter varies with the different mutation intensity and if it is possible to find a relation between both parameters. This relation is shown in Fig. 5b. In that figure, we can observe that there is a clear (inverse) relation between the σ and the intensity of the mutation. This result is quite expected since a higher value of mutation intensity

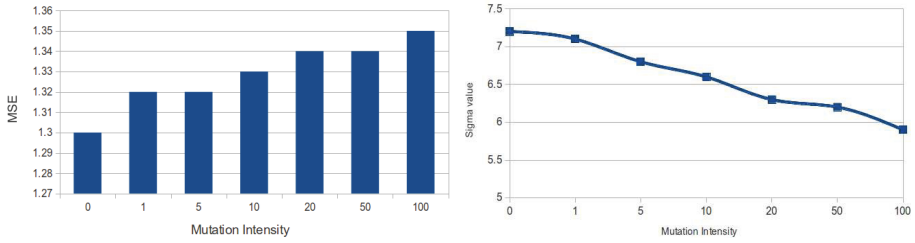


Fig. 5. (a) Mean square error of our model for different mutation intensity and (b) relation between σ parameter and mutation intensity for OneMax problem

represent a faster convergence of the method (more solutions can be the optimal one in a small number of generations), and the σ value controls the slope of the growth curve. This result is quite promising since the σ parameter can be used to summarize the configuration (behaviour) of the method in a single value, but a deeply study is needed.

5 Advanced Uses of the Proposed Model

In this section, we discuss briefly two possible advanced uses of the proposed models

- *Building self-adaptive methods*: As it can analyzed in the previous section, there exists a relation between the parameter of the method and the parameter, σ, which controls the model, and we can use that information to change the setting of the algorithm to force a specific behaviour. However, our model makes several assumptions that cannot be met in real scenarios where the method is to be applied. Therefore, our first challenge consists in adjusting the mathematical models to work for real cases but some initial results about this topic can be found in [11].
- *Comparing and classifying algorithms*: Since the parameter used by our model allows to summarize the behaviour of the complete algorithm in a single real value, it can be used as a metric to compare algorithms (combined with other existing ones as execution time, solution quality, ...) and even to detect classes of equivalence among the different values of the configuration parameters of optimization techniques.

6 Conclusions

In this paper we have performed an analysis of the growth curves and takeover regime of population-based algorithms. We compared the well-known Goldberg model, logistic model, a hypergraph model and a newly proposed model based on the cumulative function of the Rayleigh distribution. In this work we have shown how our models is able to capture the behaviour of the method, obtaining the most accurate prediction of the takeover time.

Also, we have describe how it is possible to calculate a good estimation of the parameter which controls our model without the necessity of execute the complete algorithm. We have extended this work to consider some easy variation operators (mutation one), and we have observed that our model captures its effect accurately, and it is possible to establish a clear relation between the mutation intensity and the σ parameter. Finally, we have shown several practical uses of this mathematical models: to build new algorithms and to compare (or classify) population-based techniques.

As a future work we plan to check the results presented in this paper on additional operator (even recombination ones) and problems. We can also want to perform a comprehensive study about the meaning of σ parameter, and its utilization in practical cases.

Acknowledgments. Authors acknowledge funds from the Spanish Ministry of Sciences and Innovation European FEDER under contract TIN2008-06491-C04-01 (M* project, available in URL `http://mstar.lcc.uma.es`), TIN2011-28194 (RoadME project available in URL `http://roadme.lcc.uma.es`, TIN2011-28194, and CICE Junta de Andalucía under contract P07-TIC-03044 (DIRICOM project, available in URL `http://diricom.lcc.uma.es`).

References

1. Goldberg, D.E., Deb, K.: A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In: Rawlins, G.J. (ed.) Foundations of Genetic Algorithms, pp. 69–93. Morgan Kaufmann, San Mateo (1991)
2. Sarma, J., De Jong, K.: An Analysis of Local Selection Algorithms in a Spatially Structured Evolutionary Algorithm. In: Bäck, T. (ed.) Proceedings of the 7th International Conference on Genetic Algorithms, pp. 181–186. Morgan Kaufmann, San Francisco (1997)
3. Alba, E., Luque, G.: Theoretical Models of Selection Pressure for dEAs: Topology Influence. In: Corne, D., et al. (eds.) 2005 IEEE Congress on Evolutionary Computation (CEC 2005), Edinburgh, UK, pp. 214–222 (2005)
4. Sprave, J.: A Unified Model of Non-Panmictic Population Structures in Evolutionary Algorithms. In: Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A. (eds.) Proceedings of the Congress of Evolutionary Computation, Mayflower Hotel, Washington D.C., USA, vol. 2, pp. 1384–1391. IEEE Press (July 1999)
5. Chakraborty, U.K., Deb, K., Chakraborty, M.: Analysis of Selection Algorithms: A Markov Chain Approach. Evolutionary Computation 4(2), 133–167 (1997)
6. Gorges-Schleuter, M.: An Analysis of Local Selection in Evolution Strategies. 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, Orlando, Florida, USA, vol. 1, pp. 847–854. Morgan Kaufmann (July 1999)
7. Rudolph, G.: Takeover Times in Spatially Structured Populations: Array and Ring. In: Lai, K.K., Katai, O., Gen, M., Lin, B. (eds.) 2nd Asia-Pacific Conference on Genetic Algorithms and Applications, pp. 144–151. Global-Link Publishing (2000)

8. Cantú-Paz, E.: Migration, Selection Pressure, and Superlinear Speedups. In: Efficient and Accurate Parallel Genetic Algorithms, pp. 97–120. Kluwer (2000)
9. Giacobini, M., Tomassini, M., Tettamanzi, A.G.B.: Modeling Selection Intensity for Linear Cellular Evolutionary Algorithms. In: Liardet, P., Collet, P., Fonlupt, C., Lutton, E., Schoenauer, M. (eds.) EA 2003. LNCS, vol. 2936, pp. 345–356. Springer, Heidelberg (2004)
10. Forbes, C., Evans, M., Hastings, N., Peacock, B.: Rayleigh Distribution. In: Statistical Distributions. John Wiley & Son (2010)
11. Osorio, K., Luque, G., Alba, E.: Distributed Evolutionary Algorithms with Adaptive Migration Period. In: 2011 11th International Conference on Intelligent Systems Design and Applications (ISDA), pp. 259–264 (November 2011)

Variable Transformations in Estimation of Distribution Algorithms

Davide Cucci, Luigi Malagò, and Matteo Matteucci

Department of Electronics and Information, Politecnico di Milano,
Via Ponzio, 34/5, 20133 Milano, Italy
{cucci,malago,matteucci}@elet.polimi.it

Abstract. In this paper we address model selection in Estimation of Distribution Algorithms (EDAs) based on variables trasformations. Instead of the classic approach based on the choice of a statistical model able to represent the interactions among the variables in the problem, we propose to learn a transformation of the variables before the estimation of the parameters of a fixed model in the transformed space. The choice of a proper transformation corresponds to the identification of a model for the selected sample able to implicitly capture higher-order correlations. We apply this paradigm to EDAs and present the novel Function Composition Algorithms (FCAs), based on composition of transformation functions, namely I-FCA and Chain-FCA, which make use of fixed low-dimensional models in the transformed space, yet being able to recover higher-order interactions.

Keywords: Function Composition Algorithm, Transformation of Variables, Minimization of Mutual Information, Chain Model.

1 Introduction

Estimation of Distribution Algorithms (EDAs) belong to the class of metaheuristics for optimization where the search is guided by a statistical model able to capture the interactions among the variables in the problem. When the model is not given a priori, model selection becomes crucial in order for the algorithm to be able to detect global optimal solutions. Indeed if the model chosen is not expressive enough, or if the wrong interactions are considered, model-based search strategies are prone to converge to local optima, cf. [16,10,6].

As a consequence, much of the literature in the EDAs community is focused on applying efficient algorithms for model selection, able to identify the correct interactions of the function from a sample of observations. Among the others we mention algorithms which reconstruct the topology of a Bayesian Network, as in BOA [12], clustering algorithms for the variables that appear to be correlated, eCGA [8], or model selection for Markov Random Fields (MRFs), as in DEUM [13]. When no prior information about the problem is available, EDAs need an efficient and scalable policy for model selection. In the general case learning an accurate model is exponential in the number of variables thus it is a

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 428–437, 2012.

common practice to reduce the search space for the models, for example by limiting the interactions considered to the second order, when learning a MRF, by constraining the number of incoming edges in a BN or employing variable clustering techniques as in [14]. These restrictions can limit the performance of the algorithms in presence of certain structures of interactions among the variables.

In this paper we propose an approach to the problem of model selection based on the idea of applying a transformation of the variables and then employing a fixed low dimensional statistical model in the new transformed space. Obviously we moved much of the computational complexity from model selection to the choice of a good transformation; on the other side it becomes easier to select models able to capture higher-order interactions among the variables. Instead of limiting the search up to a given order of interactions, due to the family of transformations we introduce, we are able to identify non hierarchical models that can be efficiently employed in an EDAs.

To the best knowledge of the authors, the approach of transforming the original variables first appeared in [17], where the UMDA [11] is run over a set of new variables obtained applying ICA on the selected sample. More recently, Toussaint proposed to employ compression algorithms to the sample of promising solutions [15], Cho and Zhang cluster similar individuals in a group and explain the high order interactions with latent variables [3], Grosset et al. introduce physically meaningful auxiliary variables related to the application domain [7].

The paper is organized as follows. In Section 2 we review the MBS approach to optimization. In Section 3 we present the idea of employing variable transformations to perform model selection. In Section 4 we describe the Function Composition Algorithms (FCAs) family. First we review and discuss more in detail I-FCA, originally presented in [4], next we introduce a novel algorithm called Chain-FCA, which makes use of a fixed chain model. In Section 5, we discuss and compare the preliminary performance of the above-mentioned algorithms. In Section 6, we conclude by presenting some future directions of research.

2 Model Based Search and Stochastic Relaxation

We are interested in the minimization of a real-valued function f defined over a vector of binary variables. For mathematical convenience and without loss of generality we consider values in $\{\pm 1\}$, rather than classic $0/1$ encoding. Let us introduce the notation that will be used in the following. Let $x = (x_1, \ldots, x_n) \in \Omega = \{\pm 1\}^n$ a vector of n binary variables, any $f : \Omega \mapsto \mathbb{R}$ can be represented uniquely as a square-free polynomial, i.e., the finite sum of monomials

$$f(x) = \sum_{\alpha \in F} c_\alpha x^\alpha, \quad c_\alpha \in \mathbb{R}^n, \tag{1}$$

where we employed the multi-index notation $\alpha = (\alpha_1, \ldots, \alpha_n) \in F \subset \{0,1\}^n$, and $x^\alpha = \prod_{i=1}^n x_i^{\alpha_i}$. For instance, let $n = 3$ and $f = x_1 x_2 + x_2 x_3$, then $F = \{(1,1,0),(0,1,1)\}$. The monomials $\{x^\alpha\}$ with $\alpha \in \{0,1\}^n$ defines a basis for any function, while those identified by F correspond to the interactions present in f.

The paradigm of Model-Based Search (MBS) in stochastic optimization, consists in finding the minimum of f by solving the optimization problem of the stochastic relaxation of f, i.e., the minimization of the expected valued of f with respect to a density in a statistical model $\mathcal{M}$.

From now on, we consider models $\mathcal{M}$ that belong to the exponential family of probability distributions of the form

$$p(x;\theta) = \exp\left\{\sum_{i=1}^{k} \theta_i T_i(x) - \psi(\theta)\right\}, \quad \theta \in \mathbb{R}^k, \tag{2}$$

where $\theta = (\theta_1, \ldots, \theta_k)$ is the vector of natural parameters, $T_i(x) : \Omega \to \mathbb{R}$ are the sufficient statistics, and $\psi(\theta)$ is a normalizing factor. Such choice is not restrictive, indeed many models used in MBS belong to this family, such as log-linear models, the Gibbs distribution, and more in general MRFs.

Since the sum of the sufficient statistics is a function defined over Ω, with no prior information about f, it is convenient to choose the basis $\{x^\alpha\}$ itself as the set of sufficient statistics. However, the basis has $2^n - 1$ monomials, so is computationally intractable. For this reason, we consider lower-dimensional models identified by a subset of the sufficient statistics identified by a small subset of indices $M \subset \{0,1\}^n$, usually polynomial in n. Each monomial in M identifies one of the possible correlations between groups of variables.

The choice of the model is central in MBS. From a theoretical point of view, the best choice would be to chose $\mathcal{M}$ such that $M \triangleq F$, so that the stochastic relaxation admits no local minima [16]. Models with smaller number of monomials may admit local minima, so that algorithms are more prone to convergence to local minima for f. On the other side, larger models imply more computational costs for parameter estimation. Dealing with the exponential family, one possible approach for model-selection is to test all possible second-order interactions, and then in case move to higher-order correlations, as in e.g. [13]. The computational complexity of these techniques grows with the maximum order of f in Equation (1), c.f. [6]. Dealing with BNs, hBOA [12] solves this issue by introducing trees between variables, which allow to efficiently learn hierarchies between variables and thus higher-order correlations. Instead of employing standard statistical techniques able to learn high-dimensional models directly, we propose to employ variable transformations to perform implicit model selection.

3 Variable Transformations

In this section we describe an implicit approach to model selection in MBS, and in particular in EDAs, where the problem of identifying a model is replaced by a search for a transformation of the variables of f. By employing a fixed model for the transformed variables, we are implicitly choosing a different model in the original space, which depends on the transformation. We are interested in those transformations such that a model in the transformed space corresponds to a model in the original space which is able to capture the interactions of f.

Let us introduce a new vector of variables $y = (y_1, \ldots, y_n)$ in Ω and a one-to-one map h such that $y = h(x)$. We can express f as the composition of a function $g(y) : \Omega \to \mathbb{R}$ with h, i.e., $f = g \circ h$, and $g = f \circ h^{-1}$. Since h defines a permutation of the points in Ω, follows that $\min g = \min f$.

We can express h component-wise, i.e., $h = (h_1(x), \ldots, h_n(x)$. Since $h_i(x) : \Omega \to \{\pm 1\}$, each h_i admits an expansion as in Equation (1), i.e.,

$$h_i = \sum_{\alpha \in H_i} c_{i,\alpha} x^{\alpha}, \quad 1 \leq i \leq n. \tag{3}$$

Let $q(y;\xi) \in \mathcal{N}$ be a density for Y from the exponential family in (2), by expanding all the products obtained by substituting y_i with $h_i(x)$, we obtain a polynomial in x whose monomials are identified by a set of indices M, i.e.,

$$\begin{aligned} &\exp\left\{\sum_{\alpha \in N} \xi_\alpha y^\alpha - \phi(\xi)\right\} = \exp\left\{\sum_{\alpha \in N} \xi_\alpha \prod_{i=1}^{n} (h_i)^{\alpha_i} - \phi(\xi)\right\} = \\ &\exp\left\{\sum_{\alpha \in N} \xi_\alpha \prod_{i=1}^{n} \Big(\sum_{\gamma \in H_i} c_{i,\gamma} x^\gamma\Big)^{\alpha_i} - \phi(\xi)\right\} = \exp\left\{\sum_{\beta \in M} l_\beta(\xi) x^\beta - \psi(\xi)\right\}. \end{aligned} \tag{4}$$

By setting $\theta_\beta = l_\beta(\xi) \in \mathbb{R}$, we expressed $q(y, \xi)$ as the probability distribution $p(x;\theta)$ for the original variables, where l_β is a function which maps the parameters of the two exponential families. Notice that since h is one-to-one the dimension of the θ and the ξ parameter space are the same.

In other words, suppose we apply a transformation from x to y and consider an exponential family $\mathcal{N} = \{q(y;\xi), \xi \in \mathbb{R}^k\}$ over Y identified by the sufficient statistics in N. By Equation (4), $q(y;\xi) = p(x;\theta)$, with $y = h(x)$ and $\theta = l(\xi)$, thus $\mathcal{N}$ maps into the exponential family $\mathcal{M}$ for X, identified by a different set of sufficient statistics M. Such mapping is one-to-one, so that $\mathbb{E}_p[f] = \mathbb{E}_q[g]$, and $\min_{q \in \mathcal{N}} \mathbb{E}_q[g] = \min_{p \in \mathcal{M}} \mathbb{E}_p[f]$, so that the minimization of stochastic relaxation of f with respect to $\mathcal{M}$ is equivalent to those of g with respect to $\mathcal{N}$. Consider the following example. The function $f = x_1x_2 + x_2x_3$, $x \in \{\pm 1\}^3$ admits two global minima $x = (-1, 1, -1)$ and $(1, -1, 1)$. Let us apply following one-to-one map $y = h(x)$ and its inverse h^{-1}

$$h : \begin{cases} y_1 = x_1x_2 \\ y_2 = x_2x_3 \\ y_3 = x_3 \end{cases} \qquad h^{-1} : \begin{cases} x_1 = y_1y_2y_3 \\ x_2 = y_2y_3 \\ x_3 = y_3 \end{cases}$$

Let $\mathcal{N}$ be the exponential family defined over Y with $\{y_1, y_2, y_3\}$ as sufficient statistics. By expanding $q(y;\xi) \in \mathcal{N}$ we have that

$$\begin{aligned} \mathcal{N} \ni q(y, \xi) &= \exp\{\xi_1 y_1 + \xi_2 y_2 + \xi_3 y_3 - \phi(\xi)\} = \\ &= \exp\{\theta_1 x_1 x_2 + \theta_2 x_2 x_3 + \theta_3 x_3 - \psi(\theta)\} = p(x, \theta) \in \mathcal{M}, \end{aligned}$$

where $\theta = \xi$ and $\psi(\theta) = \phi(\xi)$. The sufficient statistics of $\mathcal{M}$ include the interactions on f, i.e., $F \subset M$. Follows that the stochastic relaxation of f with respect

to $\mathcal{M}$ does not admint local minima, for every $p \in \mathcal{M}$ the gradient of $\mathbb{E}_p[f]$ points into the direction of the global optimum of the relaxed problem, c.f., [10]. We can explicitly compute $g(y) = f \circ h^{-1}$. Since in the binary case $x_i^2 = 1$, $g = h_1^{-1}h_2^{-1} + h_2^{-1}h_3^{-1} = y_1 + y_2$, which is linear in y. The minimization of f can thus be performed considering the stochastic relaxation of g with respect to $\mathcal{N}$. This problem is simpler than the original one since we are minimizing the expected value of a linear function with respect to the independence model and the stochastic relaxation does not admit local minima, c.f., [9]. This example shows how the use of a properly chosen variable transformation can greatly affect the complexity of an optimization problem from the point of view of model-based search strategies.

4 Function Composition Algorithms

In this section we present the idea of learning a transformation of the variables before estimating the parameters of a fixed low-dimensional model in the transformed space. Such approach to model selection is applied to the EDAs paradigm, leading to a novel family of algorithm called Function Composition Algorithms (FCAs). Preliminary work appeared in [4].

Recall the basic iteration of an EDA,

$$\mathcal{P}^t \xrightarrow{\text{selection}} \mathcal{P}_s^t \xrightarrow{\text{estimation}} p(x;\theta^t) \in \mathcal{M} \xrightarrow{\text{sampling}} \mathcal{P}^{t+1}.$$

At each iteration t of an EDA, a subset $\mathcal{P}_s^t$ of the population $\mathcal{P}^t$ is selected according to a given selection policy. Then, a statistical model $\mathcal{M}$ is learned from the subsample, and the parameters of a distribution $p(x;\theta^t)$ are estimated. Finally, a new population $\mathcal{P}^{t+1}$ is generated by sampling. Some algorithms, such as PBIL [1] or UMDA [11], make the assumption of independent variables, others use low-dimensional models, such as the chain model, see MIMIC [5], while more powerful EDAs, e.g., hBOA [12] or DEUM [13,2] perform model selection in a larger class of models, able to capture higher-order correlations among variables.

In FCA, we implicitly learn a model by first choosing a variable transformation, and then using a fixed model for the new set of transformed variables. We introduce the following variation of the iteration of an EDA. Estimation and sampling are preceded and followed by two transformations. First a one-to-one map $y = h(x)$ is applied to each individual in the selected population obtaining $\tilde{\mathcal{P}}_s^t$, then after sampling from the estimated distribution, the population $\tilde{\mathcal{P}}^{t+1}$ is transformed back to the original space by means of h^{-1}, i.e.,

$$\mathcal{P}_s^t \xrightarrow{h} \tilde{\mathcal{P}}_s^t \xrightarrow{\text{estimation}} q(y;\xi^t) \in \mathcal{N} \xrightarrow{\text{sampling}} \tilde{\mathcal{P}}^{t+1} \xrightarrow{h^{-1}} \mathcal{P}^{t+1}.$$

From Equation (4), estimating a probability distribution $q(y;\xi) \in \mathcal{N}$ for the transformed sample $\tilde{\mathcal{P}}_s^t$ is equivalent to estimate a distribution $p(x;\theta) \in \mathcal{N}$ for $\mathcal{P}_s$. In general $\mathcal{N}$ and $\mathcal{M}$ are different, since the latter depends on the map h employed, so that the choice of h corresponds to choice of a model $\mathcal{M}$.

4.1 Independence-FCA

In the following we briefly review I-FCA, first introduced in [4]. I-FCA employs the independence model for the transformed variables y, and if the map h is properly chosen, the resulting low-dimensional model $\mathcal{M}$ can achieve a better approximation of the sample $\mathcal{P}_s$ with respect to the independence model for x.

The non-linear maps used in I-FCA are defined as follows. Consider the maps h indexed by j, $k \in \{1, \ldots, n\}$, with $j \neq k$, such that each h_i is defined as

$$h_i^{(j,k)} : \begin{cases} y_i = x_i x_k & \text{if } i = j \\ y_i = x_i & \text{otherwise.} \end{cases}$$

We have $n(n-1)$ different $h^{(j,k)}$ transformations. It is easy to see that they are one-to-one and that $h^{-1} = h$, since $x_i^2 = 1$. Next we extend the class of transformations we consider by allowing elements h to be the composition of a finite number $\overline{m}$ of maps of the form $h^{(j,k)}$:

$$h = h^{(j_1,k_1)} \circ \ldots \circ h^{(j_m,k_m)} \circ \ldots \circ h^{(j_{\overline{m}},k_{\overline{m}})}. \tag{5}$$

Since the inverse of each transformation in the sequence of compositions is the element itself, it is easy to see that h^{-1} is the composition of all the $h^{(j_m,k_m)}$ in the reversed order. Moreover, if the sufficient statistics of $\mathcal{N}$ are monomials in y, the sufficient statistics of the resulting model $\mathcal{M}$ for X are monomials in x.

In I-FCA we propose a strategy for the choice of map h based on the maximization of the likelihood of the transformed selected sample $\tilde{\mathcal{P}}_s$ with respect to the estimated distribution $q(y, \hat{\xi}) \in \mathcal{N}$, where $\mathcal{N}$ is the independence model for Y. This is equivalent to minimize the Kullback-Leibler divergence between the empirical distribution representing the selected population and its projection on the independence model (i.e. $KLD[\tilde{\mathcal{P}}_s \| q(y, \hat{\xi})] = -H[\tilde{\mathcal{P}}_s] - \mathcal{L}[\tilde{\mathcal{P}}_s \| q(y, \hat{\xi})]$), which gives a measure of the loss of information which occurs when $\tilde{\mathcal{P}}_s$ is approximated with $q(y, \xi)$. Note that since h is one-to-one $H[\tilde{\mathcal{P}}_s]$ does not depend oh h.

In order to make the search for h feasible, we choose a greedy approach. We initialize h to be the identity map $y = x$, then we iteratively examine all the $n(n-1)$ maps $h^{(j,k)}$ and compose the h map obtained at the previous step with the map $h^{(j,k)}$ which better improves the likelihood of $(h \circ h^{(j,k)})(\mathcal{P}_s)$ with respect to the independence model. The iteration stops when no improvement in the likelihood is achievable composing further maps of the form $h^{(j,k)}$ or when the maximum number $\overline{m}$ of transformations in h has been reached.

The representation of h as a composition of maps of the form $h^{(j,k)}$ is highly redundant, i.e., there exists more than one sequence of indices (j_m, k_m) which transforms the independence model $\mathcal{N}$ to the same exponential family $\mathcal{M}$. As a consequence, in order to reduce the complexity of the search strategy for h, we discard maps that produce models already examined in earlier stages of the search process. Each time a new map $h^{(j,k)}$ is considered, the sufficient statistics y_i are transformed and the corresponding monomials $x^\beta = y_i$ with $\beta \in M$ are computed. Next, maps for which the monomial x^β does not contain x_i, i.e., $\beta_i = 1$, or, for all i, the degree of x^β decrease when $h^{(j,k)}$ is applied, are discarded.

The worst case time complexity of the greedy search strategy for h is $\mathcal{O}(n^2\overline{m}N)$, where n is the number of variables in f and N is the population size. Note that it is possible to take advantage of the following log-likelihood decomposition:

$$\mathcal{L}[\tilde{\mathcal{P}}_s \| q(y, \hat{\xi})] = \frac{1}{N} \sum_{y \in \tilde{\mathcal{P}}_s} \log \Big(\prod_{i=1}^{n} q_i(y_i; \hat{\xi}_i) \Big) = \frac{1}{N} \sum_{i=1}^{n} \overbrace{\sum_{y \in \tilde{\mathcal{P}}_s} \log q_i(y_i; \hat{\xi}_i)}^{\mathcal{L}_i},$$

since q belongs to the independence model. When a new map $h^{(j_m,k_m)}$ is considered, since $y_i = x_i$, for $i \neq j$, we do not need to compute the terms $\mathcal{L}_i$ and the values already evaluated at the previous step $m-1$ can be used.

4.2 Chain-FCA

The variable transformation paradigm is general, and different models can be chosen for transformed variables. In the following we introduce Chain-FCA, a novel algorithm in FCAs family, where we fix a model with interactions, rather than the independence model, as in I-FCA. Consider the family of probability distributions for which the joint probability function factorizes as

$$p(y, \xi) = p(y_1) \prod_{j=2}^{n} p(y_j | y_{j-1}). \tag{6}$$

This is a chain model whose structure is fixed and each variable except the first depends only on the previous one. The parameter vector ξ has $2(n-1)+1$ components, one for the marginal probability of y_1, and two for each of the conditional probabilities, and can be easily estimated by means of max-likelihood estimation. The log-likelihood of a sample with respect to this model is given by

$$\mathcal{L}[\tilde{\mathcal{P}}_s \| q(y, \hat{\xi})] = \sum_{j=1}^{n-1} I(Y_j | Y_{j+1}) - \sum_{j=1}^{n} H(Y_j), \tag{7}$$

where $H(Y_j)$ is the marginal entropy and $I(Y_j|Y_k)$ is the mutual information.

Chain-FCA employs the chain model defined in (6) and a greedy search strategy to choose the sequence of maps $h^{(j,k)}$ which maximizes the likelihood of the transformed set of selected individuals $\tilde{\mathcal{P}}_s$. The order of the variables in the chain is fundamental. For this reason the class of the maps h is enriched by allowing the swap of couples of variables. This operation is equivalent to the composition of three maps $h^{(j,k)}$. Consider for example two variables x_1, x_2 and the map $y = h_1^{(1,2)} \circ h_2^{(2,1)} \circ h_3^{(1,2)}$. It turns out that $y_1 = x_2$ and $y_2 = x_1$, in fact

$$\{x_1, x_2\} \overset{h^{(1,2)}}{\Rightarrow} \{\overbrace{x_1 x_2}^{y_1}, \overbrace{x_2}^{y_2}\} \overset{h^{(2,1)}}{\Rightarrow} \Rightarrow \{\overbrace{x_1 x_2}^{y_1}, \overbrace{x_1}^{y_2}\} \overset{h^{(1,2)}}{\Rightarrow} \{\overbrace{x_2}^{y_1}, \overbrace{x_1}^{y_2}\}$$

The map h implies the swap of the variables x_1 and x_2. Notice that this was useless in I-FCA since the order of the variables is not relevant in the independence model, and such maps are discarded a priori. On the other hand, in Chain-FCA this allows to implicitly adapt the fixed structure of the interactions among

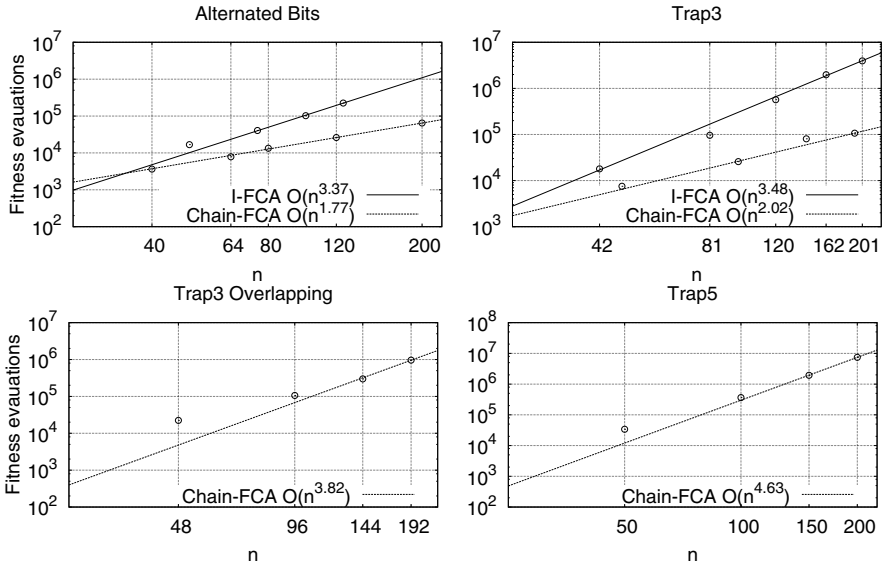


Fig. 1. Scalability of I-FCA and Chain-FCA

the variables in the chain model to the ones appearing in the set of candidate solutions. Notice that if the search for h is restricted to consider *only* variables swaps the result is a model selection algorithm very similar to MIMIC [5].

By means of a fixed structure chain model and a greedy search strategy for the choice of the transformation, Chain-FCA is able to implicitly learn a richer model compared to I-FCA, characterized by $2(n-1)+1$ parameters. The worst case time complexity of Chain-FCA is $\mathcal{O}(n^2\overline{m}N)$, since only $\frac{n(n-1)}{2}$ variables swaps have to be examined, along with the $n(n-1)$ maps of the form $h^{(j,k)}$.

5 Experimental Results

In this section we present the results of a preliminary scalability evaluation for I-FCA and for the novel Chain-FCA algorithm, over a set of well known benchmarks functions: Alternated Bits, Trap3, Trap3 overlapping, and Trap5. In Alternated Bits the variables interact in a chain structure and higher fitness is given to the instances for which the variables take opposite values with respect to their neighbors in the chain. Trap3 and Trap5 are deceptive functions and are composed of independent blocks of 3 and 5 variables, respectively. Each block has a global optimum and a deceptive local optimum. Trap3 overlapping is similar with respect to Trap3 but the blocks fully overlap.

In our algorithms we perform truncation selection and we choose the best S individuals. After a preliminary parameter tuning we fix $S = 10n$ for I-FCA and $S = 5n$ for Chain-FCA for all the problems considered, independently from the population size. Experiments show that in the case of I-FCA no improvement is

achievable when $\overline{m} > n$. This result is also supported by an empirical analysis on the set of models $\mathcal{M}$ which can be obtained mapping the independence model into the original space through h. In the case of Chain-FCA the class of the models that can be obtained by means of h is wider, so we set $\overline{m} = 4n$.

For each problem and for each dimension we determine the population size which ensure at most one failure out of 24 run. Then we estimated the average number of fitness evaluations performed when the global optimum for f first appears in the population. The results are presented in Figure 1, along with an asymptotic estimation obtained by means of a least square fit of the curve an^b. Notice that, with the only exception of Alternated Bits, these functions are *not* solved by other EDAs based on a fixed or low-dimensional model such as PBIL, UMDA, or MIMIC, and in general one has to move to more EDAs which perform model selection on a large class of complex models, such as BOA or DEUM. I-FCA solves Alternated Bits and Trap3 while Chain-FCA robustly solves all the benchmark functions considered. This proves the viability of the variable transformations approach. Both algorithms are part of the Evoptool toolkit. Source code and the detailed experimental settings are public availabe[1].

6 Conclusions and Future Works

Variables transformations can be employed as an alternative approach to model selection in MBS. In this paper we presented theoretical foundations of such approach, and proposed a novel algorithm in the FCA family, called Chain-FCA, which chooses a variable transformation maximizing maximum likelihood with respect to a fixed chain model. Both I-FCA and Chain-FCA choose the variable transformation to apply by iteratively composing basic modular maps. Besides the usual EDAs parameters, selection policy and population size, these algorithms have one more parameter which is the length of the composition sequence in the variable transformation, even though we argue that this parameter is problem independent and could be fixed a priori.

A preliminary experimental evaluation of the performances of Chain-FCA compared to I-FCA showed that these algorithms are able to solve functions characterized by higher-order interaction yet only employing fixed low dimensional models. This shows the viability of the variable transformation approach.

Some directions of future works include testing on different and more complex benchmark functions, experimenting more expressive classes of variables transformations and different models for the transformed variables, such as Chow-Liu trees. Performance enhancements could also come by the replacement of the greedy search strategy for h with more advanced policies.

References

1. Baluja, S., Caruana, R.: Removing the genetics from the standard genetic algorithm. In: Machine learning: proceedings of the Twelfth International Conference on Machine Learning, pp. 38–46. Morgan Kaufmann (1995)

[1] http://airlab.elet.polimi.it/index.php/Evoptool

2. Brownlee, A.E.I., McCall, J.A.W., Shakya, S.K., Zhang, Q.: Structure Learning and Optimisation in a Markov Network Based Estimation of Distribution Algorithm. In: Chen, Y.-p. (ed.) Exploitation of Linkage Learning. ALO, vol. 3, pp. 45–69. Springer, Heidelberg (2010)
3. Cho, D., Zhang, B.: Evolutionary optimization by distribution estimation with mixtures of factor analyzers. In: Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002, vol. 2, pp. 1396–1401 (2002)
4. Corsano, E., Cucci, D., Malagò, L., Matteucci, M.: Implicit model selection based on variable transformations in estimation of distribution. In: Learning and Intelligent OptimizatioN Conference LION 6. LNCS, vol. 7219. Springer (to apppear, 2012)
5. De Bonet, J., Isbell, C., Viola, P.: Mimic: Finding optima by estimating probability densities. In: Advances in Neural Information Processing Systems, p. 424. The MIT Press (1996)
6. Echegoyen, C., Zhang, Q., Mendiburu, A., Santana, R., Lozano, J.: On the limits of effectiveness in estimation of distribution algorithms. In: 2011 IEEE Congress on Evolutionary Computation (CEC), pp. 1573–1580 (June 2011)
7. Grosset, L., LeRiche, R., Haftka, R.: A double-distribution statistical algorithm for composite laminate optimization. Structural and Multidisciplinary Optimization 31, 49–59 (2006)
8. Harik, G.: Linkage learning via probabilistic modeling in the eCGA, 1999. Harik, G. R (1999); Linkage Learning via Probabilistic Modeling in the ECGA (IlliGAL Report No. 99010). University of Illinois at Urbana-Champaign
9. Hohfeld, M., Rudolph, G.: Towards a theory of population-based incremental learning. In: Proceedings of the 4th IEEE Conference on Evolutionary Computation, pp. 1–5. IEEE Press (1997)
10. Malagò, L., Matteucci, M., Pistone, G.: Towards the geometry of estimation of distribution algorithms based on the exponential family. In: Proceedings of the 11th Workshop on Foundations of Genetic Algorithms, FOGA 2011, pp. 230–242. ACM, New York (2011)
11. Mühlenbein, H., Mahnig, T.: Mathematical analysis of evolutionary algorithms. In: Essays and Surveys in Metaheuristics, Operations Research/Computer Science Interface Series, pp. 525–556. Kluwer Academic Publishers (2002)
12. Pelikan, M., Goldberg, D.: Hierarchical Bayesian Optimization Algorithm. In: Pelikan, M., Sastry, K., Cant Paz, E. (eds.) Scalable Optimization via Probabilistic Modeling. SCI, vol. 33, pp. 63–90. Springer, Heidelberg (2006)
13. Shakya, S., Brownlee, A., McCall, J., Fournier, F., Owusu, G.: A fully multivariate DEUM algorithm. In: IEEE Congress on Evolutionary Computation (2009)
14. Thierens, D.: The Linkage Tree Genetic Algorithm. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 264–273. Springer, Heidelberg (2010)
15. Toussaint, M.: Compact Genetic Codes as a Search Strategy of Evolutionary Processes. In: Wright, A.H., Vose, M.D., De Jong, K.A., Schmitt, L.M. (eds.) FOGA 2005. LNCS, vol. 3469, pp. 75–94. Springer, Heidelberg (2005)
16. Zhang, Q.: On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm. IEEE Transactions on Evolutionary Computation 8(1), 80–93 (2004)
17. Zhang, Q., Allinson, N., Yin, H.: Population optimization algorithm based on ica. In: 2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, pp. 33–36 (2000)

Controlling Overfitting in Symbolic Regression Based on a Bias/Variance Error Decomposition

Alexandros Agapitos, Anthony Brabazon, and Michael O'Neill

Financial Mathematics and Computation Research Cluster,
Natural Computing Research and Applications Group,
University College Dublin, Ireland
{alexandros.agapitos,anthony.brabazon,m.oneill}@ucd.ie

Abstract. We consider the fundamental property of generalisation of data-driven models evolved by means of Genetic Programming (GP). The statistical treatment of decomposing the regression error into bias and variance terms provides insight into the generalisation capability of this modelling method. The error decomposition is used as a source of inspiration to design a fitness function that relaxes the sensitivity of an evolved model to a particular training dataset. Results on eight symbolic regression problems show that new method is capable on inducing better-generalising models than standard GP for most of the problems.

1 Introduction

Reliable learning in the field of Machine Learning (ML) revolves around the property of *generalisation*, which is the ability of a learned model to correctly explain data that are drawn from the same distribution as the training data, but have not been presented during the training process. This is the very important property that ML algorithms aim to optimise. The generalisation performance of a model relates to its prediction capability on an independent test dataset. Assessment of this performance guides the choice of a model, and provides a measure of the quality of the ultimately chosen model. The loss of generalisation is referred to as the problem of *overfitting* [4].

In the case of learning regression models, the task is to discover a target function $f(X)$ that map a vector of real-valued inputs X to a real-valued target variable Y. A prediction model $\hat{f}(X)$ is trained on a training dataset $D = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$ of size n, where model accuracy on an individual training case is specified using a loss function for measuring the errors between Y and $\hat{f}(X)$ denoted by $L(Y, \hat{f}(X))$. Typical choice is the square error $(Y - \hat{f}(X))^2$, and the error over the entire set D is taken as the average of individual losses. The use of least squares, can lead to severe overfitting if complex regression models are trained over limited-sized datasets [4]. In an example of polynomial curve fitting [4](pages 4-11), model complexity is measured by the order of the polynomial. It is shown that a polynomial of a low order and few coefficients gives poor predictions on test data since the polynomial function has

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 438–447, 2012.

too little flexibility to be learning anything at all during training. On the other hand, a polynomial with too many coefficients has poor generalisation since it fits to closely to the noise on the training data. The issue of model complexity is central to overfitting. There is a trade-off between achieving a good fit to the training data, and obtaining a model which is not very complex, and thus does not overfit. Significant insight into this trade-off can be obtained by introducing the statistical concept of *bias/variance* error decomposition, under which the generalisation error of a model is decomposed into the sum of *bias* squared plus the *variance*. The *bias* measures the accuracy of the estimated $\hat{f}(X)$, while the *variance* measures the extend to which $\hat{f}(X)$ is sensitive to the particular dataset D used during training.

Genetic Programming (GP) [9] tackles regression problems by means of searching a model space for the most appropriate functional model-form along with the optimal coefficients given a training set of input-output pairs. A plethora methods for learning good-generalising models have been investigated in previous research; some are reported in the works of [1,2,3,10,11,12].

In this study, we draw inspiration from the bias/variance error decomposition and devise a method to improve on the sensitivity of an evolved model to a particular training dataset. The method is based on the bootstrap resampling method to randomly draw datasets with replacement from the training data, and calculate the variance of the error on all bootstrap samples. The variance is then used along with the error on the original training dataset in a single-objective fitness function that takes the form of their weighted sum that is to be minimised. Given the two conflicting objectives of *bias* and *variance*, a Pareto-based multi-objective fitness function would be a sensible line of attacking this problem. At this preliminary study, we chose to aggregate the two objectives in a scalar fitness function, and explicitly investigate the effect of different kinds of trade-off for biasing the search towards good-generalising regression models.

The rest of the paper is organised as follows. Section 2 introduces the statistical concept of bias/variance decomposition of regression error. Section 3 presents a new method for relaxing the sensitivity of evolved models to a particular dataset used during training. Section 4 presents the symbolic regression problems that will be used in this study, and details the experiment method. Section 5 analyses the results, and finally Section 6 draws our conclusions.

2 Bias and Variance for Regression

This section presents the basic background on the statistical concept of bias/variance regression error decomposition, and motivates the development of the new method for tackling overfitting. The material is based on the textbook of Bishop [4] (pages 147-152).

Consider we wish to model the underlying generator of a dataset, so that the best possible predictions for the target vector t can be made when a trained model is presented with a new value of the input vector x. For that, we are estimating a model $y(x)$ for a target function $\langle t|x\rangle$ using a training dataset D,

where $\langle t|x\rangle$ denotes the conditional average of the target data, so that $\langle t|x\rangle = \int tp(t|x)dt$. The most general descriptor of the generator of D is in terms of the probability density $p(x,t) = p(t|x)p(x)$ in the joint input-target space. Our training algorithm minimises the sum-of-squares error function, thus each individual error is calculated as $\{y(x) - \langle t|x\rangle\}^2$, and depends on the training dataset D and on the particular datapoint x. Integrating this quantity over x will give the usual sum-of-squares error measure.

Suppose we have a large ensemble of datasets of the same size, each drawn independently from the distribution $p(t,x)$ of D. We can eliminate the dependency of a model on a particular training dataset by measuring the performance of a model using the average of the ensemble of datasets, which we write as:

$$\mathbb{E}_D[\{y(x) - \langle t|x\rangle\}] \tag{1}$$

where $\mathbb{E}_D[\cdot]$ denotes the expectation, or ensemble average, which represents the error of model $y(x)$ when trained over equal-sized samples of D. If the trained model was a perfect predictor of the target function $\langle t|x\rangle$, then this error would be zero. Nevertheless a non-zero error can occur for two distinct reasons. It may be that the estimated model $y(x)$ is different from the target function $\langle t|x\rangle$, which is called the *bias*. Alternatively, it may be that the method is sensitive on the particular sample training dataset, and as a result, at a given x its prediction is either larger or smaller than the target t depending on the dataset used for training. This is called the *variance*. We can decompose Equation 1 into bias and variance using the notion of an *average model* $\mathbb{E}_D[y(x)]$, which is the average of all predictions at point x of various models trained on different samples of D:

$$\begin{aligned}\{y(x) - \langle t|x\rangle\}^2 &= \{y(x) - \mathbb{E}_D[y(x)] + \mathbb{E}_D[y(x)] - \langle t|x\rangle\}^2 \\ &= \{y(x) - \mathbb{E}_D[y(x)]\}^2 \\ &\quad +2\{y(x) - \mathbb{E}_D[y(x)]\}\{\mathbb{E}_D[y(x)] - \langle t|x\rangle\} \\ &\quad +\{\mathbb{E}_D[y(x)] - \langle t|x\rangle\}^2\end{aligned} \tag{2}$$

By taking the expectation of both sides over the ensemble of datasets, we can express the expected squared difference as:

$$\begin{aligned}&\mathbb{E}_D[\{y(x) - \langle t|x\rangle\}^2] = \\ &\underbrace{\{\mathbb{E}_D[\{y(x)] - \langle t|x\rangle\}^2}_{(bias)^2} + \underbrace{\mathbb{E}_D[\{y(x) - \mathbb{E}_D[y(x)]\}^2]}_{variance}\end{aligned} \tag{3}$$

The first term, the bias, measures the extent to which the average model $\mathbb{E}_D[y(x)]$ differs from the target function $\langle t|x\rangle$. The second term, the variance, measures the extent to which a model trained on a specific dataset varies around the average model, and hence measures the sensitivity of a particular model to the particular choice of dataset. There is a trade-off between bias and variance, with very flexible models having low bias and high variance, whereas relatively rigid models having

high bias and low variance. The next section introduces a simple measure that quantifies the sensitivity of a model to a particular training dataset, and presents a new fitness function to relax this sensitivity in pursue of better generalisation.

3 Minimising the Error Variance on Bootstrap Datasets

Suppose we have a model $y(x)$ trained on a dataset $D = \{(x_1, t_1), \ldots, (x_N, t_N)\}$ using an error function that takes the form of mean Canberra distance (C).

$$C(D) = \frac{1}{N} \sum_{i=1}^{N} \frac{abs(y(x_i) - t_i)}{abs(y(x_i)) + abs(t_i)} \tag{4}$$

where *abs* returns the absolute value of its argument, $y(x_i)$, t_i are the predicted and target values respectively for input x_i, and N is the size of D. Canberra distance is preferred to Mean Squared Error (MSE) because it implicitly normalises the output within the [0.0, 1.0] interval, which is necessary for applying weights in the same interval during the aggregation of the objectives into a scalar fitness function.

We employ the bootstrap resampling method [6] to randomly draw B datasets with replacement from D, each sample the same size as D. For each of the bootstrap datasets D^{*b} we calculate the error value $C(D^{*b})$, thus the mean error over all dataset is given by $\overline{C^*} = \sum_{b=1}^{B} C(D^{*b})/B$.

The variance of the error from the bootstrap sampling is then simply:

$$Var(D^*) = \frac{1}{B-1} \sum_{b=1}^{B} (C(D^{*b}) - \overline{C^*})^2 \tag{5}$$

The error variance can be seen as a measure of the sensitivity of a model to the training dataset D, with overfitted models achieving a large error variance on the bootstrap datasets, whereas more general models obtaining a lower error variance. In order to relax the dependence on a particular dataset, a new fitness function to be minimised is defined as:

$$fitness = w_b C(D) + w_v Var(D^*) \tag{6}$$

which is the weighted sum of the mean error on the original dataset plus the variance of error on the bootstrap datasets, and w_b, w_v are the coefficients for error and variance respectively.

It is important to note that previous work investigated a bias/variance decomposition in GP [8] from the point of view of ensemble learning methods like *Bagging*. The output of an *average model* was calculated by averaging the outputs of an ensemble of models so that the expected generalisation error of the ensemble reduced to the bias error alone. In addition, the work of [5] successfully employed a Pareto-based bi-objective fitness function that was based on the MSE and the variance of the independent squared errors over a single training dataset. Our fitness function differs substantially from the one used in [5] in that we calculate the variance over a collection of bootstrap datasets.

4 Experiment Design

We designed a set of experiments to assess the effectiveness of the new method (BVGP) on the generalisation ability of evolved models. The method is contrasted against standard GP (SGP). We used three datasets: *training*, *validation*, and *testing*. The training dataset is used to fit the models. At the end of every generation the best model on training data is stored in an *elitist-list*. At the end of evolution, the validation set is used to estimate the prediction error of every element of the elitist-list in order to perform model selection. The test set is used for assessing the generalisation error of the final chosen model. The size of training and validation sets is the same for a problem, and the three datasets share no common elements. In the case of BVGP, the bootstrap resampling is performed on the training dataset.

Table 2 presents eight symbolic regression problems that are tackled in this work. Problems F_2, F_4, F_5, F_7 were chosen from [13] due to their pronounced difficulty as test problems for GP. Problems F_8, F_9, F_{10}, F_{11} were chosen from [7]. For these four problems we deliberately used small training datasets (20 points) in order to render the GP systems more prone to overfitting.

Table 1 summarises the setup of the GP systems. For the fitness function of BVGP in Equation 6, both $C(D)$ and $Var(D^*)$ are normalised into the same [0.0, 1.0] interval. We considered an exhaustive set of combinations with a step of 0.1 for the w_b and w_v, in order to test the effect of different trade-offs. On the other hand, SGP used Equation 4 as the fitness function. Note that the number of program evaluations are exactly the same in both fitness function calculations. This is because Equation 6 needs to calculate $C(D)$ on the original training dataset D, and afterwards the calculation of every bootstrap $C(D^{*b})$ can be based on the individual losses that were cached during the program evaluation with the training cases of D.

We performed 50 independent evolutionary runs for each GP system on each problem. Statistical significance of the differences in performance is evaluated using the Mann-Whitney U-test, considering a confidence of 95% and a pairwise Bonferroni correction for the value of α.

Table 1. GP systems setup

GP systems under comparison	BVGP, SGP
EA used in GP systems	elitist, generational, expression-tree representation
Function set	$+$, $-$, $*$, / (protected)
Terminal set	Regressor variables, 5 random constants in [0.0, 1.0]
No. of generations	51
Population size	500
Tournament size	4
Tree creation	ramped half-and-half (depths of 2 to 6)
Max. tree depth	20
Subtree crossover	30% (90% inner nodes, 10% leaf-nodes)
Subtree mutation	40%
Point mutation	30%
Fitness function	BVGP: Equation 6 (no. of bootstrap datasets: 500) SGP: Equation 4

Table 2. Symbolic regression problems with the respective data sampling ranges for training, validation and test datasets. Notation x=rand(a,b) means that the x variable is sampled uniform randomly from the interval [a, b]. Notation $x_1 = (a_1 : c_1 : b_1)$, $x_2 = (a_2 : c_2 : b_2)$ determines a uniform mesh with step length (c_1, c_2) on an interval $[a_1, b_1] \times [a_2, b_2]$. Both training and validation sets are of the same size for a problem.

	Problem	Training/Validation	Test
F_2	$f(x) = e^{-x}x^3cos(x)sin(x)(cos(x)sin^2x - 1)$	100 points x=rand(0.05, 10)	221 points x=(-0.5 : 0.05 : 10.5)
F_4	$f(x_1, x_2, x_3) = 30\frac{(x_1-1)(x_3-1)}{x_2{}^2(x_1-10)}$	300 points x_1, x_3=rand(0.05, 2) x_2=rand(1, 2)	2,701 points x_1, x_3=(-0.05 : 0.15 : 2.1) x_2 = (0.95 : 0.1 : 2.05)
F_5	$f(x_1, x_2) = 6sin(x_1)cos(x_2)$	50 points x_1, x_2=rand(0.1, 5.9)	961 points x_1, x_3=(0.05 : 0.02 : 6.05)
F_7	$f(x_1, x_2) = \frac{(x_1-3)^4+(x_2-3)^3-(x_2-3)}{(x_2-2)^4+10}$	50 points x_1, x_2=rand(0.05, 6.05)	1,157 points x_1, x_2=(-0.25 : 0.2 : 6.35)
F_8	$f(x_1, x_2) =$ $x_1x_2 + sin((x_1 - 1)(x_2 - 1))$	20 points x_1, x_2=rand(-3, 3)	361,201 points x_1, x_2=(-3 : 0.01 : 3)
F_9	$f(x_1, x_2) = {x_1}^4 - {x_1}^3 + {x_2}^2/2 - x_2$	20 points x_1, x_2=rand(-3, 3)	361,201 points x_1, x_2=(-3 : 0.01 : 3)
F_{10}	$f(x_1, x_2) = \frac{8}{2+{x_1}^2+{x_2}^2}$	20 points x_1, x_2=rand(-3, 3)	361,201 points x_1, x_2=(-3 : 0.01 : 3)
F_{11}	$f(x_1, x_2) = {x_1}^3/5 + {x_2}^3/2 - x_2 - x_1$	20 points x_1, x_2=rand(-3, 3)	361,201 points x_1, x_2=(-3 : 0.01 : 3)

5 Results

Table 4 summarises the performance statistics accrued from 50 independent runs of each experiment setup. The median value is preferred over the mean as it is more robust to outliers. The table reports the training Root Mean Squared Error (RMSE) obtained at the end of an evolutionary run, the test RMSE of models selected based on the validation set, the best-generalising model size in terms of number of tree-nodes, and the generation number when model selection took place. For the case of test RMSE the minimum value indicates the best-generalising model out of 50 runs. Table 3 summarises the p-values obtained by comparing the differences in the median test RMSE, median model size, median generation of model selection of BVGP against SGP using the Mann-Whitney U-test.

Observing the training error obtained by the different GP systems, results suggest that for all problems considered, both BVGP and SGP obtained a similar fit during training. Interestingly, the different trade-offs created by the different coefficient combinations did not seem to affect the training accuracy. When comparing the generalisation performance of BVGP against SGP we observe that in six out of eight problems BVGP outperformed SGP. The differences in median test RMSE are statistically significant (Table 3). For the remaining two problems the use of BVGP was deemed equivalent with that of SGP. It

Table 3. The p-values obtained by comparing the differences in the median test RMSE, median model size, median generation of model selection of BVGP against SGP using the Mann-Whitney U-test. Bold face indicates confidence of at least 95%.

		SGP		
		Test RMSE	**Model size**	**Model selection generation**
BVGP	F_2	0.45	0.15	0.36
	F_4	0.49	0.24	0.18
	F_5	**0**	0.22	0.23
	F_7	**0.04**	0.20	0.47
	F_8	**0.0004**	**0**	**0**
	F_9	**0.02**	0.12	0.14
	F_{10}	**0.0034**	0.14	0.31
	F_{11}	**0.001**	0.13	0.08

is important to note that for the problems F_8, F_9, F_{10}, and F_{11}, where small training sets were employed, all systems overfitted the training data. This is evidenced by the large degradation in test error as opposed to the cases of F_2, F_4, F_5, and F_7. However, the use of BVGP system appeared more resilient to overfitting. When inspecting the trade-off between error and variance that is required to enhance model generalisation, we observe no apparent trend to the combination of w_b and w_v coefficients that yields the best generalisation. The trade-off necessary to counteract overfitting appears to be problem dependent. Finally, the minimum test RMSE that is accrued from 50 independent runs of each system configuration suggests that for the majority of problems, BVGP produced the best-generalising model as opposed to SGP.

Early research on the relationship between model size and generalisation has advocated an intrinsic interaction between the two. The sixth column of Table 4 shows the median size of best-generalising models. For all the problems but F_8, we found no statistically significant differences in the sizes of the expression-trees representing the evolved models (Table 3). This is in accordance to the latest findings on the relationship between the expression-tree size and overfitting [5], ascertaining that in light of bloat in variable-length GP representations, model complexity is not directly associated with the number of tree-nodes.

Finally, the generation number when model selection is performed shows the point within an evolutionary run in which overfitting is becoming apparent. Table 4 shows that for the problems of F_2, F_4, F_5, F_7 that use intermediate to large training sets ranging from 50 to 300 data points, there appears to be a relationship between the size of the training set and the speed of generalisation loss. Contrasting between the case of F_4 that used a training set of 300 points against the cases of F_2, F_5, and F_7 that employed smaller training sets (sizes of 50 and 100), we noted that the smaller the training dataset, the quicker the model selection needs to be performed in order to avoid overfitting. On the other hand, for the cases of F_8, F_9, F_{10}, and F_{11} that utilised the smallest training datasets of size 20, this trend is not apparent; all GP systems were allowed to train for longer than those for the problems of intermediate sized training datasets of 50 points. Summarising, for the problems studied, we noted that the cases of very small and large training datasets allowed the models to train for longer before

Table 4. Summary of results. Statistics based on 50 independent runs. *Train RMSE* is the end-of-run training root mean squared error. *Test RMSE* is the generalisation root mean squared error of models selected in each run. *Model size* is the size of the best-generalising models in terms of number of tree-nodes. *Model selection generation* is the generation number when model selection is performed. Highlighting indicates that a statistically significant difference was found in the median generalisation RMSE between BVGP and SGP (Table 3).

Problem	w_b w_v	Train RMSE (median)	Test RMSE (median)	Test RMSE (minimum)	Model size (median)	Model selection generation (median)
F_2	0.2 0.8	0.29	0.32	0.26	15.00	2.00
	0.3 0.7	0.29	0.32	0.25	13.00	2.00
	0.4 0.6	0.29	0.32	0.26	31.00	1.00
	0.5 0.5	0.29	0.32	0.26	15.00	3.00
	0.6 0.4	0.29	0.32	0.27	45.00	3.00
	0.7 0.3	0.29	0.32	0.26	11.00	1.00
	0.8 0.2	0.30	0.32	0.26	15.00	2.00
	SGP	0.29	0.32	0.28	31.00	3.00
F_4	0.2 0.8	0.22	0.25	0.05	83.00	39.00
	0.3 0.7	0.22	0.26	0.04	81.00	40.00
	0.4 0.6	0.21	0.25	0.08	85.00	38.00
	0.5 0.5	0.22	0.26	0.09	87.00	40.00
	0.6 0.4	0.20	0.26	0.06	81.00	42.00
	0.7 0.3	0.23	0.27	0.14	99.00	39.00
	0.8 0.2	0.22	0.27	0.11	57.00	40.00
	SGP	0.19	0.22	0.09	81.00	37.00
F_5	0.2 0.8	3.45	3.03	2.66	19.00	3.00
	0.3 0.7	3.42	3.46	2.68	31.00	3.00
	0.4 0.6	3.61	3.41	2.82	25.00	2.00
	0.5 0.5	3.43	3.57	2.94	19.00	2.00
	0.6 0.4	3.66	3.59	2.36	19.00	2.00
	0.7 0.3	3.39	3.41	2.63	15.00	1.00
	0.8 0.2	3.39	3.75	2.52	29.00	2.00
	SGP	3.52	3.66	2.54	21.00	2.00
F_7	0.2 0.8	1.57	2.00	1.43	15.00	3.00
	0.3 0.7	1.55	2.41	1.77	17.00	4.00
	0.4 0.6	1.56	2.00	1.58	19.00	3.00
	0.5 0.5	1.56	1.97	1.69	15.00	2.00
	0.6 0.4	1.56	2.09	1.53	15.00	3.00
	0.7 0.3	1.58	1.98	1.44	21.00	4.00
	0.8 0.2	1.58	1.97	1.58	19.00	4.00
	SGP	1.58	2.65	1.73	15.00	3.00
F_8	0.2 0.8	0.50	0.68	0.68	17.00	11.00
	0.3 0.7	0.50	0.84	0.68	71.00	16.00
	0.4 0.6	0.50	17.46	0.68	57.00	11.00
	0.5 0.5	0.50	42.16	0.65	91.00	13.00
	0.6 0.4	0.50	17.74	0.63	75.00	14.00
	0.7 0.3	0.50	30.03	0.68	115.00	14.00
	0.8 0.2	0.49	27.52	0.67	123.00	27.00
	SGP	0.50	56.90	0.68	151.00	38.00
F_9	0.2 0.8	0.29	13.28	4.66	67.00	15.00
	0.3 0.7	0.30	12.76	3.03	103.00	14.00
	0.4 0.6	0.29	10.96	2.66	111.00	24.00
	0.5 0.5	0.31	11.02	1.75	97.00	18.00
	0.6 0.4	0.29	48.54	1.55	135.00	24.00
	0.7 0.3	0.29	10.52	2.64	79.00	24.00
	0.8 0.2	0.31	27.90	2.96	99.00	32.00
	SGP	0.28	13.57	2.40	119.00	31.00
F_{10}	0.2 0.8	0.22	94.31	4.13	173.00	47.00
	0.3 0.7	0.20	116.14	1.99	151.00	47.00
	0.4 0.6	0.16	40.22	5.64	167.00	46.00
	0.5 0.5	0.19	32.66	0.97	175.00	46.00
	0.6 0.4	0.19	48.36	3.69	171.00	49.00
	0.7 0.3	0.20	27.27	0.41	143.00	48.00
	0.8 0.2	0.21	85.04	0.77	157.00	43.00
	SGP	0.21	35.36	4.71	161.00	48.00
F_{11}	0.2 0.8	0.61	12.07	1.37	135.00	33.00
	0.3 0.7	0.62	23.50	1.44	127.00	40.00
	0.4 0.6	0.61	5.54	1.69	117.00	20.00
	0.5 0.5	0.63	5.25	1.26	115.00	23.00
	0.6 0.4	0.62	12.81	1.18	117.00	29.00
	0.7 0.3	0.62	18.54	1.44	93.00	26.00
	0.8 0.2	0.63	41.35	2.39	127.00	17.00
	SGP	0.62	16.36	2.32	107.00	34.00

they overfitted, as opposed to the cases of intermediate-sized training datasets, where overfitting was evident very quickly. Whether this is problem dependent remains to be seen in future studies.

6 Conclusions

The decomposition of regression error in bias and variance terms suggests that the generalisation error is due to the degree of difference between the average model (over all datasets) and the target function, as well as the degree of sensitivity of the particular model on the training dataset. We drew inspiration from this error decomposition and devised a method to relax the inherent sensitivity to the data used for training. In this method, boostrapping was employed to create an ensemble of datasets, and the variance of the error on the ensemble was used in combination with the error on the original dataset to form a new fitness function. Results on a suite of symbolic regression problems are encouraging, showing that this method is able to induce better-generalising models for most of the problems considered as opposed to standard GP.

The task of inducing a model from real-world data usually suffers from two problems: the degree to which the underlying data-generating distribution is statistically under-represented, and the degree of noise in the data points. Tackling noisy as well as unbalanced datasets is the immediate plan for future application of our method to classes of ill-defined learning environments.

Acknowledgement. This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/SRC/FM1389.

References

1. Agapitos, A., O'Neill, M., Brabazon, A.: Evolutionary Learning of Technical Trading Rules without Data-Mining Bias. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part I. LNCS, vol. 6238, pp. 294–303. Springer, Heidelberg (2010)
2. Agapitos, A., O'Neill, M., Brabazon, A., Theodoridis, T.: Maximum Margin Decision Surfaces for Increased Generalisation in Evolutionary Decision Tree Learning. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 61–72. Springer, Heidelberg (2011)
3. 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 IV. LNCS, vol. 1141, pp. 300–309. Springer, Heidelberg (1996)
4. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
5. Castelli, M., Manzoni, L., Silva, S., Vanneschi, L.: A comparison of the generalization ability of different genetic programming frameworks. In: IEEE Congress on Evolutionary Computation (CEC 2010), July 18-23. IEEE Press, Barcelona (2010)

6. Efron, B., Tibshirani, R.: An introduction to the bootstrap. Chapman and Hall (1993)
7. 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)
8. Keijzer, M., Babovic, V.: Genetic Programming, Ensemble Methods and the Bias/Variance Tradeoff - Introductory Investigations. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 76–90. Springer, Heidelberg (2000)
9. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (2008), http://www.gp-field-guide.org.uk, (With contributions by J. R. Koza)
10. Theodoridis, T., Agapitos, A., Hu, H.: A gaussian groundplan projection area model for evolving probabilistic classifiers. In: Genetic and Evolutionary Computation Conference, GECCO 2011, July 12-16. ACM, Dublin (2011) (forthcoming)
11. 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)
12. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: Tackling Overfitting in Evolutionary-Driven Financial Model Induction. In: Brabazon, A., O'Neill, M., Maringer, D. (eds.) Natural Computing in Computational Finance. SCI, vol. 380, pp. 141–161. Springer, Heidelberg (2011)
13. 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)

On Spectral Invariance of Randomized Hessian and Covariance Matrix Adaptation Schemes

Sebastian U. Stich and Christian L. Müller

MOSAIC group, Institute of Theoretical Computer Science,
and Swiss Institute of Bioinformatics,
ETH Zürich, CH 8092 Zürich, Switzerland
{sstich,christian.mueller}@inf.ethz.ch

Abstract. We evaluate the performance of several gradient-free variable-metric continuous optimization schemes on a specific set of quadratic functions. We revisit a randomized Hessian approximation scheme (D. Leventhal and A. S. Lewis. Randomized Hessian estimation and directional search, 2011), discuss its theoretical underpinnings, and introduce a novel, numerically stable implementation of the scheme (RH). For comparison we also consider closely related Covariance Matrix Adaptation (CMA) schemes. A key goal of this study is to elucidate the influence of the distribution of eigenvalues of quadratic functions on the convergence properties of the different variable-metric schemes. For this purpose we introduce a class of quadratic functions with parameterizable spectra. Our empirical study shows that (i) the performance of RH methods is less dependent on the spectral distribution than CMA schemes, (ii) that adaptive step size control is more efficient in the RH method than line search, and (iii) that the concept of the evolution path allows a paramount speed-up of CMA schemes on quadratic functions but does not alleviate the overall dependence on the eigenvalue spectrum. The present results may trigger research into the design of novel CMA update schemes with improved spectral invariance.

Keywords: gradient-free optimization, variable metric, Randomized Hessian, Covariance Matrix Adaptation, quadratic functions

1 Introduction

Randomized gradient-free (or black-box) optimization schemes are nowadays a ubiquitous tool for solving many practical problems in science and engineering where gradient or higher order information about the objective are difficult to compute or do not exist. Among the first proposed schemes that are still of considerable (theoretical) importance are adaptive step size random search (aSSRS) [1] and the (almost identical) well-known (1+1)-Evolution Strategy (ES) [2] in Evolutionary Computation (EC). To improve the poor performance of these schemes on ill-conditioned problems several fully adaptive schemes known as gradient-free variable-metric methods have been designed in the past 50 years.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 448–457, 2012.

All variable-metric schemes are iterative algorithms that share the idea of adapting a position vector and a quadratic form that defines a local metric between search points to best reflect the *local structure* of the underlying function. In gradient-free optimization two distinct classes of variable-metric methods are known: Randomized Hessian (RH) approximation schemes and Covariance Matrix Adaptation (CMA) schemes.

Randomized Hessian schemes closely follow their deterministic counterparts in nonlinear optimization. However, rather than using exact first- or second-order information they rely on approximations of gradients or Hessians found by finite differences or by estimators based on a finite collection of samples. Such approaches date back at least to 1970's [3]. In an excellent paper Marti [4] proposed several randomized Hessian update schemes taking the perspective of optimal control. Recently, Leventhal and Lewis [5] introduced a genuine RH algorithm with provable convergence guarantees which we further detail in Sec. 2.2.

Covariance Matrix Adaptation schemes follow the principle of sampling search points from the multivariate normal distribution and adapting mean and covariance according to different design principles. The first scheme of this kind, Gaussian Adaptation (GaA) [6], follows the principle of maximum entropy. A very popular modern algorithm is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [7,8]. One recent instantiation of this scheme comprises a derandomization of the sampling termed mirrored sampling [9] which we consider in Sec. 2.3.

One key strength of variable-metric methods is their invariance property to affine transformations. In addition (and more importantly in practice), they achieve the same convergence rate on all functions from the same coset modulo affine transformations, once the affine transformation has been learned in the course of optimization. How fast the different schemes learn affine transformations T is thus of fundamental importance. While theory suggests that the number of samples needed should be at least quadratic in the dimension, it is not yet fully understood how the efficiency of different variable-metric schemes depends on the *eigenvalue spectrum* of TT^T. In the EC community a small number of specific quadratic models have been proposed to probe this dependency. Key instances are the tablet, the discus, the two-axes, and the cigar function, as well as ellipsoidal functions with exponentially increasing eigenvalues [7,8]. Rather than using these specific functions we here propose a novel set of quadratic functions that varies the shape of the spectral distribution (i) in an easy *parameteric* manner and (ii) with certain common constraints that ease a simpler interpretation of performance results. We demonstrate the excellent discriminative power of the set by numerical experiments with different variable-metric schemes.

The reminder of this paper is structured as follows. In Sec. 2 we outline a standard randomized optimization framework and revisit several representative RH and CMA schemes. In Sec. 3 we introduce the design of the quadratic function set. We also review the Rosenbrock function that serves as a test model with smoothly changing Hessian. In Sec. 4 we summarize the key results of the empirical study. We discuss these results and conclude the paper in Sec. 5.

2 Variable-Metric Gradient-Free Optimization Schemes

We here present all optimization methods considered in this study. We first detail two non-adaptive randomized schemes, a specific (1+1)-ES and Random Pursuit (RP) [10], that serve as base algorithms. We then show how to couple these algorithms with Leventhal and Lewis's RH scheme. We then detail one instance of GaA [11,12] and (1,4)-CMA-ES with mirrored sampling and sequential selection [9] with and without evolution path as representative CMA schemes.

genericSearch($\mathbf{x}_0, H_0, N, [\epsilon, \mu, \sigma_0, p]$)

for $k = 1$ *to* N **do**
  if *variable metric* **then**
    $H_k \leftarrow \text{updateHess}(H_{k-1}, \mathbf{x}_{k-1}, \epsilon)$
  else $H_k \leftarrow H_{k-1}$
  $\mathbf{u}_k \sim \mathcal{N}(0, H_k^{-1})$
  if *line search* **then**
    $\mathbf{x}_k \leftarrow \text{lineSearch}(\mathbf{x}_{k-1}, \mathbf{u}_k/\|\mathbf{u}_k\|, \mu)$
  else $(\mathbf{x}_k, \sigma_k) \leftarrow \text{aSS}(\mathbf{x}_{k-1}, \mathbf{u}_k, \sigma_{k-1}, p)$
return $\mathbf{x}_N$

updateHess($H, \mathbf{x}, \epsilon$)

$\mathbf{u} \sim S^{n-1}$
$\Delta_u \leftarrow \frac{f(\mathbf{x}+\epsilon\mathbf{u})-2f(\mathbf{x})+f(\mathbf{x}-\epsilon\mathbf{u})}{\epsilon^2} - \mathbf{u}^T H \mathbf{u}$
if $J := H + \Delta_u \cdot \mathbf{u}\mathbf{u}^T$ *psd* **then**
  $H_+ \leftarrow H + \Delta_u \cdot \mathbf{u}\mathbf{u}^T$
else
  $v \leftarrow \text{smallestEVec}(J)$
  $\Delta_v \leftarrow \frac{f(\mathbf{x}+\epsilon\mathbf{v})-2f(\mathbf{x})+f(\mathbf{x}-\epsilon\mathbf{v})}{\epsilon^2} - \mathbf{v}^T J \mathbf{v}$
  $H_+ \leftarrow \left(H + \Delta_v \cdot \mathbf{v}\mathbf{v}^T\right) + \Delta_u \cdot \mathbf{u}\mathbf{u}^T$
return H_+

aSS($\mathbf{x}, \mathbf{u}, \sigma, p$) *(adaptive step size)*

if $f(\mathbf{x} + \sigma\mathbf{u}) \leq f(\mathbf{x})$ **then**
  $\mathbf{x}_+ \leftarrow \mathbf{x} + \sigma\mathbf{u}$; $\sigma_+ \leftarrow \sigma \cdot \exp(1/3)$
else
  $\mathbf{x}_+ \leftarrow \mathbf{x}$; $\sigma_+ \leftarrow \sigma \cdot \exp\left(-\frac{p}{3(1-p)}\right)$
return $(\mathbf{x}_+, \sigma_+)$

lineSearch($\mathbf{x}, \mathbf{u}, \mu$)

let $\mathbf{x}^* := \mathbf{x} + \arg\min_\lambda f(\mathbf{x} + \lambda\mathbf{u}) \cdot \mathbf{u}$
if *relative accuracy* **then**
  find $\mathbf{x}_+ \in \left[(1-\mu)\mathbf{x} + \mu\mathbf{x}^*, \mathbf{x}^*\right]$
else
  find $\mathbf{x}_+ \in \left[\mathbf{x}^* - \mu\mathbf{u}, \mathbf{x}^* + \mu\mathbf{u}\right]$
return x_+

Fig. 1. Basic building blocks for variable-metric gradient-free optimization

2.1 Isotropic Gradient-Free Optimization Schemes

We consider two basic optimization schemes that iteratively generate a sequence of approximate solutions to the optimization problem $\min_{\mathbf{x}} f(\mathbf{x})$ for $f\colon \mathbb{R}^n \mapsto \mathbb{R}$. In each step a search direction is drawn $\mathbf{u} \sim \mathcal{N}(0, \mathbb{1}_n)$. The choice of the step size $\lambda \in \mathbb{R}$ is the key difference between the two schemes.

In Random Pursuit (RP), first proposed in [13] and analyzed in [10], the step size λ is determined by minimizing the objective function in direction $\mathbf{u}$, i.e. $\lambda \approx \arg\min_c f(\mathbf{x} + c\mathbf{u})$. For quadratic functions $f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x}$ with Hessian A, the expected one-step progress can be estimated as:

$$\mathbb{E}\left[f(\mathbf{x}_+) \mid \mathbf{x}\right] \leq \left(1 - \kappa(A^{-1})/n\right) f(\mathbf{x}), \tag{1}$$

where $\mathbf{x}$ is the current iterate, $\mathbf{x}_+ := \mathbf{x} + \lambda\mathbf{u}$ the next iterate, and $\kappa(A^{-1})$ denotes the condition number of A^{-1}. This statement can also be generalized to arbitrary smooth convex functions [10]. Stich et al. [10] showed that both relative and absolute errors in the line search do not hamper the convergence guarantees of RP. We use the built-in MATLAB routine `fminunc.m` with `optimset('TolX'=1E-4)` as numerical gradient-free line search method with absolute tolerance μ.

In the $(1+1)$-ES the step size is dynamically controlled such as to approximately guarantee a certain probability p of finding an improving iterate. Depending on the underlying test function different optimality conditions can be formulated for the probability p. Schumer and Steiglitz [1] suggest the setting $p = 0.27$ which is considered throughout this work. We use immediate exponential step size control as explicitly formulated in the aSS sub-routine in Fig. 1. Jägersküpper [14] showed that, for quadratic functions, the dependence of the expected one-step progress of the (1+1)-ES on $\kappa(A)$ is almost identical to the one shown in Eq. (1).

2.2 Randomized Hessian Approximation Schemes

Assume that the random search direction $\mathbf{u}$ is not chosen from the standard normal distribution, but rather $\mathbf{u} \sim \mathcal{N}(0, H^{-1})$ for a positive definite matrix H. Then a standard analysis [5] shows that the factor of the one-step RP progress in Eq. (1) changes to $(1 - \kappa(HA^{-1})/n)$ for quadratic functions. A refined analysis by Stich et. al. [15] shows a dependence on $\mathrm{Tr}(AH^{-1})$. Hence, if a suitable matrix H with $AH^{-1} \approx \mathbb{1}_n$ can be found, the convergence of RP will be linear with the optimal rate $(1 - 1/n)$. Leventhal and Lewis [5] proposed the following iterative scheme to generate a sequence of Hessian estimates H that converge to A. In each step, a new iterate H_+ is generated as follows:

$$H_+ = H + \mathbf{u}^T (A - H)\, \mathbf{u} \cdot \mathbf{u}\mathbf{u}^T , \tag{2}$$

where $\mathbf{u} \sim S^{n-1}$ is a uniform random unit vector and $\mathbf{u}^T A\mathbf{u}$ is calculated by:

$$\mathbf{u}^T A\mathbf{u} = \left(f(\mathbf{x} + \epsilon\mathbf{u}) - 2f(\mathbf{x}) + f(\mathbf{x} - \epsilon\mathbf{u})\right)/\epsilon^2 , \tag{3}$$

for arbitrary $\epsilon > 0$. Whilst equality only holds for quadratic functions, for general twice differentiable functions the value can be approximated by choosing ϵ sufficiently small. It can be shown [5, Thm. 1] that

$$\mathbb{E}\left[\|H_+ - A\|_F\right] \leq (1 - 2/(n(n+2)))\, \|H - A\|_F , \tag{4}$$

holds, and the sequence $(H_k)_{k\geq 1}$ of estimates $H_k \to A$ a.s for $(k \to \infty)$.

Unfortunately, H_+ generated by Eq. (2) is not necessarily positive definite. We thus propose an additional correction step in our implementation of the update. If H_+ is not positive definite we perform a second deterministic update in the direction of the eigenvector that corresponds to the smallest (and the only negative) eigenvalue of H_+. By standard results from matrix perturbation theory, the resulting "twice updated" matrix will be positive (semi-)definite. The algorithm is also illustrated in Fig. 1. As we directly operate on the Cholesky decomposition of H, the condition on line 3 can be efficiently checked. For all quadratic functions we arbitrarily set $\epsilon = 1$, for the Rosenbrock function (see Section 3) we use $\epsilon = 1\text{E-}9$.

Combining the Hessian update with the different step size update schemes from the previous section, we arrive at two variable-metric gradient-free optimization schemes. We will refer to them as RH RP (Randomized Hessian Random Pursuit) and RH (1+1) (Randomized Hessian with aSSRS).

2.3 Covariance Matrix Adaptation Schemes

CMA schemes are conceptually different from the presented RH scheme. New search points are sampled from a multivariate normal distribution whose parameter are updated in each iteration based on the information present in the evaluated samples. Many different adaptation schemes exist today. The covariance matrix can be adapted using different rank-1 [6,7] or rank-k updates [8]. In addition, the CMA-ES scheme is augmented by an auxiliary variable called evolution path that takes into account the correlation of successive means taken over a finite horizon. This is similar in spirit to Rao-Blackwellization techniques in Marko Chain Monte Carlo methods [16] and Polyak's heavy ball method in first-order optimization [17]. We here select two specific instances of CMA schemes: (i) one that is as close as possible to the described RH scheme and (ii) one that is the fastest scheme for quadratic functions known today. The first scheme is a variant of GaA [12] in the (1+1) setting. In every iteration a single sample $\mathbf{x}_k \sim \mathcal{N}(\mathbf{m}_k, \sigma_k^2 C_k)$ is drawn. The aSS sub-routine is employed for step size adaptation. If $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k)$ the mean $\mathbf{m}_{k+1} = \mathbf{x}_{k+1}$ and the covariance matrix is updated according to $C_k = (1-\alpha)C_k + \alpha(\mathbf{x}_{k+1} - \mathbf{m}_k)(\mathbf{x}_{k+1} - \mathbf{m}_k)^T$ with $\alpha = \log(n+1)/(n+1)^2$ [12]. This constitutes the simplest covariance update without evolution path. The second scheme considered here is the (1,4)-CMA-ES with mirrored sampling and sequential selection. Brockhoff and co-workers state that this scheme "is unbiased and appears to be faster, more robust, and as local as the (1+1)-CMA-ES" [9]. We also refer to [9] for a full description of this scheme and all parameter settings used. For the (1,4)-CMA-ES scheme has been retrieved from `http://coco.gforge.inria.fr/doku.php?id=bbob-2010-results`. We used the GaA code from `http://www.mosaic.ethz.ch/Downloads/GaA`.

3 Benchmark Functions

For the presented variable-metric methods there is either theoretical or a large body of empirical evidence that, on quadratic functions, the sequence of estimated Hessians (or inverse Hessians, respectively) will converge after sufficiently many iterations. A reasonable assumption is that the difficulty of the approximation task is mainly determined by the distribution of the eigenvalues of the underlying Hessian. Thus far, this influence has been extensively studied for CMA schemes on specific quadratic model functions such as the tablet, the cigar, or ellipsoidal functions with exponentially increasing eigenvalues (see, e.g., [7,8]). The exact dependency of variable-metric schemes on the spectral distribution remains, however, largely elusive because the spectral properties such as trace and condition are not constant across experiments. For RH schemes, we are not aware of any systematic empirical study. From the theory of RH schemes we know, however, that the expected progress for a fixed Hessian estimate H depends on $\kappa(AH^{-1})$ as well as on $\mathrm{Tr}(AH^{-1})$ where A denotes the Hessian [5,15]. We thus propose a class of quadratic functions with different spectra under the constraint of equal trace and condition number L. The functions are constructed as follows: We choose the distribution of the Hessian eigenvalues according to

Table 1. List of benchmark functions. All functions are quadratic except f_{Rosen}. For $f_{\text{Sigm}(a)}$ we use $a = 15, 8, 5, 2.8$ and for $f_{\text{Flat}(a)}$ we use $a = 6, 3.2, 2, 1.25$. The spectra of all the quadratic functions are depicted in Fig. 2b.

$$f_{\text{Sigm}(a)}(\mathbf{x}) = \sum_{i=1}^{n} \text{normalize}_i\left(\left(1 + e^{a - \frac{2a(t-1)}{n-1}}\right)^{-1} + \tfrac{1}{2}\right)(x_i - 1)^2$$

$$f_{\text{Flat}(a)}(\mathbf{x}) = \sum_{i=1}^{n} \text{normalize}_i\left(-\log\left(\left(10^{-a} + \tfrac{(t-1)(1-2\cdot 10^{-a})}{n-1}\right)^{-1} - 1\right)\right)(x_i - 1)^2$$

$$f_{\text{Lin}}(\mathbf{x}) = \sum_{i=1}^{n} \text{normalize}_i\left(\tfrac{2t}{n+1} - 1\right)(x_i - 1)^2$$

$$f_{\text{Nes}}(\mathbf{x}) = \sum_{i=1}^{n} \text{normalize}_i\left(\sin\left(\tfrac{t\pi}{n+1} - \tfrac{\pi}{2}\right)\right)(x_i - 1)^2$$

$$f_{\text{Rosen}}(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$$

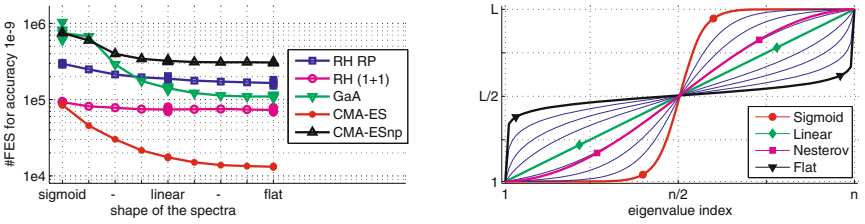
$$\text{normalize}_i(f(t)) = \tfrac{L-1}{2}\tfrac{f(i)}{|f(1)|} + \tfrac{L+1}{2}$$

three specific parametric functions outlined below. We then normalize the spectra such that the smallest eigenvalue equals 1, the largest equals L, and the trace equals $n(L+1)/2$ where n denotes the dimension. The function set is summarized in Tab. 1. For f_{Lin} the eigenvalues are linearly spaced. For f_{Sigm} the eigenvalues lie on the sigmoidal curve $(1 + e^{-t})^{-1}$ resulting in many eigenvalues being close to 1 or close to L with only a few intermediate eigenvalues. By distributing the eigenvalues proportional to the inverse of the sigmoid function $\log(1/t - 1)$ we find a family of suitable quadratic functions where most eigenvalues are concentrated around the mean $L/2$. The exact parameterizations are summarized in Tab. 1. The shape of the spectra is depicted in Fig. 2b. For large a we note that (i) $f_{\text{Sigm(a)}}$ becomes similar to the two-axes function [8] (half of the eigenvalues are 1, half of them are L) and (ii) $f_{\text{Flat(a)}}$ gets close to a cigar-like function (with one small eigenvalue and all others on the order of L). Another important feature of our parametric family is the fact that the sigmoidal function can closely approximate Nesterov's worst case function f_{Nes} [18] which has been used to show a lower complexity bound for first-order optimization (see again Fig. 2b for a sketch). Note that the present trace constraint prohibits the design of quadratic functions with exponentially distributed eigenvalues.

Finally, we also include the standard Rosenbrock function f_{Rosen} in the test set. The function serves as a test model with smoothly changing Hessian in order to study the valley-following abilities of the different variable-metric schemes.

4 Empirical Study

We now highlight the key results of our empirical study. All algorithms and functions have been implemented in MATLAB and will be made publicly available at the authors' website. The (1,4)-CMA-ES with mirrored sampling and sequential selection (referred to as CMA-ES in the following) has been run both with and without evolution path (setting `CMA.ccum=1` in the referred MATLAB code). The latter variant is referred to as CMA-ESnp. For all performed experiments the initial settings were $\mathbf{x}_0 = \mathbf{0}$, $H_0 = \mathbb{1}_n$ ($\mathbf{m}_0 = \mathbf{0}$, $C_0 = \mathbb{1}_n$, respectively).



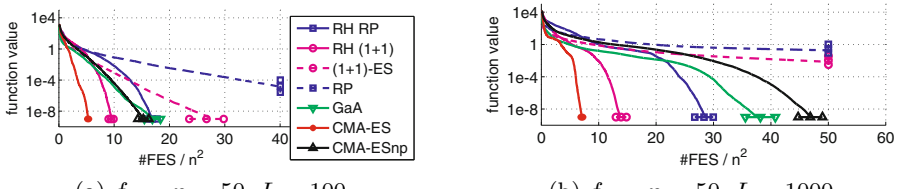
(a) Comparison for different spectra. (b) The spectra of the functions.

Fig. 2. (a): Relation between method performance and spectral distribution in $n = 50$ for $L = 1\text{E}6$. We recorded #FES needed to reach accuracy 1E-9 on all parametrized functions f_{Sigm}, f_{Flat} and f_{Lin}; the median of 51 runs is indicated by a marker. (b): Shape of the spectra of the quadratic benchmark functions. Thin blue lines show $f_{\text{Sigm(a)}}$ and $f_{\text{Flat(a)}}$ for intermediate a values.

The initial step size of the algorithms with adaptive step size control was (empirically) set such that the target success probability $p = 0.27$ is met for $\mathbf{x}_0$. As performance measure we count the number of function evaluations (#FES) needed to reach a target function value below 1E-9.

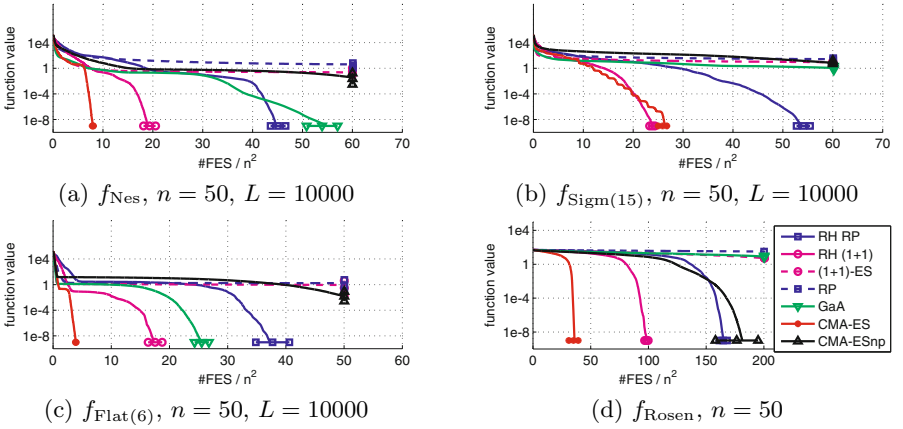
We first demonstrate the general influence of the spectral distribution on the performance of all introduced variable-metric schemes. Experimental set-up and results are summarized in Fig. 2. For all CMA schemes (CMA-ES, CMA-ESnp, and GaA) we see a strong monotone dependence of their performance on the spectral shape. The sigmoidal-shaped eigenspectrum presents the hardest problem, the flat spectrum the easiest. Both CMA-ES and GaA show the strongest run time dependence on the spectra with CMA-ES being the fastest algorithm on all functions and CMA-ESnp the slowest one. The performance of both RH schemes is much less dependent on the shape with RH (1+1) being almost invariant to the spectral distribution. We observe that RH (1+1) achieves the same performance on $f_{\text{Sigm}(15)}$ (the leftmost datum in Fig. 2a) as CMA-ES.

To study the influence of the condition number L on the qualitative convergence behavior of the different algorithms we present results on f_{Nes} as representative example in Figs. 3 and 4a for fixed dimension $n = 50$. The same qualitative behavior has been observed for the other quadratic functions.



(a) f_{Nes}, $n = 50$, $L = 100$ (b) f_{Nes}, $n = 50$, $L = 1000$

Fig. 3. Evolution of function value vs. #FES on f_{Nes} for $L = 100$ (a) and $L = 1000$ (b). We recorded #FES needed to reach accuracy 1E-9. The median trajectory of 11 runs is depicted; mean and one standard deviation are indicated by markers.

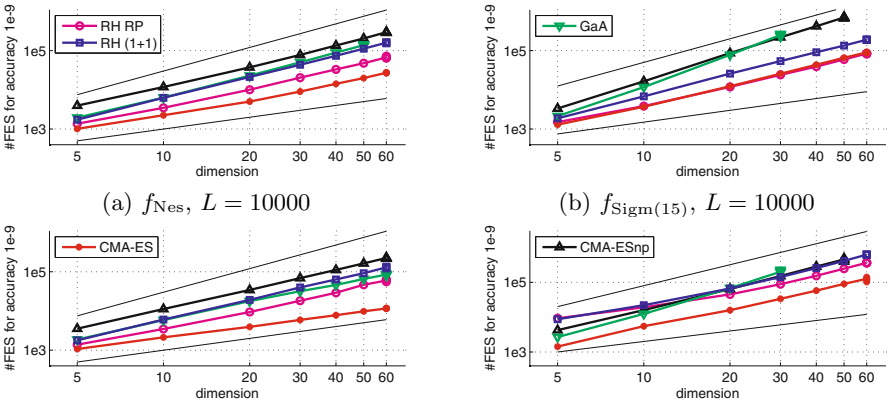


(a) f_{Nes}, $n = 50$, $L = 10000$ (b) $f_{\mathrm{Sigm}(15)}$, $n = 50$, $L = 10000$

(c) $f_{\mathrm{Flat}(6)}$, $n = 50$, $L = 10000$ (d) f_{Rosen}, $n = 50$

Fig. 4. Evolution of function value vs. #FES for different functions. We recorded #FES needed to reach accuracy 1E-9. The median trajectory of 11 runs is depicted; mean and one standard deviation are indicated by markers.

We see that the non-adaptive schemes RP and (1+1)-ES are (as expected) not even competitive for $L = 100$. We will thus concentrate on variable-metric schemes in the further discussion. For $L \geq 1000$ we observe that the convergence behavior of all variable-metric methods can be divided into three phases, (i) an initial short tune-in phase with rapid progress, (ii) a learning (or adaptation) phase with marginal progress in function value (of length quadratic in n), and (iii) a convergence phase with strong function value decrease (of length linear in n). Moreover, we see that the slope of the trajectory *at the level of the target accuracy* is distinct for all schemes. This measured convergence rate reflects the efficiency of the adaptation process of the different schemes at this function value level. CMA-ES and RH (1+1) show the steepest descent, CMA-ESnp and GaA the flattest one. For $L = 10000$ CMA-ESnp and the non-adaptive methods do not reach the target accuracy within a FES budget of $60n^2$. These observations are generally confirmed on other quadratic functions having different spectral shapes with a few notable exceptions. We here exemplify the performance of the schemes on the two most extreme functions $f_{\mathrm{Sigm}(15)}$ and $f_{\mathrm{Flat}(6)}$ (with $L = 10000$ in $n = 50$) as well as on f_{Rosen} (as shown in Fig. 4). On $f_{\mathrm{Flat}(6)}$ (cf. Fig. 4c) the Hessian is well-approximated by all convergent schemes. The convergence rate in phase (iii) is best for CMA-ES followed by RH (1+1) and GaA. RH RP' convergence takes longer because per line search 5-10 FES are needed on average. CMA-ESnp is still in the adaptation phase within the displayed FES budget. On $f_{\mathrm{Sigm}(15)}$ (cf. Fig. 4b) we observe that CMA-ES' convergence rate is slower than the one of RH (1+1) above accuracy 1E-7 eventually converging at optimal rate below this level. This indicates that CMA-ES is still in adaptation phase even at low function value level. Both CMA-ESnp and GaA are still in the adaptation phase within the displayed FES budget. Inspection of the convergence trajectories also reveals that the length of learning phase is responsible for CMA-ES' observed dependence on the spectral shape.



(a) f_{Nes}, $L = 10000$ (b) $f_{\mathrm{Sigm}(15)}$, $L = 10000$

(c) $f_{\mathrm{Flat}(6)}$, $L = 10000$ (d) f_{Rosen}

Fig. 5. #FES to reach the target accuracy vs. dimension n in log-log scale. The median of 11 runs is depicted by a marker for all converged runs within the considered #FES budget. Thin lines indicate quadratic scaling (top) or linear scaling (bottom) .

The experiments on f_{Rosen} confirm that all variable-metric schemes (except GaA) can efficiently learn a smoothly changing Hessian (without tune-in phase) confirming and extending known results for RH schemes [5] and CMA schemes [7]. We finally show the scaling behavior of the algorithms on selected functions in Fig. 5. All algorithms show the expected quadratic scaling with dimension (for $n \geq 20$) with two notable exceptions. While GaA and CMA-ESnp on $f_{\mathrm{Sigm}(15)}$ and GaA on f_{Rosen} exhibit scaling of higher order than quadratic, CMA-ES shows super-linear convergence on $f_{\mathrm{Flat}(6)}$. The latter result is in full agreement with the empirical tests of CMA-ES on the cigar function [7,8].

5 Discussion and Conclusions

We have empirically tested the performance of several randomized gradient-free variable-metric optimization schemes on a novel set of quadratic functions whose spectral distribution ranges (for any fixed dimension n and condition number L) from a near-flat distribution to a sigmoidal shape under constant trace constraint. Using this benchmark set we have been able to show a clear *monotonic dependence* of the performance of CMA schemes on the shape of the spectrum. From the data we also conclude that the concept of the evolution path allows a paramount speed-up of CMA schemes but does not alleviate the dependence on the eigenvalue spectrum. The presented Randomized Hessian (RH) approximation schemes [5], on the other hand, have been shown to be less dependent or almost invariant to the specific distribution of eigenvalues. Our empirical results also indicate that coupling our novel, numerically stable implementation of the RH scheme with adaptive step size control is more efficient than a scheme with approximate line search on all tested problems. We believe that the present results may trigger research into the design of novel CMA update schemes with improved spectral invariance. We also advocate the embedding

of the proposed function set (most prominently the sigmoidal ones) in modern black-box optimization benchmark test suites. Investigating quadratic function sets under constant determinant and condition constraints (thus allowing exponentially distributed eigenvalues) will be subject of future research.

References

1. Schumer, M., Steiglitz, K.: Adaptive step size random search. IEEE Transactions on Automatic Control 13(3), 270–276 (1968)
2. Rechenberg, I.: Evolutionsstrategie; Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog (1973)
3. Betro, B., De Biase, L.: A Newton-like method for stochastic optimization. In: Towards Global Optimization, vol. 2, pp. 269–289. North-Holland (1978)
4. Marti, K.: Controlled random search procedures for global optimization. In: Stochastic Optimization. Lecture Notes in Control and Information Sciences, vol. 81, pp. 457–474. Springer (1986)
5. Leventhal, D., Lewis, A.S.: Randomized Hessian estimation and directional search. Optimization 60(3), 329–345 (2011)
6. Kjellström, G., Taxen, L.: Stochastic Optimization in System Design. IEEE Trans. Circ. and Syst. 28(7) (July 1981)
7. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaption in Evolution Strategies. Evolutionary Computation 9(2), 159–195 (2001)
8. Hansen, N., Muller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol. Comput. 11(1), 1–18 (2003)
9. Brockhoff, D., Auger, A., Hansen, N., Arnold, D.V., Hohm, T.: Mirrored Sampling and Sequential Selection for Evolution Strategies. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 11–21. Springer, Heidelberg (2010)
10. Stich, S.U., Müller, C.L., Gärtner, B.: Optimization of convex functions with Random Pursuit (2011), http://arxiv.org/abs/1111.0194
11. Müller, C.L., Sbalzarini, I.F.: Gaussian Adaptation Revisited – An Entropic View on Covariance Matrix Adaptation. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., Neri, F., Preuß, M., Togelius, J., Yannakakis, G.N. (eds.) EvoApplicatons 2010, Part I. LNCS, vol. 6024, pp. 432–441. Springer, Heidelberg (2010)
12. Müller, C.L., Sbalzarini, I.F.: Gaussian Adaptation as a unifying framework for continuous black-box optimization and adaptive Monte Carlo sampling. In: 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2010)
13. Mutseniyeks, V.A., Rastrigin, L.A.: Extremal control of continuous multiparameter systems by the method of random search. Eng.Cyb. 1, 82–90 (1964)
14. Jägersküpper, J.: Rigorous Runtime Analysis of the (1+1) ES: 1/5-Rule and Ellipsoidal Fitness Landscapes. In: Wright, A.H., Vose, M.D., De Jong, K.A., Schmitt, L.M. (eds.) FOGA 2005. LNCS, vol. 3469, pp. 260–281. Springer, Heidelberg (2005)
15. Stich, S.U., Gärtner, B., Müller, C.L.: Variable Metric Random Pursuit. In Preparation for Math. Prog. (2012)
16. Andrieu, C., Thoms, J.: A tutorial on adaptive MCMC. Statistics and Computing 18(4), 343–373 (2008)
17. Polyak, B.: Introduction to Optimization. Optimization Software - Inc., Publications Division, New York (1987)
18. Nesterov, Y.: Introductory Lectures on Convex Optimization. Kluwer, Boston (2004)

Variable Neighborhood Search and GRASP for Three-Layer Hierarchical Ring Network Design*

Christian Schauer and Günther R. Raidl

Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria
{schauer,raidl}@ads.tuwien.ac.at

Abstract. We introduce the Three-Layer Hierarchical Ring Network Design Problem, which arises especially in the design of large telecommunication networks. The aim is to connect nodes that are assigned to three different layers using rings of bounded length. We present tailored Variable Neighborhood Search (VNS) and GRASP approaches to solve large instances of this problem heuristically, and discuss computational results indicating the VNS' superiority.

1 Introduction

In this paper we present the *Three-Layer Hierarchical Ring Network Design* (3-LHRND) problem, which belongs to the research field of network design and has not been considered in the literature before. 3-LHRND finds applications in the planning of larger hierarchical communication networks where survivability in case of failures plays a major role. The nodes of the network are assigned to different layers—in our case three. The aim is to connect these nodes within each layer and, additionally, the layers among each other hierarchically using rings of bounded length for the matter of survivability. As a possible application one might imagine a telecommunication network, where different technologies are interconnected, e.g., high-speed fiber for layer 1, low-speed fiber for layer 2 and copper for layer 3.

In the context of survivability fault tolerance is an important issue especially in the case of wide area networks to guarantee high reliability. A common way to achieve these demands is the use of so-called *self healing rings* (often referred to as rings for short) that ensure connectivity of two nodes in case of the failure of a third node due to the possible routing in two different directions. Rings are, in fact, the simplest node-biconnected structures.

With the increasing size of networks a single ring connecting all nodes would, however, not be efficient anymore. Due to the large diameter of such a network, capacity requirements of single links as well as communication delays would soon be too high. Furthermore, simultaneous failures in more than one node would in general disconnect large parts of the network. This issues can be addressed by

* This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT10-027.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 458–467, 2012.

dividing the network into subnetworks of limited size, i.e., the network consists of smaller rings that are hierarchically interconnected. Interconnection nodes are often referred to as *hub nodes*. When the interconnection is realized by ring structures the network is called a *hierarchical ring network*. To ensure survivability also in case of failures in hub nodes, the rings are interconnected via two different nodes—a so-called *dual homing* approach. In our case we consider a hierarchy spanning nodes on three different layers using dual homing, and consequently we refer to this problem as the *Three-Layer Hierarchical Ring Network Design problem* (3-LHRND). While much work already exists for related problems, 3-LHRND has so far not been addressed. As we will argue in Section 3, 3-LHRND is NP-hard and also difficult to solve in practice. In order to approximately solve larger instances, we propose a *Variable Neighborhood Search* (VNS) and a *Greedy Randomized Adaptive Search Procedure* (GRASP). Both exploit the special structure of the problem in various ways.

The rest of the paper is organized as follows. In Section 2 we summarize related work. Section 3 defines the 3-LHRND, and Section 4 presents our VNS and GRASP. Computational results are discussed in Section 5. Finally, Section 6 concludes this contribution, also pointing out suggestions for future research.

2 Related Work

While network design is a fast growing field of research, 3-LHRND with its strict definition and constraints has not been considered yet. Nevertheless, a lot of work has been contributed to similar problems, which contain some (but never all) of the aspects of 3-LHRND.

In the field of network design, the hierarchical/layered aspect has first been explicitly considered in 1986, when Current *et al.* [5] introduced the *Hierarchical Network Design Problem* (HNDP). In the HNDP the network consists of two primary nodes that are connected to a path and secondary nodes that connect to this path such that the whole network comprises to a tree. The authors present an Integer Linear Programming (ILP) formulation. Additionally, they designed a heuristic for the HNDP based on a K-shortest path algorithm, to find the best path connecting the primary nodes, and a minimum spanning tree heuristic, to connect the remaining nodes to the best path found.

Balakrishnan *et al.* introduced the *Multi-Level Network Design* (MLND) in [3], which is a generalization of the well-known Steiner network problem [6]. By definition nodes are assigned to L different levels in MLND. In the core of their work, the authors focus on the $L = 2$ case and first present an ILP formulation, which they later extend to a multi-commodity flow formulation.

In [12] Thomadsen and Stidsen give a detailed overview about the advantages of rings in survivable networks and present a Branch-and-Price approach for the Hierarchical Ring Network Problem with two levels using single homing, i.e., each ring connects to one node of the interconnection ring. The authors assume that node-to-layer assignments are not prespecified but are to be found during the design process.

In the past the attention in hierarchical network design has been primarily drawn to two level problems. However, in recent years the interest in more general multi (> 2) level problems has increased. Trampont, Destré and Faye [13] describe a three level problem, like our 3-LHRND but with tree structure, where the authors study a variant of the multi-source Weber problem. In this case terminals (layer 3) must be connected to a central equipment (layer 1) via concentrators (layer 2), which are not fixed at the beginning. The objective is to find the best location for the concentrators, i.e., determine the layer 2, such that the overall costs of the network are minimized. To solve this problem the authors present two stabilized column generation approaches.

The *Capacitated m-Ring-Star Problem* was introduced by Baldacci *et al.* in [4]. There the nodes are partitioned into three subsets, one depot node, customer nodes, and transit points (Steiner nodes). The network consists of m rings of bounded length and edges connected to the ring. All rings must contain the depot node and connect the customers directly or via a transit point (i.e., the additional edges). The authors compare two mathematical programming formulations, namely a two-indexed and a two-commodity flow formulation.

In [2] Aringhieri *et al.* apply different Metaheuristics to solve two ring network problems. One is comparable to the previously mentioned [12], the other does not use an interconnection ring but a set of paths to connect the level two rings. The authors designed a construction heuristic and two neighborhood structures embedded in a Tabu Search. Moreover, they consider Path Relinking, "eXploring Tabu Search", and Scatter Search approaches.

From another field of research originates the *Two-Echelon Location-Routing Problem* [11], which is a combination of the Facility Location Problem (FLP) and the Vehicle Routing Problem (VRP) but, nevertheless, shares similarities with 3-LHRND. Here the node set is partitioned in platforms (layer 1), satellites (layer 2) and customers (layer 3). In this case the locations for the platforms and satellites to be opened must be determined (i.e., the FLP aspect). Additionally, two different vehicle fleets transport goods either from the platforms to the satellites or from the satellites to the costumers (i.e., the VRP aspect). The main differences to 3-LHRND are that the platforms need not be connected and not all platforms and satellites need to be opened but only the beneficial (or sometimes mandatory) ones. The authors present a VNS with a total of 21 specific neighborhood structures.

3 Three-Layer Hierarchical Ring Network Design

This section introduces the 3-LHRND with all its details. We are given an undirected, complete graph $G = (V, E)$ with vertex set V, edge set E, and a cost function that assigns costs $c_{ij} \geq 0$ to each edge $(i, j) \in E$. Each vertex is assigned to one of three layers, i.e., the vertex set is partitioned into three disjoint subsets V_1, V_2 and V_3 with $V_1 \cup V_2 \cup V_3 = V$ and $V_i \cap V_j = \emptyset,\ \forall i, j \in \{1, 2, 3\},\ i \neq j$. A feasible solution to 3-LHRND is a subgraph $G_L = (V, E_L)$ connecting all the nodes V and satisfying the following constraints; see Figure 1 for an example.

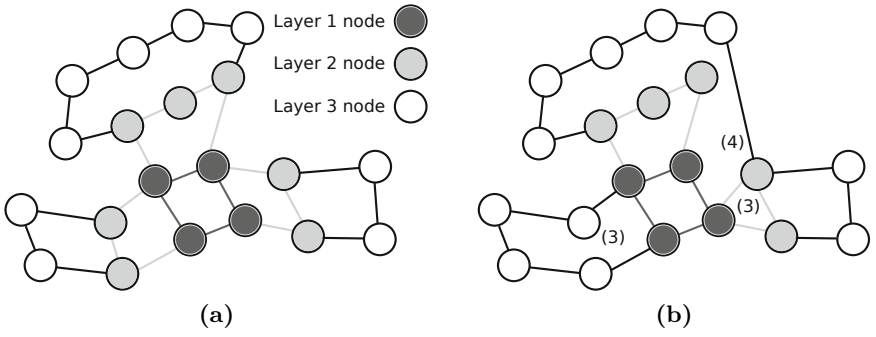


Fig. 1. Schematic representations of (a) a feasible solution and (b) an infeasible solution (the numbers in parentheses indicate the violated constraints)

1. The nodes in V_1 are connected by a single independent ring containing no other node.
2. The nodes from layers 2 and 3 are connected by two respective sets of paths, containing no nodes from other layers. Each node appears in exactly one path.
3. The end nodes of each path at layers 2 and 3 are further connected to two different nodes (hubs) in the directly preceding layer; i.e., dual homing is realized. We refer to the edges connecting paths to hubs as *uplinks*. Consequently, there are no edges directly connecting a node from layer 1 to a node from layer 3.
4. The two hub nodes a path is connected to must themselves be connected by a simple path at their layer; i.e., the connection to a ring may not be established via more than two layers.
5. The lengths of layer $k \in \{2, 3\}$ paths in terms of the number of nodes is bounded below and above by specified limits $b_k^{\mathrm{l}} \geq 2$ and $b_k^{\mathrm{u}} \geq 2$, respectively.

The objective is to find a feasible solution with minimum total costs $c(E_L) = \sum_{(i,j)\in E_L} c_{ij}$.

Considering these definitions, we observe that finding the layer 1 ring resembles the classical Traveling Salesman Problem (TSP), which can be solved independently. In contrast, optimal structures of layers 2 and 3 strongly depend on each other. Only a suitable, concerted choice of layer 2 and layer 3 paths allow for efficient uplinks and overall connectivity. Thus, the layers 2 and 3 cannot be treated separately. The rings connecting layer 2 with layer 1 consist each of at most $\lfloor |V_1|/2 \rfloor + b_2^{\mathrm{u}} + 1$ nodes and edges, while the rings connecting layer 3 and layer 2 have up to $b_2^{\mathrm{u}} + b_3^{\mathrm{u}}$ nodes/edges.

3-LHRND obviously is NP-hard even when looking at each layer independently: As mentioned, for layer 1 the subproblem directly corresponds to the TSP. Concerning layers 2 and 3, the classical capacitated vehicle routing problem (CVRP) can be reduced to each by assuming all nodes of the preceding layer are connected via a single ring and together represent the CVRP's depot.

4 A VNS and GRASP for 3-LHRND

As already mentioned, the problem to connect all layer 1 nodes to a single ring resembles the TSP and can be solved independently. We apply the Concorde TSP solver [1] and focus in the following only on the more interesting layers 2 and 3.

Due to the 3-LHRND's complexity and relations to other network design problems and CVRP variants, we decided to solve it approximately using VNS and GRASP, as these techniques are known to work well in these domains, see, e.g., [11]. At first we designed a simple construction heuristic, which we used to create initial solutions for the VNS and then randomized for the GRASP. For general introductions to VNS and GRASP, we refer to [8,10].

4.1 Construction Heuristic

Our construction heuristic is inspired by the simple nearest-neighbor heuristic for the TSP. To create the layer 2 paths we start with an unvisited layer 2 node i, search for the nearest hub node $h \in V_1$ according to the edge costs c_{ih}, and add the uplink e_{ih} to our solution. Then we determine in layer 2 the closest unvisited node j to our current node i, add the edge e_{ij} to our solution, mark i as visited, and make j our current node. We repeat this procedure until the given upper bound for layer 2 paths b_2^{u} is reached. Then we search for the nearest hub node different from h back to layer 1 and add this uplink to our solution. Following this approach we add layer 2 paths to our solution until all layer 2 nodes are marked as visited. In case that the length of the last path would be less than the lower bound for layer 2 paths b_2^{l} we make the second to last path shorter to satisfy all constraints.

To create the layer 3 paths we apply the same procedure, but now we further have to ensure that the second uplink connects to the same layer 2 path as the first uplink.

4.2 Variable Neighborhood Descent

Variable Neighborhood Descent (VND) [8] extends simple local search by systematically considering a set of different neighborhood structures in a deterministic way until a solution is reached that is locally optimal w.r.t. all of them. Thus, the improvement potential strongly relies on the neighborhood structures and the order of their application.

Our VND, which is used within VNS as well as GRASP, searches eight neighborhood structures that are induced by the following operators in the given order.

Two-Edge-Exchange (2EE): Based on the well known operator for the TSP this operator investigates all feasible candidate solutions that differ in at most two edges. In this case 2EE is applied to each layer 2 and layer 3 path separately starting with the first uplink and ending with the second. The number of neighbors and, consequently, the runtime for completely searching this neighborhood is bounded by $O((b_k^{\mathrm{u}})^2 \cdot \lceil V_k / b_k^{\mathrm{l}} \rceil)$ for $k \in \{2, 3\}$.

Three-Edge-Exchange (3EE): After the use of 2EE a reduced form of three-edge-exchange is applied to all layer 2 and layer 3 paths. This reduced form, as presented in [9], only deals with the cases that are not covered by 2EE, i.e., when indeed three edges are replaced by three new edges. This limitation and the fact that the number of neighbors lies in $O((b_k^{\mathrm{u}})^3 \cdot \lceil V_k / b_k^{\mathrm{l}} \rceil)$, $k \in \{2, 3\}$, makes searching 3EE relatively fast in relation to the standard three-edge-exchange for the TSP. Like 2EE this operator is applied to all edges of a path including the uplinks, and only feasible solutions are considered.

Split-Rings (SR): This operator splits one path into two subpaths and relinks the end of the first and the beginning of the second subpath back to the preceding layer. Care must be taken as some layer 2 nodes along a split path might serve as hub nodes for layer 3 paths. If the split is performed between two hub nodes of an attached layer 3 path, this layer 3 path would not be connected to the same layer 2 path anymore. Each layer 3 path affected in this way is therefore relinked by determinig its cheapest feasible pair of hub node connections. Additionally, the splitting point must be chosen so that both subpaths exceed b_k^{l}, $k \in \{2, 3\}$.

Two-Node-Exchange (TNE): As the name indicates this operator considers all solutions in which two nodes from different paths on the same layer are exchanged. If an exchange on layer 2 affects a hub node, then the attached layer 3 paths must be relinked as described above. This is achieved be relinking either within the old or within the new path of the exchanged hub node.

One-Node-Move (ONM): This operator moves a single node from one path to another, i.e., the node is removed from its old path and inserted in another path from the same layer when the corresponding path length bounds b_k^{l} and b_k^{u} are satisfied. Again, if a layer 2 hub node is moved then relinking is necessary for the attached layer 3 paths. All nodes as well as all feasible insertion points are considered.

Append-Rings (AR): This operator represents the complement to SR. It connects two paths on the same layer. The length of the new path must not exceed the upper bound b_k^{u}. For layer 2 the existing uplinks can be used but relinking of layer 3 paths might be necessary, when two layer 3 paths are merged, which are connected to different layer 2 paths.

Change-Uplinks (CU): The CU operator optimizes the uplinks for all layer 2 and layer 3 paths. For each path, the overall cheapest pair of hubs is determined, taking care that the two hub nodes must be different and the hub nodes of each layer 3 must appear in the same layer 2 path.

Merge-Rings (MR): This operator provides an additional opportunity to merge short paths into a new one. While AR only considers their appendance, MR tries to insert one path between any two nodes of another path from the same layer, providing the path length limit is not exceeded.

We implemented the VND with its eight neighborhood structures with three different step functions, namely classical next and best improvement and, additionally, we used a function that applies a move immediately, when an improvement

was found (like next improvement) but then (in contrast to next improvement) stick with this operator until no further improvement can be found within this neighborhood.

4.3 Variable Neighborhood Search (VNS)

A general VNS, as used for this paper, is a stochastic algorithm that combines VND as local search procedure with an outer search mechanism performing random moves in typically larger neighborhoods in order to escape the local optima of the VND. The outer random moves are also called *shaking* [8].

In preliminary tests we searched for the best combination of shaking neighborhood structures. It appeared that only slight changes in the solution lead to a better and smooth improvement during the VNS.

Larger moves perturbated the solution in a way so that the VND could not steadily improve this solution. We finally ended up with the following four shaking operators based on TNE and ONM: (1) exchange two random nodes between layer 3 paths, (2) exchange two random nodes between layer 2 paths, (3) move one random layer 3 node to another path, and (4) move one random layer 2 node to another path. In all cases, only moves respecting all the constraints are performed.

Since, TNE and ONM are used later within the VND, the previous operators (2EE, 3EE, SR) typically already found improvements for the affected paths so that TNE or ONM will not simply undo the changes performed within the shaking.

4.4 Greedy Randomized Adaptive Search Procedure (GRASP)

GRASP [10] marks another metaheuristic approach to avoid getting trapped in local optima. A construction heuristic is randomized in order to create diverse initial solutions, and each of which is successively optimized by local search. The randomization of a construction heuristic is usually performed by turning from a pure greedy selection of the next component by which a partial solution is extended towards using a *Restricted Candidate List* (RCL) of promising candidate components and choosing from it at random.

For our purpose we randomized the construction heuristic from Section 4.1. In each iteration, when a new edge is added to a path from the current node i to the next node j, this randomized heuristic creates an RCL that contains the r-nearest neighbors of i, with r being a strategy parameter. Node j is then chosen randomly from the RCL, based on a uniform distribution. For r, i.e., the length of the RCL, we set $b_k^{\mathrm{u}}/2, k \in \{2, 3\}$. However, we still choose for each path the best uplinks deterministically. Due to the fact that the choice of the uplinks is a local decision with respect to each path we know that in the end the CU neighborhood structure will find the best uplinks for all paths. Therefore, a randomized choice of the uplinks would not meaningfully increase our search space but cause needless effort. After the randomized heuristic created an initial solution, it is improved by the previously described VND. This procedure is repeated until a given time-limit is reached.

Table 1. The underlying TSPLIB instances of the test set with the number of generated test cases, upper bounds, time limits and the average objective values for initial and final solutions and corresponding standard deviations obtained from VNS and GRASP (30 runs per test case).

				VNS				GRASP			
TSPLIB	#	b_k^u	t[s]	init	dev	final	dev	init	dev	final	dev
ulysses22	1	5-7	150	166.82	0.00	128.28	1.70	166.82	0.00	129.56	0.00
att48	4	5-7	150	76560.30	4406.66	61721.55	1053.39	82084.04	4146.29	63527.21	2120.45
eil51	4	5-7	150	1043.27	36.85	756.75	21.60	1075.17	50.44	779.55	36.27
berlin52	4	5-7	150	19078.36	1233.71	13709.26	428.23	19800.12	939.20	14309.64	769.90
eil76	12	5-12	150	1227.28	59.19	933.07	40.52	1462.40	130.43	971.26	37.23
gr96	18	5-12	300	1220.23	76.19	964.20	43.04	1514.36	172.12	1045.29	43.34
kroA100	18	5-12	300	59707.86	5263.22	41742.56	1584.33	74798.61	6745.93	46171.74	2974.85
kroB100	18	5-12	300	59259.01	6148.42	41796.04	1644.53	73918.06	8229.28	46204.49	2656.25
bier127	12	8-15	300	263995.65	8274.02	209636.01	3386.62	385901.48	24744.77	228707.75	5359.88
ch150	18	8-15	300	15589.56	575.06	12286.63	440.28	24454.02	1784.15	13988.46	436.43
kroA200	18	8-15	300	73801.91	3111.26	55314.71	1371.25	113770.39	8222.59	65670.67	1904.39
kroB200	18	8-15	300	75040.20	3855.24	58406.59	1665.32	118601.24	9128.76	66267.41	2340.33
gr229	12	12-20	600	3858.00	166.33	3008.36	59.19	7559.52	619.88	3588.33	141.86
pr299	12	12-20	600	116878.18	4526.67	96944.53	1507.31	246754.55	19372.46	119237.01	4592.66
lin318	18	12-20	600	100782.81	3423.72	83170.33	1789.88	204233.35	15512.07	104900.96	4322.02
gr431	18	12-20	900	4544.14	163.13	3651.96	105.69	9230.29	829.13	4525.99	258.50
pr439	18	12-20	900	267944.92	11269.23	222507.49	10831.91	560935.73	46642.47	287415.51	17304.18

5 Experimental Results

For testing purposes we created a benchmark instance set based on TSPLIB[1]. We used 17 TSPLIB instances and performed a k-means clustering to determine the layer 1 and layer 2 nodes. By varying the number of nodes in the layers we derived 42 instances. All graphs are complete, with the exception that edges between layer 1 and layer 3 were removed as they cannot appear in feasible solutions.

Moreover, we defined combinations of useful upper bounds for the path lengths depending on the number of nodes in the graph, which resulted in 223 overall test cases. As a lower bound we assumed a minimum length of two for all paths.

For each of the test cases we performed 30 runs executed on a single core of an Intel Xeon (Nehalem) Quadcore CPU with 2.53 GHz and 3GB of RAM. As stopping criterion we used the same CPU time limits for both VNS and GRASP as indicated in Table 1. We always applied next improvement as step function, which provided the best results according to preliminary tests. These tests also indicated that creating initial solutions with more restricted path lengths of up to $b_k^u - 2$, $k \in \{2, 3\}$, only instead of b_k^u yields slightly worse solutions but with a significantly higher improvement potential. Consequently, we followed this strategy in our VNS and GRASP. In the following section we describe our results based on this test set.

5.1 Test Results

Results are summarized in Table 1. The columns have the following meaning: *TSPLIB* indicates the underlying instances our derived test cases are based on

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

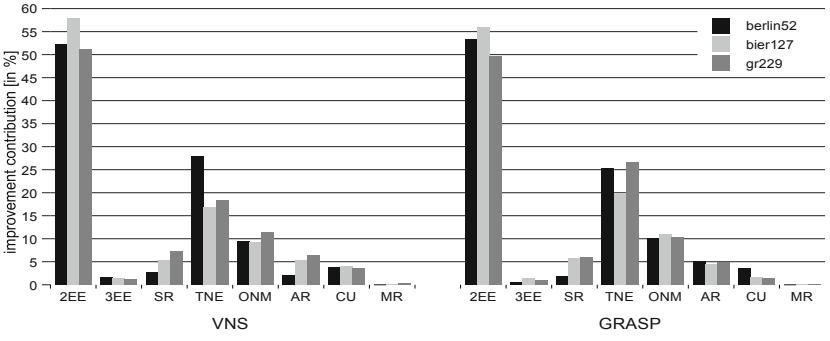


Fig. 2. Relative number of total improvements achieved by the individual neighborhood structures within VND

(the number at the end of the name denotes the number of nodes in the graph); # denotes the numbers of different test cases for each TSPLIB instance; b_k^{u} lists the range of used upper bounds for layer 2 and 3; t describes the CPU-time limits in seconds; for the *VNS* and *GRASP* sections *init* lists the average objective values of the initial solutions together with their standard deviations labeled by *dev*, while *final* denotes the average objective values of the final solutions found in each run, again together with corresponding standard deviations *dev*.

Since, in GRASP initial solutions are generated by the randomized construction heuristic, the significantly worse initial objective values and higher standard deviations are natural. As can be seen easily, the successive VND could improve these initial solutions dramatically, also leading to substantially decreased standard deviation.

Concerning VNS, it can be easily seen that it clearly outperforms GRASP. In fact, average final objective values from the VNS are always smaller than those from GRASP. Student t-tests indicated the statistical significances of these differences with error probabilities smaller than 1%. Since, there is no opportunity for GRASP to escape local optima during local search, the GRASP approach cannot easily break up the path structure of the initial solution as is done by the shaking in VNS. On the other hand, it appears that the randomized construction heuristic cannot easily provide promising path structures from the beginning.

In VND, all eight neighborhood structures contribute to the overall success, although the impacts vary. Figure 2 shows for three exemplary cases, which reflect the typical behavior well, how many times the application of each neighborhood structure led to an improved solution in relation to all improvements achieved. The test settings were for berlin52 $|V_1| = 4$, $|V_2| = 10$, $b_2^{\mathrm{u}} = 5$, $b_3^{\mathrm{u}} = 7$; for bier127 $|V_1| = 10$, $|V_2| = 40$, $b_2^{\mathrm{u}} = 12$, $b_3^{\mathrm{u}} = 15$; for gr229 $|V_1| = 12$, $|V_2| = 80$, $b_2^{\mathrm{u}} = 17$, $b_3^{\mathrm{u}} = 20$. One can further see here that the success of neighborhood structures only loosely depends on the upper bounds for the path lengths. The highest improvement was achieved by 2EE, which is also applied first, while 3EE had only a minor impact. TNE proves to be an important operator, followed by ONM. MR was rarely successful.

6 Conclusions and Future Work

We introduced the Three-Layer Hierarchical Ring Network Design Problem and presented a VNS and GRASP for solving it heuristically. Both strategies use a common construction strategy as well as a VND for local improvement. The VND explores eight tailored neighborhood structures, which have been shown to augment each other well. In our experiments, the VNS clearly outperformed GRASP. In the future we will investigate alternatives for the construction heuristic, e.g., based on the prominent savings heuristic from vehicle routing problems, as well as study further potentially more powerful neighborhood structures for VND/VNS. In particular, we expect further improvements by considering larger neighborhood search concepts such as cyclic exchanges over more than two paths, as well as hybrid techniques involving mixed integer linear programming for solving reasonably sized subproblems. An important practical necessity is to test the approaches also on more realistic sparse graphs.

References

1. Concorde TSP Solver, www.tsp.gatech.edu/concorde.html, (accessed: February 02, 2012)
2. Aringhieri, R., Dell' Amico, M.: Comparing Metaheuristic Algorithms for Sonet Network Design Problems. Journal of Heuristics 11, 35–57 (2005)
3. Balakrishnan, A., Magnanti, T.L., Mirchandani, P.: The Multi-level Network Design Problem. Tech. rep., Massachusetts Institute of Technology, Cambridge (1991)
4. Baldacci, R., Dell'Amico, M., Gonzalez, J.S.: The Capacitated m-Ring-Star Problem. Operations Research 55(6), 1147–1162 (2007)
5. Current, J.R., ReVelle, C.S., Cohon, J.L.: The hierarchical network design problem. European Journal of Operational Research 27(1), 57–66 (1986)
6. Dreyfus, S.E., Wagner, R.A.: The Steiner Problem in Graphs. Networks 1, 195–207 (1972)
7. Gendreau, M., Potvin, J.Y. (eds.): Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146. Springer (2010)
8. Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M.: Variable Neighborhood Search. In: Gendreau and Potvin [7], pp. 61–86
9. Potvin, J.Y., Rousseau, J.M.: An Exchange Heuristic for the for the Routing Problems with Time Windows. Journal of the Operational Research Society 46, 1433–1446 (1995)
10. Resende, M.G., Ribeiro, C.C.: Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications. In: Gendreau and Potvin [7], pp. 283–320
11. Schwengerer, M., Pirkwieser, S., Raidl, G.R.: A Variable Neighborhood Search Approach for the Two-Echelon Location-Routing Problem. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 13–24. Springer, Heidelberg (2012)
12. Thomadsen, T., Stidsen, T.: Hierarchical Ring Network Design Using Branch-and-Price. Telecommunication Systems 29(1), 61–76 (2005)
13. Trampont, M., Destré, C., Faye, A.: Solving a hierarchical network design problem with two stabilized column generation approaches. In: International Network Optimization Conference 2009, Pisa, Italy (2009)

Extracting Key Gene Regulatory Dynamics for the Direct Control of Mechanical Systems

Jean Krohn and Denise Gorse

Department of Computer Science, UCL (University College London),
Gower Street, London WC1E 6BT, UK
{j.krohn,d.gorse}@cs.ucl.ac.uk

Abstract. Evolution produces gene regulatory networks (GRNs) able to control cells. With this inspiration we evolve artificial GRN (AGRN) genomes for the reinforcement learning control of mechanical systems with unknown dynamics, a problem domain similar in its sparse feedback to that of controlling a biological cell. From the fractal GRN (FGRN), a successful but complex GRN model, we obtain the Input-Merge-Regulate-Output (IMRO) abstraction for GRN-based controllers, in which the FGRN's complex fractal operations are replaced by simpler ones. Computational experiments on reinforcement learning problems show significant improvements from the use of this simplified approach. We also present the first evolutionary solution to a hardened version of the acrobot problem, which previous evolutionary methods have failed on.

Keywords: Gene regulatory network, IMRO, FGRN, genetic algorithm, ALPS, control, reinforcement learning, pole balancing, acrobot.

1 Introduction

Gene regulatory networks (GRNs) act as controllers in situations as different as single cell bacteria and multi-cell organisms, with orders of magnitude of variation in size. GRNs are a product of evolution, optimised for controlling their host cell in a myriad ways, depending on context (single-cell vs multi-cell organism, during development, etc).

However it is generally desirable to avoid unnecessary complexity when developing systems inspired by nature and hence we will seek here to extract the key dynamics of the FGRN model, currently the most successful AGRN model for control, to improve its control capabilities. In parallel with nature we use artificial evolution, in the form of a genetic algorithm (GA), to evolve AGRNs for the control of mechanical systems.

This paper focuses on mechanical systems reinforcement learning problems, in which the system's dynamics are completely unknown and the reinforcement feedback is limited. Unknown dynamics is important for real world applications (for example coal furnace combustion control [5]), and in addition by minimising domain knowledge we are ensuring the wide applicability of our method. This work additionally represents the first solution of the double pole and acrobot problems with an AGRN system.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 468–477, 2012.

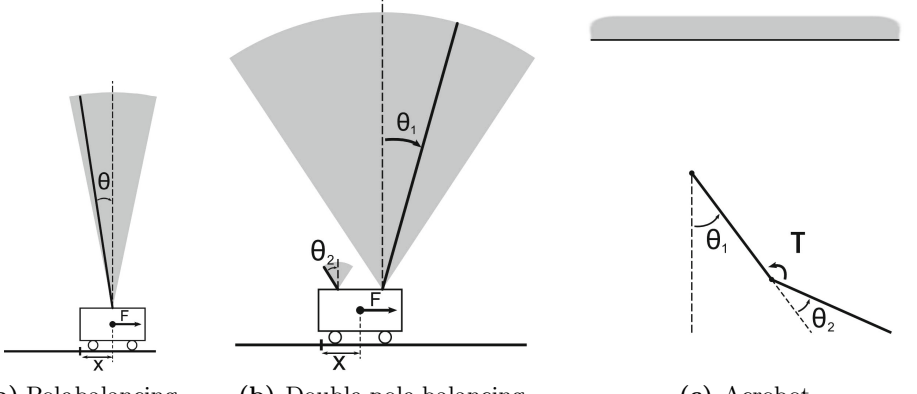
2 Related Work on Artificial GRNs

Surprisingly, given the role of natural GRNs in biological control, AGRN applications were initially focused only on development. But recently AGRN models have known increased use for control: Nicolau et al[12], and Lopes and Costa[11], have applied GRNs based on a binary genome to single pole balancing, while Joachimczak and Wróbel[8] used a GRN based on a linear genome for evolving simple foraging behaviours.

The FGRN system, introduced by Bentley[2], is a complex evolutionary model of a GRN in which proteins are bitmaps generated from the Mandelbrot set fractal. The FGRN was originally devised as a developmental system with applications such as pattern generation, or an algorithm producing increasingly precise estimations of π[9], though it also had clear potential for control purposes. There have been a number of previous FGRN control applications, evolving behaviours such as wall-following[1], grid-world box-pushing[17], and robotic locomotion[16]. In particular Krohn and Gorse[10] have applied the FGRN system to multiple versions of the single pole balancing problem, a starting point for the more ambitious work to be presented here.

3 Problem Domain: Reinforcement Learning

Reinforcement learning consists of discovering what actions to take for any given environmental state in order to maximise a scalar reward[15]. In this paper the focus will be on problems in which the full state of the environment is given, but in which, as discussed above, the environmental dynamics are unknown. Figure 1 displays the problems considered in this work.



(a) Pole balancing **(b)** Double pole balancing **(c)** Acrobot

Fig. 1. Reinforcement learning problems considered in this work

3.1 Pole Balancing

Pole balancing is a well-known, well-studied control problem that has been used as a benchmark for the design and test of many controllers[6]. Consequently, standard equations of motion and constants have arisen; these will be used in the current work and are described in ref [6].

The single pole version, shown in Figure 1a, is usually (and here) defined to be the problem of keeping the angular position θ of the 1.0m tall hinged pole within 12° of vertical, and the distance x of the cart on which it is mounted within 2.4m of the centre of the track, using only 'bang-bang' control (a force F of $\pm$10N being applied to the cart at each time step). The system state information given to the controller consists of x, $\dot{x}$, θ, $\dot{\theta}$.

The double pole version, shown in Figure 1b, consists of simultaneously balancing two poles of different size, with different starting angles, on the cart. The poles are respectively 1.0m and 0.1m tall, and the acceptable range for pole angles θ_1 and θ_2 is here within 36° of vertical. In this case the system state is composed of x, $\dot{x}$, θ_1, $\dot{\theta}_1$, θ_2, $\dot{\theta}_2$. Most other studies involving double pole balancing use as integration method two-step fourth order Runge-Kutta, and for consistency this will be used throughout the current work.

Gomez et al. have produced an extensive comparison of the performance of machine learning methods on the single and double pole balancing problems with or without velocities included in the inputs[6]. Controllers developed using the pole balancing problem have also been used in a variety of real-world control applications[5].

3.2 Acrobot

The acrobot, shown in Figure 1c, is a two-link underactuated robot[13]; it is roughly analogous to a gymnast hanging from a bar and only able to act by bending at the hips. The acrobot has been extensively studied both as a control and machine learning problem.

Unlike the pole balancing problem the acrobot problem definition varies significantly from one study to the next. However acrobot goals can be put into two broad categories: swing-up and handstand. Swing-up consists of generating actions such that the acrobot's tip (the gymnast's feet) reaches a one link height above the bar in the shortest possible amount of simulated time. Handstand is the harder task of swinging up the acrobot and then keeping both links vertically balanced; *all* solutions to the acrobot handstand problem have so far included pre-existing knowledge of the problem, e.g. the equations of motion, the desired energy level of the goal position, or the coordinates of the target position[3]. Solutions to the swing-up problem have frequently also involved pre-existing domain knowledge, though Sutton[14] successfully applied a combination of SARSA with coarse input coding to the 5Hz swing up problem, with bang-zero-bang.

More recently, and most significantly for the current work, da Motta Salles Barreto and Anderson[4] have introduced a harder version of the acrobot swing-up problem by multiplying by four the frequency of control actions (using a 20Hz

rather than 5Hz control frequency), reporting successful results with a SARSA-based method and a policy iteration algorithm but being unable to obtain any viable solution using several evolutionary methods. In contrast this paper will show that by extracting key operational features from the FGRN model, and by using for inputs the continuous state representation $\sin\theta_1$, $\cos\theta_1$, $\sin\theta_2$, $\cos\theta_2$, $\dot{\theta_1}$, and $\dot{\theta_2}$, it is indeed possible to address this more challenging version of the problem using an evolutionary method.

4 System Description

4.1 Input-Merge-Regulate-Output (IMRO) System

The IMRO system is an abstraction of the GRN model that underpins the FGRN. An IMRO genome is a set of genes with one of three possible types (input, regulatory, or output). An IMRO controller is the combination of a genome and a *merging* module which, similarly to a biological cell, provides the environment for the 'execution' of the genome. The merging module takes in proteins and merges them into a cell state that changes with the addition of new proteins. The cell state is an array of real values of length N.

An *input* gene takes in a scalar input and produces a protein output; it is equivalent to a combination of the FGRN's environmental and receptor genes. Both *regulatory* and *output* genes take in the cell state, outputting proteins in the case of the former, and a scalar in the case of the latter. The outputs of the genes are functions only of their latest input, whereas the merging module stores past proteins until they have decayed by the mechanism to be described below.

One control iteration of the IMRO controller (in which the controller receives input and gives output) consists of the following steps: (i) the existing proteins are decayed; (ii) for each scalar input a corresponding input protein is generated and added to the existing proteins in the merging module; (iii) these proteins are combined into a new cell state by the merging module; (iv) the cell state determines the activation of the regulatory genes, which output proteins to the merging module, and also the activation of the output genes, which produce the controller's scalar outputs.

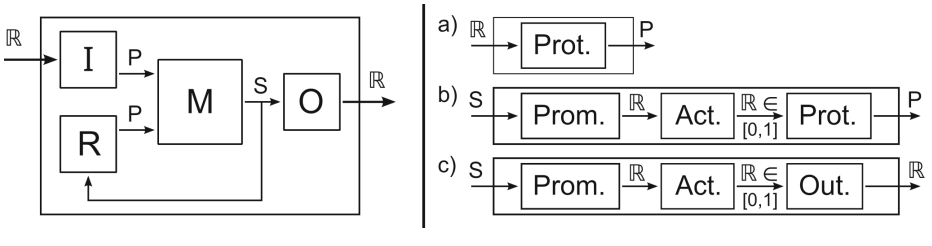


Fig. 2. The IMRO system. Left, the data flow of a controller. Right, the internal details of the a) **I**nput, b) **R**egulatory, and c) **O**utput genes. Key: P = protein, S = cell state, ℝ = real scalar

Proteins and Cell State. A protein is composed of an array of integers L of length N, a lifespan τ and a real value v. L is an array of levels, determining how prominently the protein will feature in the cell state; τ is the protein's time to live; and v determines how the protein will influence, through the cell state, the activation of regulatory and output genes. Proteins are decayed by decreasing τ by one; when $\tau = 0$ the protein is deleted.

The cell state array is generated by taking, for each i in N, the value v of the protein with the highest level L_i for that index (or the average of the v values, if several proteins have the same maximum L_i); if for the index i there is no existing protein value L_i superior to a fixed threshold set to 0, the corresponding value in the cell state is set to 0. This allows the evolution of proteins which only influence part of the cell state.

4.2 Gene Components

As detailed in Figure 2, the genes are composed of combinations of four components: promoter, activation, protein-output, and scalar-output. All component parameters are subject to evolution.

Gene Promoter. The role of a natural gene's promoter section is to regulate the activation of the gene based on the presence/absence of certain proteins or combinations of proteins. The IMRO gene promoter accomplishes this regulatory role by masking away part of the merged protein cell state, and then producing a matching score that is used further on to determine the activation of the gene.

In detail, the IMRO promoter consists of a pair of evolvable arrays of the same size N as the cell state: a boolean vector M acting as a mask, and a real vector W providing weights for the corresponding values in the cell state. Formally, the matching score $m_{i,t}$ of the promoter of gene i at time step t of a given simulation (e.g. a pole balancing run) is $m_{i,t} = \sum_{j=1}^{N} M_{i,j} W_{i,j} S_{j,t}$, where $M_{i,j} \in \{0, 1\}$ is the jth element of the promoter mask vector of gene i; $W_{i,j} \in \mathbb{R}$ is the jth element of the promoter weight vector of gene i; and $S_{j,t} \in \mathbb{R}$ is the jth element of the cell state at time t.

Gene Activation. The gene activation function of IMRO regulatory and output genes is similar to the FGRN's activation function, but removes the need for arbitrary constants. The activation function of gene i is defined by its scale α_i and its threshold $\theta_i \in [-1, 1]$. For gene i at time t, the activation $a_{i,t} \in [0, 1]$ is given by

$$a_{i,t} = \begin{cases} \frac{\max(v_{i,t}, \theta_i) - \theta_i}{1 - \theta_i} & \text{if } \theta_i > 0 \\ \frac{\min(v_{i,t}, |\theta_i|)}{|\theta_i|} & \text{if } \theta_i < 0 \end{cases}, \text{where } v_{i,t} = \frac{\tanh(\alpha_i m_{i,t}) + 1}{2}$$

This allows for a large variety in the direction and scale of the activation function, while preserving the general shape of its natural equivalent : a 0 or 1 plateau, followed or preceded by a smooth curve to/from the other end of the $[0, 1]$ range. This function also preserves both the digital aspect of natural gene activation (a gene can be activated or not), and the analog aspect (once activated, a variable amount of protein can be produced, depending on the activation level).

Protein Output. When in a regulatory gene, the protein output component generates a protein on activation (when the activation value is non null). The component defines the protein's L level array, as well as its initial time-to-live τ. The protein's value v is determined from the combination of a scaling factor β and the activation value. For gene i at time t, the protein value is $v_{i,t} = \beta_i a_{i,t}$, and similarly when in an input gene, except an external scalar input then replaces the activation value. In randomly initialised genomes, the protein output components of input genes are initialised with a τ of one, and only have one positive value amongst the levels, to avoid a flooding of the cell state.

Scalar Output. The scalar output component determines the return value of an output gene. If a boolean value is desired, the output of gene i at time t is $o_{i,t} = 0$ iff $a_{i,t} = 0$, otherwise 1. If a real value is desired, the component has a scale parameter β, and a threshold parameter T; the output of gene i at time t is then $o_{i,t} = \max(a_{i,t}, 0)$ iif $T \geq 0$, and $|T| - \min(a_{i,t}, |T|)$ otherwise.

5 Experiments

In this section we first give some relevant parameter settings and experimental details, before presenting some preliminary experiments along with their results, and finally detailing the results of the pole balancing and acrobot experiments.

5.1 Parameter Settings and Experimental Details

A maximum of 10,000 genomes are evaluated per run for the preliminary experiments and the pole balancing. For the acrobot, following ref [4], a maximum of 3,000 genomes is evaluated per run. All experiments are run 50 times.[1]

Genetic Algorithm. IMRO and FGRN genomes are evolved using the ALPS genetic algorithm[7] with a layer size of 25 and an age gap of 10. Tournament selection is used in each layer, with a tournament size of 4 and with elitism set to 3. Parents are selected from the top 40% of each layer, except in one percent of cases, where a parent is selected randomly. The gene component mutation rate is 0.1, and uniform crossover is always applied. ALPS was found to increase the reliability with which successful FGRN controllers were found[10].

[1] The source code for all the experiments and systems described in this paper is available at `http://github.com/susano/ppsn2012`

FGRN. The FGRN genomes evolved are composed of four regulatory genes, one receptor gene, one behavioural (output) gene, and as many environmental genes as there are inputs. The zero centered input-mapping[10] is used.

IMRO. The IMRO genomes evolved are composed of two regulatory genes, one output gene, and as many input genes as required by the problem. The size of the cell state N is set to eight. The allocation of more regulatory genes to FGRN genomes than to IMRO genomes followed preliminary experiments in which FGRN performances were poorer with fewer than four regulatory genes.

5.2 Preliminary Experiments

The initial test[2] of the FGRN model's developmental capabilities was to attempt to evolve genomes able to produce specific activation patterns (see Figure 3), no input was given. The fitness of a genome was the number of matches between its activation output and the pattern. The ability to generate a variety of activation patterns, independently of any input, can allow the exploration of otherwise closed regions of the space of possible controllers, and we therefore applied both FGRN and IMRO genomes to this task.

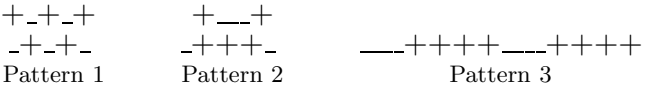


Fig. 3. Test activation patterns from ref [2]. Patterns 1 and 2 require two separate output genes per genome

Table 1. The percentage of successfully generated patterns, and the mean number of evaluations required to success (standard deviation in parenthesis).

	FGRN		IMRO	
Pattern 1	100%	810(894)	100%	275(244)
Pattern 2	100%	619(562)	100%	398(364)
Pattern 3	34%	6095(2797)	100%	1794(1758)

The results are impressive: IMRO genomes can be evolved significantly faster ($p < 0.001$) to produce the desired pattern than FGRN genomes, and in the case of pattern 3, much more reliably. (It should be noted that the FGRN results on pattern 3, despite being significantly worse than the IMRO results, are an improvement on Bentley's initial results for this pattern [2], where an additional guidance component needed to be added to the fitness to successfully evolve this pattern. We attribute this to the use here of the ALPS genetic algorithm, and to an improvement in the FGRN settings we use[10])

5.3 Pole Balancing and Acrobot Experiments

The setup of the pole balancing and acrobot problems is detailed in Section 3. The controllers are run on the pole balancing problems for 100,000 simulated timesteps (≈ 30 minutes). For the acrobot, each controller is run for a maximum of 4,000 timesteps; this was necessary instead of the 1,000 timesteps to allow the genetic algorithm to find initial solutions from which to start improving. The fitness for pole balancing is the number of timesteps before a pole falls down. For the acrobot it is the number of timesteps until swing-up, inversed.

Pole balancing. The results are detailed in Table 2. Both FGRN and IMRO genomes were able to evolve successful controllers at every run for the single pole balancing problem, but only IMRO genomes were able to evolve the ability to solve the double-pole balancing problem, the most successful FGRN controller only balancing the poles for 172 timesteps (≈ 3 seconds) out of 100,000.

Table 2. Number of failures/evaluations before a successful controller is found. Key: SD = Standard Deviation

	FGRN			IMRO		
	Mean(SD)	Best	Worst	Mean(SD)	Best	Worst
Single Pole	729(787)	50	5129	480(356)	33	1985
Double Pole	-	-	-	2200(1486)	487	8155

Acrobot. Table 3 details the results of the IMRO and FGRN systems on the acrobot, as well as those of the SARSA-RGD system, an online learning method, and of LSPI, a policy iteration method, on the same problem. The IMRO system performed significantly better than both the FGRN system and LSPI ($p < 0.001$), finding on average significantly shorter trajectories. But it performed worse than SARSA-RGD, though SARSA-RGD was less reliable, failing in some of the runs to find any swing-up trajectory. Figure 4 shows the trajectory of an acrobot controlled by the IMRO system.

Table 3. Length of the shortest trajectory found to acrobot swing-up, sorted by shortest average trajectory. The results for SARSA-RGD and LSPI are taken from ref [4].

	Mean(SD)	Best	Worst
SARSA-RGD	276.56(106.62)	238	-
IMRO	307.68(40.42)	266	500
LSPI	335.90(12.11)	315	343
FGRN	357.26(69.92)	257	588

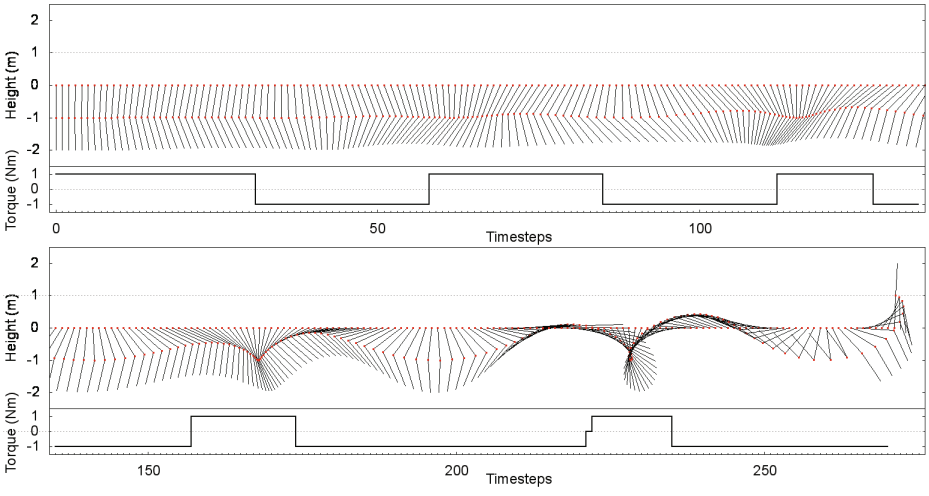


Fig. 4. An example acrobot swing-up trajectory produced by the IMRO system. Top, the position of the acrobot at each time step. Bottom, the force applied at each timestep. Long periods of the same activation, and limited use of the null force action, are typical of efficient swing-up solutions.

6 Discussion

This paper has introduced IMRO, a simplified abstraction of the GRN model underpinning the successful but complex FGRN system. These simplifications, already desirable in themselves, resulted in greatly improved performance on control tasks of a widely different nature: while the pole balancing is a stabilisation problem, the acrobot is the exact opposite, requiring the controller to destabilise the system until it reaches a remote region of the state-space.

The performance of the FGRN system in the same experiments, and particularly the combined failure on the double pole balancing problem and in the generation of pattern 3, and its limited success on the acrobot, lead us to believe the FGRN system to be more suitable for control problems not requiring very precise control sequences, but having a large variety of possible complex control strategies. This might find its root in the original developmental nature of the FGRN system, where the complexity of the system might be more useful.

Future work with the IMRO system will focus initially on finding harder control problems to which it can be applied. If it proves necessary to make changes to the system this will be facilitated by its modularity and the clearly defined interfaces between its components. However we do not seek complexity for its own sake, but to abstract from biology only those elements that prove useful in problem solving, which may indirectly also throw light on why nature's solutions to problems are frequently so effective.

References

1. Bentley, P.J.: Evolving Fractal Gene Regulatory Networks for Robot Control. In: Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., Kim, J.T. (eds.) ECAL 2003. LNCS (LNAI), vol. 2801, pp. 753–762. Springer, Heidelberg (2003)
2. Bentley, P.J.: Fractal Proteins. Genetic Programming and Evolvable Machines Journal 5, 71–101 (2004)
3. Boone, G.: Efficient reinforcement learning: Model-based acrobot control. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, pp. 229–234. IEEE (1997)
4. da Motta Salles Barreto, A., Anderson, C.W.: Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. Artificial Intelligence 172(4-5), 454–482 (2008)
5. Funkquist, J., Stephan, V., Schaffernicht, E., Rosner, C., Berg, M.: SOFCOM-Self-optimising strategy for control of the combustion process. VGB PowerTech, 49 (2011)
6. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated Neural Evolution through Cooperatively Coevolved Synapses. The Journal of Machine Learning Research 9, 937–965 (2008)
7. Hornby, G.S.: ALPS: The Age-Layered Population Structure for Reducing the Problem of Premature Convergence. In: GECCO 2006, pp. 815–822. ACM, New York (2006)
8. Joachimczak, M., Wróbel, B.: Evolving gene regulatory networks for real time control of foraging behaviours. In: Artificial Life XII, pp. 348–355 (2010)
9. Krohn, J., Bentley, P.J., Shayani, H.: The Challenge of Irrationality: Fractal Protein Recipes for PI. In: GECCO 2009, Montreal, Canada (July 2009)
10. Krohn, J., Gorse, D.: Fractal Gene Regulatory Networks for Control of Nonlinear Systems. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 209–218. Springer, Heidelberg (2010)
11. Lopes, R.L., Costa, E.: ReNCoDe: A Regulatory Network Computational Device. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 142–153. Springer, Heidelberg (2011)
12. Nicolau, M., Schoenauer, M., Banzhaf, W.: Evolving Genes to Balance a Pole. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 196–207. Springer, Heidelberg (2010)
13. Spong, M.W.: The swing up control problem for the acrobot. IEEE Control Systems Magazine 15(1), 49–55 (1995)
14. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Advances in Neural Information Processing Systems, pp. 1038–1044 (1996)
15. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 28. Cambridge Univ. Press (1998)
16. Zahadat, P., Christensen, D.J., Schultz, U.P., Katebi, S., Stoy, K.: Fractal Gene Regulatory Networks for Robust Locomotion Control of Modular Robots. In: Doncieux, S., Girard, B., Guillot, A., Hallam, J., Meyer, J.-A., Mouret, J.-B. (eds.) SAB 2010. LNCS, vol. 6226, pp. 544–554. Springer, Heidelberg (2010)
17. Zahadat, P., Katebi, S.D.: Tartarus And Fractal Gene Regulatory Networks With Inputs. Advances in Complex Systems (ACS) 11(06), 803–829 (2008)

An Evolutionary Optimization Approach for Bulk Material Blending Systems

Michael P. Cipold[1,2], Pradyumn Kumar Shukla[1], Claus C. Bachmann[2], Kaibin Bao[1], and Hartmut Schmeck[1]

[1] Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, D-76128, Germany
[2] J&C Bachmann GmbH, Bad Wildbad, D-75323, Germany

Abstract. Bulk material blending systems still mostly implement static and non-reactive material blending methods like the well-known Chevron stacking. The optimization potential in the existing systems which can be made available using quality analyzing methods as online X-ray fluorescence measurement is inspected in detail in this paper using a multi-objective optimization approach based on steady state evolutionary algorithms. We propose various Baldwinian and Lamarckian repair algorithms, test them on real world problem data and deliver optimized solutions which outperform the standard techniques.

Keywords: Bulk Material Blending, Multi-objective Evolutionary Algorithms, Chevron Stacking.

1 Introduction

Naturally occurring materials like minerals, coals, and ores are inherently inhomogeneous products. However, efficient processing of those products requires that the quality of the product does not vary beyond a limited range. While a constant quality cannot be assured during mining or refining the uniform quality can be achieved by measuring the quality and blending the material.

A whole range of bulk material blending systems are developed to buffer bulk material from a source, blend it and finally deliver a product with homogenized quality parameters. As quantity of bulk material is usually measured in ten thousands of tons, the implemented systems are using huge machines to process the material in so-called blending beds. Blending beds are areas of typical length up to 1000m and width up to 50m which can be used by the stacking and reclaiming machines to build and ablate stockpiles and can be organized in multiple ways [1]. In this paper, the focus lies on longitudinal blending beds that use a railed stacker (shown in Figure 1) and a railed bridge reclaimer. Figure 2 illustrates a blending bed with all elements as it is used for the model

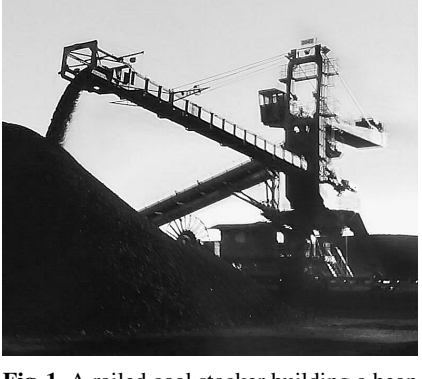


Fig. 1. A railed coal stacker building a heap

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 478–488, 2012.

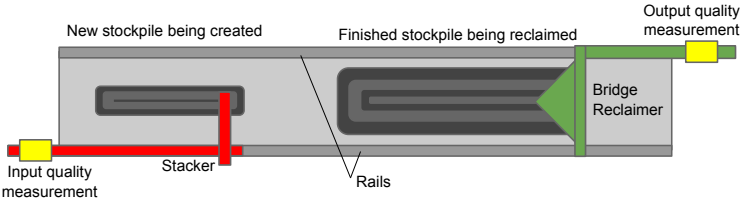


Fig. 2. Top view sketch of a blending bed with one heap being stacked and another heap being reclaimed

in this paper using simplified parameter and environment descriptions to concentrate on the main goal, i.e., to deliver a homogeneous product.

The homogenization in blending beds is usually done by stacking layers of material onto each other and reclaiming the material vertically or diagonal to the layers to get mixed material from the cross sections through the layers. As the amount of layers is set to a fixed value at the beginning of building a stockpile, in the most cases the variation of the material parameters (for the remaining part of the paper referred with the abstract description *quality*) is not homogenized in an optimal way. Most established blending systems do not really use the optimization potential what in turn results in inefficient downstream processing. As these blending systems can not be modified in general without putting in much effort and money, this paper is aiming to analyze the optimization potential with minimal intrusion to a given system by using the possibilities which inhere in such a system: spreading the material in a dynamic way according to a quality analyzing online X-ray fluorescence measurement [2].

To analyze the maximum efficiency of a blending system we require to know the full quality of the material input (hereinafter referenced as *quality input curve*) from the beginning of the optimization. In this paper we present an optimization environment which is based on this information and calculates an optimized material spread by modifying the traverse path for the stacker. The approach is also portable to blending systems with other parameter and design choices.

Many authors have analyzed and improved the blending efficiency in the field of bulk material blending. Kumral [3] describes a method to optimize a mineral blending system design before its construction to get the best performance. Using genetic algorithms and a multiple regression model they determine the best blending bed parameters simulating the stockpile with a simplified cell model. Pavloudakis and Agioutanis use a more complex stockpile simulation approach consisting of multiple layers lying on top of each other [4]. They assume a constant material flow and calculate the expected quality at the material output. Bond et. al. [1] describe methods to improve the efficiency of blending beds by dynamically modifying the volume of a stockpile and modifying the stacking speed while measuring the input quality with an online analyzer. They suggest a solution with an appropriate software to control the stacker speed dynamically during the stacking to place material with recognized quality at specific locations in the blending bed. The optimization based approach has, to the best of our knowledge, never been applied in such a system and this is the topic dealt in this paper.

This paper is divided into five sections of which this is the first. The next section describes the problem and a simulation model that is developed. Section 3 presents

individual representations and repair methodologies that are used in a steady-state multi-objective evolutionary algorithm. The optimization results are discussed in Section 4 and conclusions are presented in Section 5.

2 Problem Modeling

The described real world system consists of multiple elements which need to be mapped into a system model first. The model consists of:

- *blending bed parameters* **b** (fixed parameters such as the stockpile dimensions, maximum stacker moving speed, reclaimer angle, etc.),
- *quality input* **q** (the quality curve for each run plotted against the total amount of material),
- *traverse path* **p** (the path along which the stacker is driving during the stockpile creation process), and
- *environment parameters* **e** (material and weather conditions).

Given that the stacking system is unchangeable and that the quality input is fixed for each run, the optimization focuses on modifying the stacker driving path along the blending bed. For simplicity, the environment parameters are not considered here. Hence, values of **b** and **q** define an instance of the optimization problem. In order to apply optimization with varying input data, the system was mapped into a detailed system model. The system uses a physics simulation to calculate the behavior of particles of various size being dropped onto a stockpile. Each of these particles is assigned a quality according to the current average quality being measured (or to an earlier recorded quality curve). The simulation system calculates the full stockpile as it is created in the real world including the effects of particles slipping from the steep sides of the stockpile. After building the stockpile the reclaiming is simulated by calculating the cross section quality average in the angle the reclaimer would ablate the material. One function evaluation (heap length 300m, heap width 40m) using the detailed simulator, with one million particles takes about 20 hours on an Intel Core 2 Duo, 2.50 GHz.

As the time for stacking one stockpile usually varies between 24 and 48 hours, this near real time calculation is not useful for optimization where the quality output has to be calculated for various input paths. For the fast evaluation of a calculated solution another simplified simulation system is developed which is able to do a rough output quality calculation within 20-50 milliseconds on the same machine. The physics simulation is now replaced by a three-dimensional grid where each particle is dropped in a specific place and then it falls down until it reaches a resting particle. From this place, it falls into the direction of biggest height difference. A particle can fall in one of the eight directions (top-left, top-center, top-right, left-middle, right-middle, bottom-left, bottom-center, bottom-right) or, it can stay in its place if there is no direction with lower height to fall to. The amount of particles here was reduced to 3 particles per cubic meter to minimize the necessary amount of calculations. The results of this (simplified) simulator show a satisfying correlation with the results from the detailed simulation (see Figures 3 and 4, the Pearson's correlation ranges from 0.85 for low variance solutions and up to 0.99 for high variance solutions). Hence, this simulator can be used for a rough calculation of the optimized traverse path quality output in a short time.

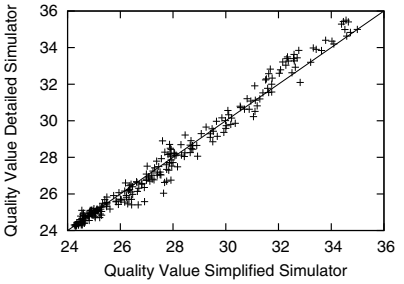


Fig. 3. Scatter plot showing high variance output quality correlation for detailed simulator against simplified simulator

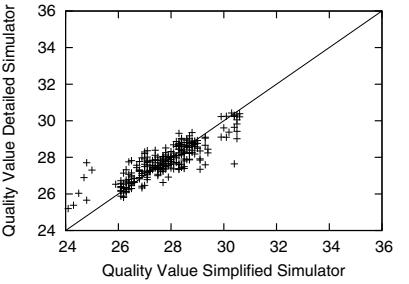


Fig. 4. Scatter plot showing low variance output quality correlation for detailed simulator against simplified simulator

Input Data. Input data was collected using an online X-ray fluorescence system delivering the material parameters of main interest in real time. This system was installed in a coal processing plant in South Africa. To gather more data for the simulation with other variation characteristics the recorded data was analyzed and replicated. The replicated data values are set in the same quality range but show a different variation timing to test the stability of the optimization system in situations with long constant quality or fast quality variation. For this, four different input quality curves with different characteristics will be used of which curve 1 is the original data from the coal processing plant. The problem corresponding to the first quality curve is termed as P1, and so on.

Objectives and Constraints. As described earlier, the traverse path $\mathbf{p}$ of the stacker has to be optimized for ideal material blending. The first objective here is the weighted variance of the quality at the reclaimer which is defined as

$$F_1(\mathbf{p}) := \frac{\tilde{n}(\mathbf{p}) \cdot \sum_{i=1}^{n} \left(w_i(\mathbf{p}) \cdot (q_i^{\text{out}}(\mathbf{p}) - \overline{q}^{\text{out}})^2 \right)}{\tilde{n}(\mathbf{p}) \cdot \sum_{i=1}^{n} w_i(\mathbf{p}) - \sum_{i=1}^{n} w_i(\mathbf{p})}, \tag{1}$$

where n is the number of cross sections, $w_i(\mathbf{p}), q_i^{\text{out}}(\mathbf{p})$ are the amount of material and average quality in cross section i, $\overline{q}^{\text{out}}$ is the average output quality (depending on the quality input $\mathbf{q}$), and $\tilde{n}(\mathbf{p})$ is the number of non-empty cross-sections. When the traverse path is changed it has a direct influence on the shape of the stockpile. This leads to the other primary objective to create a stockpile with a ridge of nearly constant height. The objective will be represented with the relative height difference defined as

$$F_2(\mathbf{p}) := \frac{h_{\max}(\mathbf{p}) - h_{\min}(\mathbf{p})}{\overline{h}(\mathbf{p})}, \tag{2}$$

where $h_{\max}(\mathbf{p}), h_{\min}(\mathbf{p})$ and $\overline{h}(\mathbf{p})$ denote the maximum, minimum, and the average stockpile heights respectively. Other objectives like having the least speed changes or driving only as fast as necessary to meet a defined threshold will not be regarded in

this paper but can be easily included in the method if required. The constraints in this optimization problem are:

- the stacker must traverse within the region $[0, p_{\max}]$ and
- the given speed range $[-v_{\max}, v_{\max}]$ may not be violated,

where $p_{\max}$ and $v_{\max}$ are the maximal length of the blending bed and the maximum speed of the stacker, respectively (the stacker moves along the rails in two directions).

3 Algorithms

The bi-objective problem described in the last section cannot be written in a closed mathematical form, prohibiting the use of exact algorithms (like [5]). Due to this, we used a steady-state version of NSGA-II (ssNSGAII) [6, 7]. For all the four problems, we use a population of size 100 and set the maximum number of function evaluations as 25,000 (250 generations). We use a standard real parameter SBX and polynomial mutation operator with $\eta_c = 15$ and $\eta_m = 20$, respectively [6]. The individual representation and the repairing methods are explained next.

3.1 Representation of Individuals

In order to map the complex traverse path of a stacker to an individual of the evolutionary algorithm (and vice versa) specific representations are defined. For this, the continuous traverse path is discretized and two representations are proposed.

Array of Speeds. This path representation uses an array of l floating point values v_i which describes the stacker driving speed in a specific time slot. To have a universal representation for the algorithm the value range is set to $[-1, 1]$. A value of 0 represents no movement and an absolute value of 1 represents movement with maximum speed. The direction of the movement is encoded in the signum of the value. Hence, an individual representation is $(v_1, \ldots, v_l)$ where $v_i \in [-1, 1]$ for all i. The length of a time slot t_{slot} is defined by the amount of time which is needed for all material to be stacked t_{total} divided by the length of the array l, i.e., $t_{\text{slot}} = \frac{t_{\text{total}}}{l}$.

Example 1. Individual $(0.5, -1.0, 0.5)$ represents a stacker movement to the right with half of the maximum speed, then full speed movement to the left and again half speed movement to the right. Each time slot is one third of the full stacking time. □

Array of Positions. This path representation also utilizes an array of l floating point values p_i corresponding to positions in the valid traverse path of the stacker. The value range $[0, 1]$ corresponds to the full traverse range. Hence, an individual representation is $(p_1, \ldots, p_l)$ where $p_i \in [0, 1]$ for all i. The time for moving between two positions t_{slot} is calculated similar to the time at the *array of speeds* but as we have a specific starting point for each individual the value of t_{slot} slightly increases. In this case, $t_{\text{slot}} = \frac{t_{\text{total}}}{l-1}$.

Example 2. Individual $(0.5, 1.0, 0.0)$ represents a stacker movement from the center of the valid traverse path to the right end and then to the left end with each time slot being half of the maximum time. □

3.2 Repair Mechanisms

The *array of speeds* representation was chosen in the first place to have a simple representation where every created individual can be mapped to a valid traverse path using a repair method. The *array of positions* representation was chosen with regard to be able to repair the individuals within the optimization and not only for representation. In the following paragraphs the used repair methods for the representations will be described in detail.

Array of Speeds. Calculating the traverse path for the *array of speeds* representation obviously results in most individuals being invalid because the calculated path exceeds the limits. Therefore, a simple repair mechanism was defined using a mirroring technique. As the integration of the speed over time results in the absolute position p_{abs} the position can be folded into the valid range to p_{mirrored} using equation (3):

$$p_{\text{mirrored}} = \begin{cases} p_{\text{abs}} - \lfloor p_{\text{abs}} \rfloor, & \text{if } \lfloor p_{\text{abs}} \rfloor \bmod 2 = 0, \\ 1.0 - (p_{\text{abs}} - \lfloor p_{\text{abs}} \rfloor), & \text{if } \lfloor p_{\text{abs}} \rfloor \bmod 2 = 1. \end{cases} \tag{3}$$

This way all positions exceeding the range $[0, 1]$ will be mirrored at the limits into the valid range. The repaired representation can not be mapped back to an individual because there are additional changes in the direction between two regular speed changes. Mapping back the mirroring would mean to change the amount of variables and the fixed time between two speed changes to be set.

Example 3. Let $v_{\max} = \frac{2 \cdot \textit{width of stockpile}}{t_{\text{total}}}$ and the individual be $(1.0, 0.5)$ which corresponds to the maximum speed being as much as necessary to traverse the full stockpile width twice. Furthermore we assume the start position as absolutely left. Going with the first speed 1.0 will result in the stacker being at the right end of the stockpile as $t_{\text{slot}} = 0.5 \cdot t_{\text{total}}$. In the second part the absolute position still increases with half of the maximum speed until the absolute position value of 1.5 is reached. With $\lfloor 1.5 \rfloor \bmod 2 = 1$ the mirrored position results in $p_{\text{mirrored}} = 1.0 - (1.5 - \lfloor 1.5 \rfloor) = 1.0 - (1.5 - 1.0) = 0.5$. □

Array of Positions. Due to the definition of this representation it is not possible to exceed the path limits (cf. for the *array of speeds* where it was not possible to exceed the speed limit) but it is possible that the speed $v_i = \frac{p_{i+1} - p_i}{t_{\text{slot}}}$ between two consecutive positions p_i and p_{i+1} exceeds the maximum allowable speed $v_{\max}$. For this case two repair mechanism will be used which utilize the maximum position difference $d_{\max} = t_{\text{slot}} \cdot v_{\max}$:

– *Direct Correction* iterates through the array once and limits each following position to the maximum difference to its predecessor:

$$p_{i+1} = \begin{cases} p_i - d_{\max}, & \text{if } p_i - p_{i+1} > d_{\max}, \\ p_i + d_{\max}, & \text{if } p_{i+1} - p_i > d_{\max}, \\ p_{i+1}, & \text{otherwise}, \end{cases} \qquad \text{with } i = 1, \ldots, n-1. \tag{4}$$

– *Iterative Balancing* also iterates though the array but does not only change the successor p_{i+1} to each position p_i but also the current position half of the distance d_{corr} which has to be corrected:

$$(p_i, p_{i+1}) = \begin{cases} (p_i - d_h, p_{i+1} + d_h), & \text{if } p_i - p_{i+1} > d_{\max}, \\ (p_i + d_h, p_{i+1} - d_h), & \text{if } p_{i+1} - p_i > d_{\max}, \\ (p_i, p_{i+1}), & \text{otherwise,} \end{cases} \tag{5}$$

$$\text{with} \quad i = 1, \ldots, n-1 \qquad d_{\text{corr}} = |p_i - p_{i+1}| - d_{\max} \qquad d_h = \frac{d_{\text{corr}}}{2}.$$

This iteration has to be done as long as one of the first cases is entered walking through the array. As clearly visible in Example 4, a flickering with inverse exponential decay can occur which ends after a specific threshold is met. In our implementation the floating point accuracy makes this threshold.

Example 4. Let $d_{\max} = 0.4$ and the individual be $(0.0, 0.4, 1.0)$. This means that the distance between two positions may not exceed 0.4 and the individual consists of two movements between three positions $p_1 = 0.0$, $p_2 = 0.4$ and $p_3 = 1.0$. As we perform the *iterative balancing* we compare p_1 and p_2 and get a distance not bigger than $d_{\max}$. In the next step p_2 and p_3 are evaluated resulting in a distance $p_3 - p_2 = 0.6 > d_{\max}$. To repair the individual the correction distance $d_{corr} = |0.6| - d_{\max} = 0.2$ is calculated and both positions are corrected half of the distance to the individual $(0.0, 0.5, 0.9)$. One can easily see that a new conflict between p_1 and p_2 arises in this individual which is the reason for this solution to be iterative.

Begin	$(0.0, 0.4, 1.0)$
Iteration 1	$(0.0, 0.4, 1.0), i = 1, \lvert p_1 - p_2 \rvert \leq d_{\max}$
	$(0.0, 0.5, 0.9), i = 2, d_{\text{corr}} = 0.2$
Iteration 2	$(0.05, 0.45, 0.9), i = 1, d_{\text{corr}} = 0.1$
...	

□

3.3 Implementation

The application of the repair methods described above can be done in multiple ways. In the Lamarckian method, the repaired path is not only used for the representation but also written back to the population individual for further evolution. In the Baldwinian method, the repaired path is only used for the evaluation and is not written back to the population individual. For the *array of speeds* only the Baldwinian way can be utilized (in the following referenced as $\mathbf{Bal}_s$) as the repaired representation can not be written back to the population individual. For the *array of positions* both methods will be presented and evaluated in the results section referenced as $\mathbf{Lam}_1$ and $\mathbf{Bal}_1$ using the Lamarckian respectively the Baldwinian way together with the first repair method and $\mathbf{Lam}_2$, $\mathbf{Bal}_2$ for the second repair method. The source code of all the algorithms is based on the jMetal framework[1] and is made publicly available[2] (the data files used in this paper are also available on request).

[1] http://jmetal.sourceforge.net/

[2] http://www.aifb.kit.edu/web/BlendingSystems

4 Simulation Results

We test the algorithm by limiting the number of variables $l = 30$ (speed / position changes). Moreover, the maximum speed is set to the speed needed to place 20 layers of material with the Chevron stacking method. The starting position for the results of the *array of speeds* is fixed at position 0 (at the left side of the stockpile). The input data is described in Section 2 and we test the algorithms on four problems corresponding to four quality curves. (Other quality curves were tested and we got similar results.) As the quality varies over the time and can be classified once as high and once as low the obvious intuitive solution of the optimization is to spread the particles with high quality equally over the full stockpile width and do the same for the low quality particles. The result is a constant average cross section quality for the whole stockpile. This is a very simplified description of a non-trivial problem because the particles do not arrive sorted by quality so one could easily do the spreading but the quality varies quickly over the full range and the stacker can only be moved with a defined maximum speed.

Figure 5 illustrates two solutions of the optimization problem for a given quality curve. The quality curve which is the high frequency varying curve in the background can be clearly divided into two quality classes of high quality and low quality. One can easily see the spread of the high quality material first being mainly divided to the left side of the stockpile (at $y = 0$) and at the second bigger batch of high quality material mainly on the right side of the stockpile (at $y = 1$). Hence, the optimal solutions corroborate the intuitive rule mentioned in the last paragraph. This result is found by the evolutionary algorithm and is visible in almost all the optimal solutions. Figure 6 shows the sample run of the position based Baldwinian and Lamarckian techniques in ssNSGA-II algorithm for problem P1. We see that both the methods are able to find many well-spread solutions. Figure 6 also shows a knee region in the efficient front. The knowledge of knees is valuable to a designer [6, 8]. In order to statistically evaluate our results, we run each algorithm 45 times. Various attainment surface plots [9] are shown in Figure 7. From these we see that there might be a weakly efficient front corresponding to the minimizers of F_1 and F_2 (hence multiple solutions if we only minimize one objective using a single-objective algorithm).

For this real world problem, we do not know the exact location of the efficient front and hence, we use a hypervolume indicator [6] to compare the methods. Table 1 shows

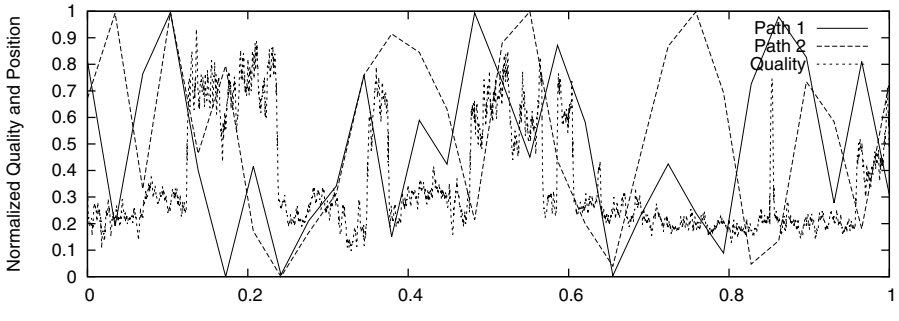


Fig. 5. Optimized solutions for the traverse path of a given input quality curve

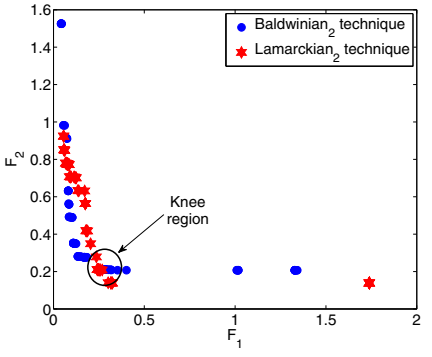


Fig. 6. Sample runs of ssNSGA-II with position based Baldwinian and Lamarckian methods for problem P1

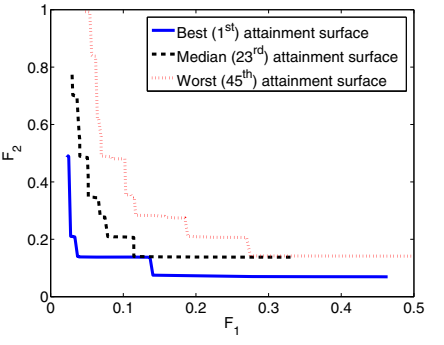


Fig. 7. Attainment surface plots of the ssNSGA-II algorithm with Lamarckian method for problem P1

Table 1. Hypervolume values for the four problems, corresponding to the reference point $(1, 1)^\top$. The hypervolume values for the Chevron method are an overestimate as we assume that the relative height difference is optimal (i.e., equal to 0). The values in dark and light grey correspond to the best and the second best algorithm (based on median values), respectively.

HV	$\mathbf{Bal}_s$	$\mathbf{Bal}_1$	$\mathbf{Bal}_2$	$\mathbf{Lam}_1$	$\mathbf{Lam}_2$
$\mathrm{HV}_{\mathrm{Chevron,P1}}$=0.85644					
$\mathrm{best}_{\mathrm{P1}}$	0.76844	0.84460	0.82855	0.81837	0.79758
$\mathrm{worst}_{\mathrm{P1}}$	0.60061	0.63456	0.67023	0.60957	0.59453
$\mathrm{median}_{\mathrm{P1}}$	0.66810	0.77881	0.76955	0.75582	0.72538
$\mathrm{IQR}_{\mathrm{P1}}$	0.07561	0.04862	0.05217	0.04637	0.06051
Chevron solution for P2 lies outside the region dominated by $(1, 1)^\top$					
$\mathrm{best}_{\mathrm{P2}}$	0.72217	0.82496	0.82939	0.85986	0.81691
$\mathrm{worst}_{\mathrm{P2}}$	0.59298	0.65282	0.62709	0.63912	0.56375
$\mathrm{median}_{\mathrm{P2}}$	0.65633	0.77066	0.74338	0.73368	0.68591
$\mathrm{IQR}_{\mathrm{P2}}$	0.05402	0.04700	0.06882	0.04597	0.09568
$\mathrm{HV}_{\mathrm{Chevron,P3}}$=0.69651					
$\mathrm{best}_{\mathrm{P3}}$	0.72205	0.82716	0.80311	0.80493	0.79385
$\mathrm{worst}_{\mathrm{P3}}$	0.57067	0.64642	0.64401	0.61633	0.59540
$\mathrm{median}_{\mathrm{P3}}$	0.62472	0.74523	0.73737	0.70537	0.71732
$\mathrm{IQR}_{\mathrm{P3}}$	0.04176	0.04757	0.05233	0.05825	0.06551
$\mathrm{HV}_{\mathrm{Chevron,P4}}$=0.16867					
$\mathrm{best}_{\mathrm{P4}}$	0.76140	0.88563	0.88149	0.86541	0.83725
$\mathrm{worst}_{\mathrm{P4}}$	0.65172	0.71243	0.73296	0.55602	0.55102
$\mathrm{median}_{\mathrm{P4}}$	0.71963	0.71253	0.73299	0.55602	0.55102
$\mathrm{IQR}_{\mathrm{P4}}$	0.04089	0.04142	0.03561	0.05149	0.05862

a hypervolume based statistical summary of the results. The reference point is chosen to be $(1, 1)^\top$ for simplicity, other values do not change the qualitative behavior. We see that the Baldwinian methods outperform the Lamarckian ones. Moreover, the *array of positions* representation usually delivered better results. The main reason for this is the higher stability of the individual, when changes are made to them by mutation or recombination. If an individual of the speed representation is modified at the begin of

the array then it affects the whole traverse path as the absolute reference position for the whole following path is shifted. This is not the case for the positions representation as changing one position only affects the traverse path in the immediate surrounding path to and from the position. For three out of four problems, we obtain a better hypervolume than the Chevron method, even if we assume that the Chevron is optimal in terms of the second objective. If we only consider the first objective, all the five algorithms produce better results than the Chevron method, for all the problems. The Chevron solution for P2 lies outside the region dominated by the reference point ($F_1 > 1.0$).

5 Conclusions and Future Work

The results presented in this paper show the optimization potential of bulk material blending beds that can be calculated and analyzed with the described methods using evolutionary algorithms. The methods presented in this paper provide a fast and flexible calculation environment with a scalable simulation system which can be adapted to various types of blending systems and parameters. We assumed in this paper that we have the full knowledge about input quality curve, however, the presented methods can be used to train and validate a Learning Classifier System for real time blending optimization. Moreover, techniques from [10, 8, 11] will be used to investigate the trade-off properties of the knee solutions in detail. Finally, attainment function [9] based statistical hypothesis and post-hoc tests shall also be conducted.

References

[1] Bond, J., Coursaux, R., Worthington, R.: Blending systems and control technologies for cement raw materials. IEEE Industry Applications Magazine, 49–59 (2000)
[2] Laurila, M.J., Bachmann, C.C.: X-ray fluorescence measuring system and methods for trace elements. US Patent US 2004/0240606 (2004)
[3] Kumral, M.: Bed blending design incorporating multiple regression modelling and genetic algorithms. International Journal of Surface Mining, Reclamation and Environment 17, 98–112 (2003)
[4] Pavloudakis, F., Agioutantis, Z.: Simulation of bulk solids blending in longitudinal stockpiles. Journal of the South African Institute of Mining and Metallurgy 106, 229–237 (2006)
[5] Fischer, A., Shukla, P.K.: A Levenberg-Marquardt algorithm for unconstrained multicriteria optimization. Oper. Res. Lett. 36(5), 643–646 (2008)
[6] Deb, K.: Multi-objective optimization using evolutionary algorithms. Wiley (2001)
[7] Durillo, J.J., Nebro, A.J., Luna, F., Alba, E.: On the Effect of the Steady-State Selection Scheme in Multi-Objective Genetic Algorithms. In: Ehrgott, M., Fonseca, C.M., Gandibleux, X., Hao, J.-K., Sevaux, M. (eds.) EMO 2009. LNCS, vol. 5467, pp. 183–197. Springer, Heidelberg (2009)
[8] Deb, K., Gupta, S.: Understanding knee points in bicriteria problems and their implications as preferred solution principles. Engineering Optimization 43(11), 1175–1204 (2011)
[9] Fonseca, C.M., Guerreiro, A.P., López-Ibáñez, M., Paquete, L.: On the Computation of the Empirical Attainment Function. In: Takahashi, R.H.C., Deb, K., Wanner, E.F., Greco, S. (eds.) EMO 2011. LNCS, vol. 6576, pp. 106–120. Springer, Heidelberg (2011)

[10] Shukla, P.K., Hirsch, C., Schmeck, H.: A Framework for Incorporating Trade-Off Information Using Multi-Objective Evolutionary Algorithms. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part II. LNCS, vol. 6239, pp. 131–140. Springer, Heidelberg (2010)

[11] Shukla, P.K., Hirsch, C., Schmeck, H.: Towards a Deeper Understanding of Trade-offs Using Multi-objective Evolutionary Algorithms. In: Di Chio, C., Agapitos, A., Cagnoni, S., Cotta, C., de Vega, F.F., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Langdon, W.B., Merelo-Guervós, J.J., Preuss, M., Richter, H., Silva, S., Simões, A., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Togelius, J., Urquhart, N., Uyar, A.Ş., Yannakakis, G.N. (eds.) EvoApplications 2012. LNCS, vol. 7248, pp. 396–405. Springer, Heidelberg (2012)

Study of Cancer Hallmarks Relevance Using a Cellular Automaton Tumor Growth Model

José Santos and Ángel Monteagudo

Computer Science Department, University of A Coruña, Spain
jose.santos@udc.es

Abstract. We studied the relative importance of the different cancer hallmarks in tumor growth in a multicellular system. Tumor growth was modeled with a cellular automaton which determines cell mitotic and apoptotic behaviors. These behaviors depend on the cancer hallmarks acquired in each cell as consequence of mutations. Additionally, these hallmarks are associated with a series of parameters, and depending on their values and the activation of the hallmarks in each of the cells, the system can evolve to different dynamics. Here we focus on the relevance of each hallmark in the progression of the first avascular phase of tumor growth and in representative situations.

1 Introduction and Previous Work

Cancer is a disease which arises from mutations in single somatic cells. These mutations alter the proliferation control of the cells which leads to uncontrolled cell division, forming a neoplastic lesion that may be invasive (carcinoma) or benign (adenoma). These two properties are in turn driven by what mutations the cells have acquired. In the invasive case the tumor grows in an uncontrolled manner up to a size of approximately 10^6 cells [4]. At this size the diffusion driven nutrient supply of the tumor becomes insufficient and the tumor must initiate new capillary growth (angiogenesis). When the tumor has been vascularized the tumor can grow further and at this stage metastases are often observed.

Although there are more than 200 different types of cancer that can affect every organ in the body, they share certain features. Thus, Hanahan and Weinberg described the phenotypic differences between healthy and cancer cells in a landmark article entitled "The Hallmarks of Cancer" [7]. The six essential alterations in cell physiology that collectively dictate malignant growth are: self-sufficiency in growth signals, insensitivity to growth-inhibitory (antigrowth) signals, evasion of programmed cell death (apoptosis), limitless replicative potential, sustained angiogenesis, and tissue invasion and metastasis. In a recent update [8] the authors included two more hallmarks: reprogramming of energy metabolism and evasion of immune destruction, that emerged as critical capabilities of cancer cells. Moreover, the authors described two enabling characteristics or properties of neoplastic cells that facilitate acquisition of hallmark capabilities: genome instability and tumor-promoting inflammation (mediated by immune system cells recruited to the tumor site).

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 489–499, 2012.

In Artificial Life terms [10], tumor growth in multicellular systems is an example of emergent behavior, which is present in systems whose elements interact locally, providing global behavior which is not possible to explain from the behavior of a single element, but rather from the "emergent" consequence among the interactions of the group. In this case, it is an emergent consequence of the local interactions between the cells and their environment. Emergent behavior was studied in Artificial Life using models like Cellular Automata (CA) and Lindenmayer Systems [9] [10]. As indicated by Ilachinski [9], CAs have been the focus of attention because of their ability to generate a rich spectrum of complex behavior patterns out of sets of relatively simple underlying rules and they appeared to capture many essential features of complex self-organizing cooperative behavior observed in real systems.

One of the traditional approaches to model cancer growth was the use of differential equations to describe avascular, and indeed vascular, tumor growth. CA approaches make easy the modeling at cellular level, where the state of each cell is described by its local environment. Thus, different works have appeared which used the CA capabilities for different purposes in tumor growth modeling [11]. For example, Bankhead and Heckendorn [2] used a CA which incorporated a simplified genetic regulatory network simulation to control cell behavior and predict cancer etiology. Ribba et al. [12] used a hybrid CA which combined discrete and continuous fields, as it incorporated nutrient and drug spatial distribution together with a simple simulation of the vascular system in a 2D lattice model, and with the aim of assessing chemotherapy treatment for non-Hodgkin's lymphoma. In the CA model of Gerlee and Anderson [4] each cell was equipped with a micro-environment response network (modeled with a neural network), that determined the behavior of the cell based on the local environment. Their focus was on the analysis of tumor morphologies under different conditions like oxygen concentration. Gevertz et al. [5] used a CA model to study the impact that organ-imposed physical confinement and heterogeneity have on tumor growth, that is, to incorporate the effects of tissue shape and structure.

Previous works have used CA models based on the presence of the hallmarks. For example, Abbott et al. [1] investigated the dynamics and interactions of the hallmarks in a CA model in which the main interest of the authors was to describe the likely sequences of precancerous mutations or pathways that end in cancer. They were interested in the relative frequency of different mutational pathways (what sequences of mutations are most likely), how long the different pathways take, and the dependence of pathways on various parameters associated with the hallmarks. In the work of Basanta et al. [3], a 2D cellular automaton modeled key cancer cell capabilities based on the Hanahan and Weinberg hallmarks. The authors focused their work on analyzing the effect of different environmental conditions on the sequence of acquisition of phenotypic traits and tumor expansion. Their results indicated that microenvironmental factors such as the local concentration of oxygen or nutrients and cell overcrowding may determine the expansion of the tumor colony.

We also used a CA model which determines the behavior of cells based on the Hanahan and Weinberg hallmarks. Nevertheless, our aim is different, as our simulation tries to determine the dependence of the cellular system behavior, at cellular level, on the presence of the different cancer cell hallmarks and their key defining parameters. We focused here on the dependence of the emergent tumor growth behavior on each individual hallmark, studying their relative importance in tumor development in the first avascular phase. These dependences are difficult to foresee without a model and associated simulating tool.

As indicated recently by Hanahan and Weinberg [8], in addition to providing a solid basis for cancer research, the hallmarks have served to identify certain cell functions that have become therapeutic targets. However, the utility of such attempts has been limited because tumor cells have demonstrated an ability to develop resistance to drugs that disrupt a single pathway. This adaptability of cancer cells suggests to Hanahan and Weinberg that simultaneous targeting of two or more hallmark pathways may be a more effective approach to therapy. So, our study can help to discern what are such most relevant hallmarks which can be targeted and in each multicellular system situation.

2 Methods for the Cellular System Modeling

2.1 Cancer Hallmarks

In the simulation each cell resides in a site in a cubic lattice and has a "genome" associated with different cancer hallmarks. The essential alterations in cell physiology that collectively dictate malignant growth are [6] [7]:

SG. Self-Growth: Growth even in the absence of normal "go" signals. Most normal cells wait for an external message (growth signals from other cells) before dividing. Cancer cells often counterfeit their own pro-growth messages.

IGI. Ignore Growth Inhibit: As the tumor expands, it squeezes adjacent tissue, which sends out chemical messages that would normally bring cell division to a halt. Malignant cells ignore the commands, proliferating despite anti-growth signals issued by neighboring cells.

EA. Evasion of apoptosis: In healthy cells, genetic damage above a critical level usually activates a suicide program (programmed cell death or apoptosis). Cancer cells bypass this mechanism.

AG. Ability to stimulate blood vessel construction: Tumors need oxygen and nutrients to survive. They obtain them by co-opting nearby blood vessels to form new branches that run throughout the growing mass (angiogenesis).

EI. Effective immortality: Healthy cells can divide no more than several times (< 100). The limited replicative potential arises because, with the duplication, there is a loss of base pairs in the telomeres (chromosomes ends which protect the bases), so when the DNA is unprotected, the cell dies. Malignant cells overproduce the telomerase enzyme, avoiding the telomere shorthening, so such cells overcome the reproductive limit.

Table 1. Definition of the parameters associated with the hallmarks

Parameter name	*Default value*	*Description*
Telomere length (tl)	100	Initial telomere length in each cell. Every time a cell divides, the lenght is shortened by one unit. When it reaches 0, the cell dies, unless the "Effective immortality" hallmark (EI) is ON.
Evade apoptosis (e)	10	A cell with n hallmarks mutated has an extra n/e likelihood of dying each cell cycle, unless the "Evade apoptosis" hallmark (EA) is ON.
Base mutation rate (m)	100000	Each gene (hallmark) is mutated (when the cell divides) with a $1/m$ chance of mutation.
Genetic instability (i)	100	There is an increase of the base mutation rate by a factor of i for cells with this mutation (GI).
Ignore growth inhibit (g)	10	As in [1], cells with the hallmark "Ignore growth inhibit" (IGI) activated have a probability $1/g$ of killing off a neighbor to make room for mitosis.
Random cell death (a)	1000	In each cell cycle every cell has a $1/a$ chance of death from several causes.

MT. Power to invade other tissues and spread to other organs: Cancers usually become life-threatening only after they somehow disable the cellular circuitry that confines them to a specific part of the organ in which they arose. New growths appear and eventually interfere with vital systems.

GI. Genetic instability: It accounts for the high incidence of mutations in cancer cells, allowing rapid accumulation of genetic damage. It is an enabling characteristic of cancer [8] since, while not necessary in the progression from neoplasm to cancer, makes such progression much more likely [3]. The simulation implies that the cells with this factor will increase their mutation rate.

2.2 Event Model

In our modeling, each cell genome indicates if any hallmark is activated as consequence of mutations. Metastasis and angiogenesis are not considered, as we are interested in this work in the first avascular phases of tumorigenesis. So, every cell has its genome which consists in five hallmarks plus some parameters particular to each cell. All the parameters are commented in Table 1. The parameters *telomere length* and *base mutation rate* can change their values in a particular cell over time, as explained in the table. The cell's genome is inherited by the daughter cells when a mitotic division occurs. The default values indicated in Table 1 are the same as those used in [1]. Also, Basanta et al. [3] worked with parameters, such as base mutation rate (10^{-5}) and mutation rate increase for cells with acquired genetic instability ($i = 100$), with the same default values.

In the simulation of the cell life cycle, most elements do not change observably each time step. The only observable changes to cells are apoptosis and mitosis. In a tissue, only a fraction of all cells are undergoing such transitions at any

given time. We used an event model, similar to that used by Abbott et al. [1], summarized in Algorithm 2.1 and which takes into account the main aspects of the cell cycle from the application point of view. A mitosis is scheduled several times in the future, being a random variable distributed uniformly between 5 and 10 time steps, simulating the variable duration of the cell life cycle (between 15 and 24 hours). Finally, a grid with 10^6 sites represents approximately 0.1 mm^3 of tissue.

Algorithm 2.1. EVENT MODEL FOR CANCER SIMULATION()

```
t ← 0    // Simulation time. Initial cell at the center of the grid.
SCHEDULE A MITOTIC EVENT(5, 10) // Schedule a mitotic event with a random time
        //  (ts) between 5 and 10 time instants in the future (t+ts). The events
        // are stored in an event queue. The events are ordered on event time.
while event in the event queue
do { POP EVENT( ) // Pop event with the highest priority (the nearest in time).
     t ← t of popped event
     RANDOM CELL DEATH TEST( ) // The cell can die with a given probability.
     GENETIC DAMAGE TEST( ) // The larger the number of hallmark mutations,
                            // the greater the probability of cell death. If
                          // "Evade apoptosis" (EA) is ON, death is not applied.
     MITOSIS TESTS( ) :
       GROWTH FACTOR CHECKING( ) // cells can perform divisions only
         // if they are within a predefined spatial boundary which sufficient
         // growth factor; beyond this area cells cannot perform mitosis,
         // unless the hallmark "Self-growth" (SG) is ON.
       IGNORE GROWTH INHIBIT CHECKING( ) // If there are not empty cells in
         // the neighborhood, the cell cannot perform a mitotic division. If the
         // "Ignore growth inhibit" hallmark (IGI) is ON, then the cell competes
         // for survival with a neighbor cell and with a likelihood of success.
       LIMITLESS REPLICATIVE POTENTIAL CHECKING( )// If the telomere length
         // is 0, the cell dies, unless the hallmark "Effective immortality"
         // (Limitless replicative potential, EI) is mutated (ON).
     if the three tests indicate possibility of mitosis
       then
       PERFORM MITOSIS( ) :
         // Increase the base mutation rate if genetic instability (GI) is ON.
         // Add mutations to the new cells according to base mutation rate(1/m).
         // Decrease telomere length in both cells.
       PUSH EVENTS( )
         // Schedule mitotic events (push in event queue) for both cells:
         // Mother and daughter, with the random times in the future.
       else PUSH EVENT( )
         // Schedule a mitotic event (in queue) for mother cell.
   }
```

The simulation begins by initializing all elements of the grid to represent empty space. Then, the element at the center of the grid is changed to represent a single normal cell (no mutations). Mitosis is scheduled for this initial cell. After the new daughter cells are created, mitosis is scheduled for each of them, and so on. Each mitotic division is carried out by copying the genetic information

(the hallmark status and associated parameters) of the cell to an unoccupied adjacent space in the grid. Random errors occur in this copying process, so some hallmarks can be activated, taking into account that once a hallmark is activated in a cell, it will be never repaired by another mutation [1].

Frequently, cells are unable to replicate because of some limitation, such as contact inhibition or insufficient growth signal. Cells overcome these limitations through mutations in the hallmarks. Regarding hallmark *self-growth* (SG), as in [1] and [3], cells can perform divisions only if they are within a predefined spatial boundary, which represents a threshold in the concentration of growth factor; beyond this area (95% of the inner space in each dimension, which represents 85.7% of the 3D grid inner space) growth signals are too faint to prompt mitosis (unless hallmark SG is ON). Moreover, cells undergo random cell death with low probability ($1/a$ chance of death, where a is a tunable parameter).

So, our model corresponds to an "on-lattice model" as called by Rejniak et al. [11], where the model is constrained by a cubic lattice structure that defines the locations of cells and cell-cell interaction neighborhods, although there are other models that describe the spatial and morphological features of cancer development in a more biologically plausible way like the Cellular Potts or the Voronoi diagram-based off-lattice models [11].

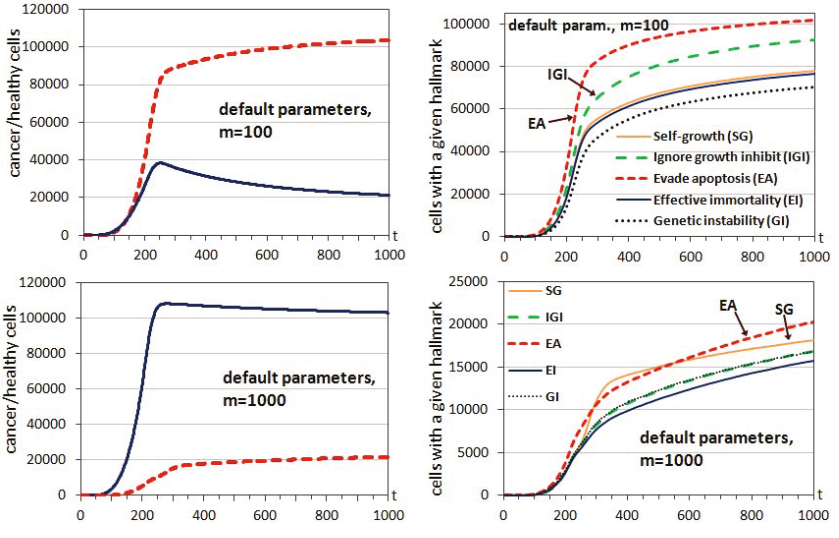


Fig. 1. Left: Evolution through time iterations of the number of healthy cells (continuous lines) and cancer cells (dashed lines) for different base mutation rates ($1/m$) and default parameters. Right: Evolution of the number of cells with a hallmark acquired.

3 Results

3.1 Simulations with Different Hallmark Parameters

First, we run several simulations with representative hallmark parameters. Figure 1 shows the evolution over time of the number of healthy and cancer cells for two

different values of the parameter m, which defines the base mutation rate, maintaining the rest of the parameters in their default values and using the same grid size (125000) employed in [1]. The number of time iterations was 1000 in the different runs. Given the stochastic nature of the problem, the graphs are always an average of 5 different runs. A cell was considered as cancerous if any of the hallmarks was present. As expected, with increasing base mutation rate ($1/m$), the increase in cancer cells becomes faster. For lower values of the base mutation rate it is difficult to obtain rapid cancer progression, so we selected those two high values.

The right part of Figure 1 shows the time evolution of the cells with a given hallmark and such standard parameters. Despite the rapid and initial cancer cell progression, with $m = 100$, two hallmarks present an advantage for cancer cell proliferation: *evade apoptosis* (EA) and *ignore growth inhibit* (IGI). The first one dominates in the cancer cell population because, as there are many mutations in the cells, the apoptosis mechanism eliminates many of the mutated cells, except those that have the hallmark EA acquired, which escape such control so they proliferate in the cell population. The second hallmark is necessary when the space is full, because in this situation there are no vacant sites for cell proliferation, except for those with hallmark IGI acquired (the free space limitation can be ignored by such cells). Using a lower base mutation rate ($m = 1000$), the hallmark *self-growth* SG is relatively more predominant than IGI, as cells with SG acquired proliferate rapidly when the cells have reached the limits of the area filled with growth factor. Remember that these hallmarks, that allow the cells to escape those limits, are acquired by the offspring, so the daughters can continue proliferating.

In Figure 2 we repeated the simulations but using a parameter set that facilitates the appearance of cancer cells. We selected values as the ones used by Abbott et al. [1] ($m = 100000$, $tl = 35$, $e = 20$, $i = 100$, $g = 4$, $a = 400$ and a grid size of 125000) for the determination of possible mutational pathways, that is, the sequence of appearance of hallmarks that end in a tumor growth. For example, the lower value of tl implies fewer mitoses in healthy cells, and the lower value of a facilitates that more vacant sites are available for cancer cells to propagate, in connection with the higher probability of replacing neighbors when making room for mitosis (lower value of g).

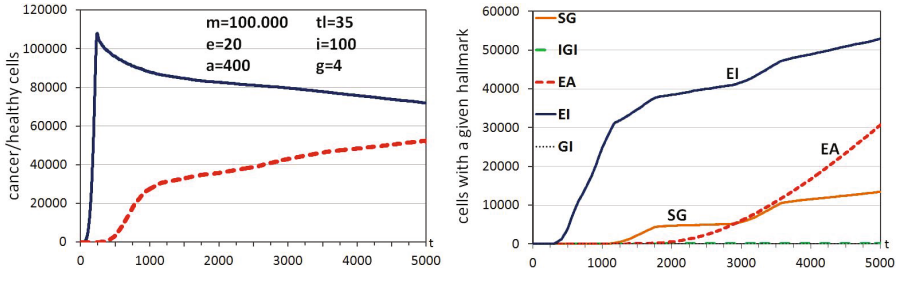


Fig. 2. Left: Time evolution of the number of healthy cells (continuous line) and cancer cells (dashed line) with a parameter set which facilitates cancer growth. Right: Time evolution of the number of cells with a hallmark acquired. All the graphs are an average of 5 independent runs.

The right part of Figure 2 shows the time evolution of the cells with a given hallmark and such parameter set. The dominant hallmark in the tumor growth is now *effective immortality* (EI), allowing the progression of the cells with such mutation even when the telomere length reaches its limit. Such cells have a clear advantage with respect to the other cells, which die after the maximum number of 35 divisions. This explains the rapid proliferation of the hallmark EI, before iteration 1000, when the healthy cells have performed their maximum number of mitotic divisions. Figure 3 shows snapshots at different time states of the multicellular system in a run with such parameters. In this case, we used a grid size of 10^6, for a better visualization of the tumor progression. These snapshots show again how EI is the dominant hallmark in such conditions (green color cells in Figure 3 have hallmark EI acquired).

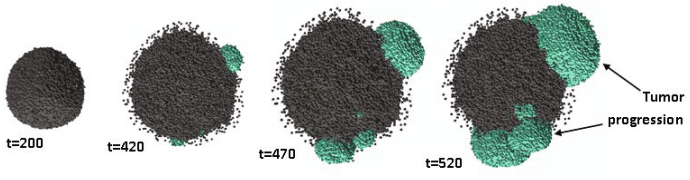


Fig. 3. Snapshots at different time steps using the parameters of Figure 2

3.2 Relevance of Hallmarks

Our aim is to inspect the relative importance of each hallmark in the emergent behavior of tumor growth. To answer this, we can analyze the growth behavior when the individual hallmarks are not present or do not imply any effect on the cellular behavior. This is the same as considering that mutations do not activate a particular hallmark. We selected two of the previous representative cases to study the effect of not considering the individual hallmarks, that is, to inspect the relative importance of each hallmark in the cancer growth behavior. First, Figure 4 (Left part) shows the evolution across time iterations of the number of cancer cells (grid size=125000), using the default parameters with $m = 100$, when all the hallmarks are considered (previously shown in Fig. 1), and when a particular hallmark is not taken into account in the rules of apoptotic and mitotic behaviors. As seen in Figure 4, the most important hallmark regarding the growth of cancer cells is *evade apoptosis* (EA), since its elimination implies a high decrease in the number of cancer cells. This is because, without the consideration of EA, all the cancer cells have a probability of death by apoptosis, so cancer cell proliferation is highly decreased.

The next most important hallmark is *ignore growth inhibit* (IGI), since its elimination implies also an important decrease in the number of cancer cells. This is because when the grid is almost full of healthy or cancer cells, after time iteration 200, the main limit for the mitotic divisions is the available free space. In this situation, the cancer cells with the hallmark IGI activated have an advantage, as they can replace (with a given probability) a neighbor cell to replicate. So, if this advantage does not exist when hallmark IGI is not considered, the cancer cells tend

to remain stable in number, even with this very high base mutation rate $(1/m)$. A hallmark with similar relevance is *genetic instability* (GI), as without its consideration there are fewer mutations or acquisition of hallmarks. The previous effects are not present with the elimination in the simulation of the other hallmarks, as it implies a smaller decrease in the number of cancer cells.

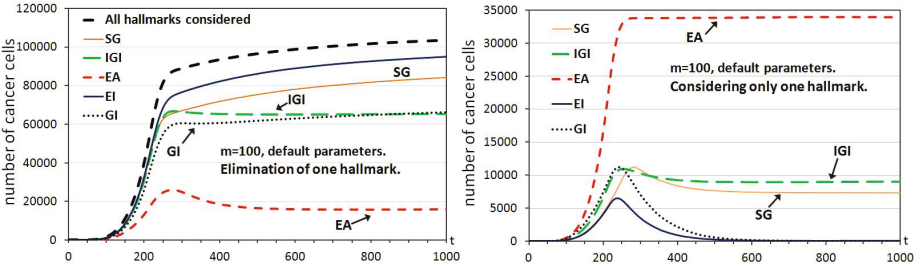


Fig. 4. Left: Effect of elimination of an individual hallmark. Right: Number of cancer cells when only one hallmark is considered. Simulations with parameter default values and $m = 100$, averaged with 5 independent runs.

The right part of Figure 4 shows the same evolution when only one particular hallmark is considered. As the Figure denotes, hallmarks EA and IGI are again the most relevant, and because the same reasons exposed. Note that now, when only *genetic instability* (GI) is considered, the number of cancer cells with only such a mutation cannot growth across time iterations. This is because GI only increments the mutations in such cells for the acquisition of the other hallmarks that have a possible effect on the proliferation of cancer cells. Note also the difference between the hallmark relevance and the number of cells with a given hallmark (Fig. 1), since the relative relevance between EA and other hallmarks is not reflected in Figure 1.

In Figure 5 we repeated the same analysis with the parameter set previously used in Fig. 2, which facilitates the appearance of cancer cells. As the Figure shows,

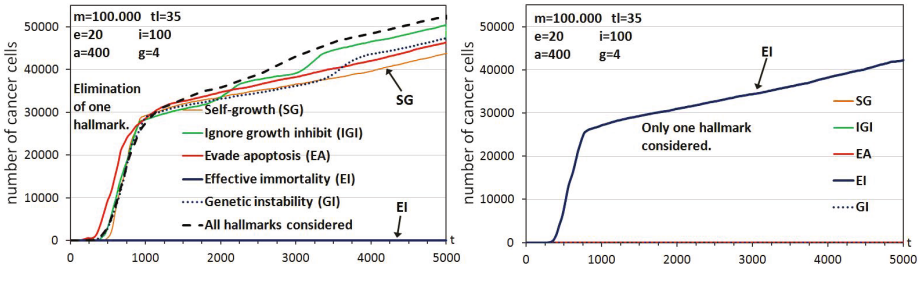


Fig. 5. Number of cancer cells when an individual hallmark is not considered (Left) and when only one hallmark is considered (Right). Simulations with parameter values of Figure 2, averaged with 5 independent runs.

when we do not consider the hallmark *effective immortality* (EI) in the simulation, the number of cancer cells is maintained to a minimum (close to 0, dark blue line). This is because, in this case, the great advantage of the limitless replicative potential is never present, so all cells have the same limit of replications imposed by the initial telomere length. The other hallmarks do not have relevance except the low relevance of *self-growth* (SG), as not considering it eliminates the final possible progression of cancer cells in the area without growth factor.

4 Conclusions

We used a cellular automaton model to simulate tumor growth at cellular level, based on the cancer hallmarks acquired in each cell. We focused here on the relevance or relative importance of the different hallmarks in the avascular tumor progression. The experimentation performed showed that the effect of elimination of hallmarks is different depending on the main advantage of cancer cells to propagate. With high mutation rates, the most relevant hallmark is *evade apoptosis*. If the space is full of cells, a relevant hallmark is *ignore growth inhibit*, as it allows cancer cell proliferation when there is no available free space. When the cells have reached the proliferation limit imposed by the telomeres, then the most important hallmark for cancer proliferation is *effective immortality*, given its advantage with respect to cells without it in such stage. So, the simulations can help to analyze what are the most relevant hallmarks which can be targeted and in each multicellular system situation.

Acknowledgments. This paper has been funded by the Ministry of Science and Innovation of Spain (project TIN2011-27294).

References

1. Abbott, R.G., Forrest, S., Pienta, K.J.: Simulating the hallmarks of cancer. Artificial Life 12(4), 617–634 (2006)
2. Bankhead, A., Heckendorn, R.B.: Using evolvable genetic cellular automata to model breast cancer. Genet Program Evolvable Mach. 8, 381–393 (2007)
3. Basanta, D., Ribba, B., Watkin, E., You, B., Deutsch, A.: Computational analysis of the influence of the microenvironment on carcinogenesis. Mathematical Biosiciences 229, 22–29 (2011)
4. Gerlee, P., Anderson, A.R.A.: An evolutionary hybrid cellular automaton model of solid tumour growth. Journal of Theoretical Biology 246(4), 583–603 (2007)
5. Gevertz, J.L., Gillies, G.T., Torquato, S.: Simulating tumor growth in confined heterogeneous environments. Phys. Biol. 5 (2008)
6. Gibbs, W.W.: Untangling the roots of cancer. Scientific American 289, 56–65 (2003)
7. Hanahan, D., Weinberg, R.A.: The hallmarks of cancer. Cell 100, 57–70 (2000)
8. Hanahan, D., Weinberg, R.A.: Hallmarks of cancer: The next generation. Cell 144(5), 646–674 (2011)
9. Ilachinski, A.: Cellular automata. A discrete universe. World Scientific (2001)

10. Langton, C.G.: Artificial Life: An overview. MIT Press (1995)
11. Rejniak, K.A., Anderson, A.R.A.: Hybrid models of tumor growth. WIREs Syst. Biol. Med. 3, 115–125 (2010)
12. Ribba, B., Alarcón, T., Marron, K., Maini, P.K., Agur, Z.: The Use of Hybrid Cellular Automaton Models for Improving Cancer Therapy. In: Sloot, P.M.A., Chopard, B., Hoekstra, A.G. (eds.) ACRI 2004. LNCS, vol. 3305, pp. 444–453. Springer, Heidelberg (2004)

Between Selfishness and Altruism: Fuzzy Nash–Berge-Zhukovskii Equilibrium

Réka Nagy[1], Noémi Gaskó[1,2], Rodica Ioana Lung[1], and D. Dumitrescu[1]

[1] Babes-Bolyai University, Cluj-Napoca, Romania
[2] Technical University of Cluj-Napoca, Romania

Abstract. Nash equilibrium in many cases is not the best choice for human players. In case of trust games the Nash equilibrium is often mutual defection which is the worst possible outcome for all players. The Berge-Zhukovskii equilibrium models a more cooperative behavior, so in case of trust games, when players gain by cooperating, it is usually a better choice than Nash equilibrium. Real life results show that players rarely follow the theoretical predictions. Our aim is to find new equilibria types that offer a more realistic modeling of human players. The fuzzy Nash–Berge-Zhukovskii equilibrium is proposed which is a fuzzy combination of the Nash and Berge-Zhukovskii equilibrium. Several continuous trust games are investigated. Numerical results indicate that fuzzy Nash–Berge-Zhukovskii equilibrium is suitable to model real-life situations.

1 Introduction

The most important equilibrium concept in game theory, Nash equilibrium, is not always the most efficient solution concept. In many cases playing Nash equilibrium is not the most favorable choice since Nash equilibrium rarely assures maximal payoffs. Trust games are a class of games where players end up with greater payoffs by trusting their opponents and choosing a cooperative strategy, than by mutual defection. Also, since the payoff for defecting with a cooperative opponent is larger than the payoff for cooperation, the temptation for defection is high. In most of the cases the Nash equilibrium of trust games is mutual defection, that is the worst possible outcome for all players.

Other solution concepts, like Pareto or Berge-Zhukovskii equilibrium, are often better choices in case of trust games. Pareto equilibrium ensures optimal payoffs for all players while Berge-Zhukovskii equilibrium models a type of altruism. Berge-Zhukovskii players, when choosing their strategy, beyond their gain, also take in consideration the gain of their opponent. For trust games, both Pareto and Berge-Zhukovskii equilibria, usually ensure greater payoffs for all players than Nash equilibrium.

However, our opinion is, that standard game equilibria presume some restrictions. In real life players are rarely rational agents only acting to maximize their payoffs. Real life players can be more or less cooperative, more or less selfish and their actions are rarely uniform. One simple step towards a more realistic approach is to relax the rationality principle, and allow different rationality types in

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 500–509, 2012.

a single game. In [2] a fuzzy Nash-Pareto equilibrium is proposed. This concept allows players to be biased towards a certain type of rationality, which ensures a more realistic modeling of human players. According to [3] Fuzzy Nash-Pareto equilibrium is a suitable concept to model the human behavior for the discrete centipede game.

A more general approach is to explore trust games with continuous strategy set. Our intuition is that fuzzy equilibria might offer a better modeling of real-world players. Since fuzzy Nash-Pareto equilibrium does not offer promising results our goal is to find an equilibrium concept that would capture a more realistic situation.

1.1 Game Theory Prerequisites

Mathematically a finite non-cooperative one shot game is a system $G = (N, (S_i, u_i), i = 1, ..., n)$, where:

- N represents the set of players, and n is the number of players;
- for each player $i \in N$, S_i is the set of available actions; $S = S_1 \times S_2 \times ... \times S_n$, is the set of all possible situations of the game. Each $s \in S$ is a strategy (or strategy profile) of the game;
- for each player $i \in N$, $u_i : S \to R$ represents the payoff function of i.

Denote by (s_{i_j}, s^*_{-i}) the strategy profile obtained from s^* by replacing the strategy of player i with s_{i_j} i.e. $(s_i, s^*_{-i}) = (s^*_1, s^*_2, ..., s^*_{i-1}, s_i, s^*_{i+1}, ..., s^*_n)$.

A solution of the game is called a *game equilibrium*. At equilibrium all players are contended with their outcome, and they are not willing to switch their strategies.

Nash Equilibrium. A strategy profile is a *Nash equilibrium* if none of the players have the incentive to unilaterally deviate [7] i.e. no player can improve her payoff by modifying her strategy while the others do not modify theirs.

More formally: a strategy profile $s^* \in S$ is a Nash equilibrium if the inequality holds: $u_i(s^*) \geq u_i(s_i, s^*_{-i}), \forall i = 1, ..., n, \forall s_i \in S_i$.

Berge-Zhukovskii Equilibrium. In contrast to the Nash equilibrium, where players are selfregarding, the Berge-Zhukovskii equilibrium [11] allows reaching cooperative features making it possible to determine cooperation in a non-cooperative game.

The strategy s^* is a *Berge-Zhukovskii equilibrium* when no group of players can improve the payoff for any of the n players by changing their strategy.

More formally: Let $N - i$ denote any group of players which excludes player i (can be excluded other players, too). The strategy profile s^* is a Berge-Zhukovskii equilibrium if the inequality $u_i(s^*) \geq u_i(s^*_i, s_{N-i})$ holds for each player $i = 1, ..., n$, and $s_{N-i} \in S_{N-i}$.

1.2 Trust Games

An interesting phenomena can be observed in case of trust games. Trust games are a class of games in which players obtain much better results, higher payoffs, if they trust each other and choose a cooperative strategy than in case of mutual defection. Moreover if one player defects while the other cooperates the payoff for the defecting player is much better, so the temptation to defect is considerably high.

Very often, when players gain more by cooperating than defecting, Nash equilibrium is not the best choice for players. In these type of games the Nash equilibrium is mutual defection and players following Nash rationality end up with the worst outcome.

The Berge-Zhukovskii equilibrium models a type of altruism. Players choosing a Berge-Zhukovskii rationality are more other-regarding when choosing their strategies, as they also consider the payoffs of the other players. So usually in trust games the Berge-Zhukovskii equilibrium represents mutual cooperation which is a favorable outcome for all players.

We think that it would be interesting to investigate rationality types that are between these two extremes (mutual defection or mutual cooperation). With fuzzy Nash–Berge-Zhukovskii (N-BZ) equilibrium various intermediate states can be depicted. Players can be more or less biased towards a certain rationality, for example a player can have a membership degree of 0.7 to Nash rationality and 0.3 to Berge-Zhukovskii rationality. Thus the fuzzy N-BZ equilibrium offers a more realistic modeling of human players.

2 Generative Relations for Game Equilibria

Game equilibria may be characterized by generative relations on the set of game strategies [5]. The idea is that the non-dominated strategies with respect to the generative relation equals (or approximate) the equilibrium set.

Let us consider a relation $\mathcal{R}$ over $S \times S$. A strategy s is non dominated with respect to relation $\mathcal{R}$ if $\nexists s^* \in S : (s, s^*) \in \mathcal{R}$. Let us denote by NDR the set of non- dominated strategies with respect to relation $\mathcal{R}$. A subset $S' \subset S$ is non-dominated with respect to $\mathcal{R}$ if and only if $\forall s \in S'$, $s \in NDR$.

Relation $\mathcal{R}$ is said to be a *generative relation* for the equilibrium E if and only if the set of non-dominated strategies with respect to $\mathcal{R}$ equals the set E of strategies i.e. $NDR = E$.

2.1 Generative Relation for Nash Equilibrium

Let s and s^* be two pure strategies and $k(s^*, s)$ denotes the number of players which benefit by deviating from s^* towards s [5]:

$$k(s^*, s) = card\{i \in N, u_i(s_i, s^*_{-i}) > u_i(s^*), s_i \neq s^*_i\}.$$

Let $s^*, s \in S$. We say the strategy s^* is better than strategy s with respect to Nash equilibrium, and we write $s^* \prec_N s$, if the following inequality holds:

$k(s^*, s) < k(s, s^*)$. $k(s^*, s)$ is a relative quality measure of s and s^* - with respect to the Nash equilibrium. The relation $\prec_N$ can be considered as the *generative relation of Nash equilibrium*, i.e. that the set of non-dominated strategies with respect to $\prec_N$ induces the *Nash equilibrium* [5].

2.2 Generative Relation for Berge-Zhukovskii Equilibrium

Consider two strategy profiles s^* and s from S. Denote by $b(s^*, s)$ the number of players who lose by remaining to the initial strategy s^*, while the other players are allowed to play the corresponding strategies from s and at least one player switches from s^* to s.

We may express $b(s^*, s)$ as [4]:

$$b(s^*, s) = card[i \in N, u_i(s^*) < u_i(s_i^*, s_{N-i})].$$

Let $s, s^* \in S$. We say the strategy s^* is better than strategy s with respect to Berge-Zhukovskii equilibrium, and we write $s^* \prec_{BZ} s$, if and only if the inequality $b(s^*, s) < b(s, s^*)$ holds. We may consider relation $\prec_{BZ}$ as a *generative relation of the Berge-Zhukovskii equilibrium*. This means the set of the non-dominant strategies with respect to the relation $\prec_{BZ}$ equals the set of Berge-Zhukovskii equilibria.

3 Evolutionary Equilibria Detection

Games can be viewed as multiobjective optimization problem, where the payoffs of the participating players are to be maximized. All of the objectives to be optimized are uniform and equally important. A solution of the game is called an equilibrium. At equilibrium all players are contended with their outcome, and they are not willing to switch their strategies.

An appealing technique is the use of generative relations and evolutionary algorithms for detecting equilibrium strategies. The payoff of each player is treated as an objective and the generative relation induces an appropriate dominance concept, which is used for fitness assignment purpose. Evolutionary multiobjective algorithms are thus suitable tools in searching for game equilibria.

A population of strategies is evolved. A chromosome is an n-dimensional vector representing a strategy profile $s \in S$. The initial population is randomly generated. Population model is generational. The non-dominated individuals from the population of strategy profiles at iteration t may be regarded as the current equilibrium approximation. Subsequent application of the search operators is guided by a specific selection operator induced by the generative relation. Successive populations produce new approximations of the equilibrium front, which hopefully are better than the previous ones.

For evolutionary equilibria detection any state of the art algorithm can be used. In our numerical experiments we use the NSGA2 [1] algorithm but the results were also tested with differential evolution [10]. Our goal is to focus on the detected equilibria types and not on the algorithm used.

4 Fuzzy Equilibria

In non-cooperative game theory each concept of equilibrium may be associated to a rationality type. A more realistic approach is allowing each player to be more or less biased towards a certain rationality type. This bias may be expressed by a fuzzy membership degree. This way several new types of equilibria, like fuzzy Nash-Pareto [2], can be obtained.

4.1 Fuzzy Nash–Berge-Zhukovskii Equilibrium

Let us consider a fuzzy set A_N on the player set N i.e. $A_N : N \to [0,1]$ $A_N(i)$ expresses the membership degree of the player i to the fuzzy class of Nash-biased players. Therefore A_N is the class of Nash-biased players. Similar a fuzzy set $A_{BZ} : N \to [0,1]$ may describe the fuzzy class of Berge-Zhukovskii-biased players.

A fuzzy Nash–Berge-Zhukovskii equilibrium concept is introduced in this section. Let us consider a game involving both Nash and Berge-biased players. It is natural to assume that $\{A_N, A_{BZ}\}$ represents a fuzzy partition of the player set. Therefore the condition $A_N(i) + A_{BZ}(i) = 1$ holds for each player i.

The relative quality measure of two strategies has to involve the fuzzy membership degrees. Let us consider the threshold function:

$$t(a) = \begin{cases} 1, & \text{if } a > 0, \\ 0, & \text{otherwise} \end{cases}$$

The fuzzy version of the quality measure $k(s^*, s)$ is denoted by $E_N(s^*, s)$ and may be defined as

$$E_N(s^*, s) = \sum_{i=1}^{n} A_N(i) t(u_i(s, s^*_{-i}) - u_i(s^*)).$$

$E_N(s^*, s)$ expresses the relative quality of the strategies s^* and s with respect to the fuzzy class of Nash-biased players.
The fuzzy version of $b(s^*, s)$ may be defined as

$$E_{BZ}(s^*, s) = \sum_{i=1}^{n} A_{BZ}(i) t(u_i(s, s^*_{N-i}) - u_i(s^*)).$$

The relative quality measure of the strategies s^* and s with respect to fuzzy Nash–Berge-Zhukovskii rationality may be defined as

$$E(s^*, s) = E_N(s^*, s) + E_{BZ}(s^*, s).$$

Using the relative quality measure E we can compare two strategy profiles.

Let us introduce the relation $\prec_{fNBZ}$ defined as $s^* \prec_{fNBZ} s$ if and only if the strict inequality $E(s^*, s) < E(s, s^*)$ holds.

Fuzzy Nash–Berge-Zhukovskii (N-BZ) equilibrium is the set of non-dominated strategies with respect to the relation $\prec_{fNBZ}$.

5 Numerical Experiments

Evolutionary method described in Section 3 is used for detecting Fuzzy Nash–Berge-Zhukovskii equilibria. The multiobjective evolutionary algorithm used for equilibria detection is NSGA2 [1] with the following parameter settings: *population size=100, no. of generations=100, probability of crossover=0.9, prob. of mutation=0.5.*

5.1 Continuous Centipede Game

Consider a continuous version of the centipede game [9], the *Symmetric Real Time Trust (SRTT) Game* [6].

There is a set of n players. The strategy space of each player is continuous on the real interval $[0, T]$. Each player can make at most a single decision that "stops the clock" at time $e \in [0, T]$. The game starts at time $t = 0$ and ends either when one of the players stops the clock at some time $t < T$ or when T is reached with no player stopping the clock.

Suppose that the game ends at time $e \in [0, T]$ with player i stopping the clock. Than the payoff for the winner i is given by $r_i = \lambda(2^{(t/\theta)})$, where $\theta \geq 1$ and $\lambda > 0$.

Each of the players not stopping the clock receives only a fraction of the winners payoff. More formally, the payoff for the remaining $n - 1$ players is computed from $r_j(t) = \delta r_i(t)$, where $0 < \delta < 1$, $j = 1, 2, ..., n$, and $j \neq i$.

As time is continuous no tie is possible at times $0 < t < T$. If m players $(1 < m \leq m)$ stop the clock at exactly $t = 0$, then one of them is chosen with probability $1/m$ to receive the payoff λ, and the other $m - 1$ players receive $\delta\lambda$.

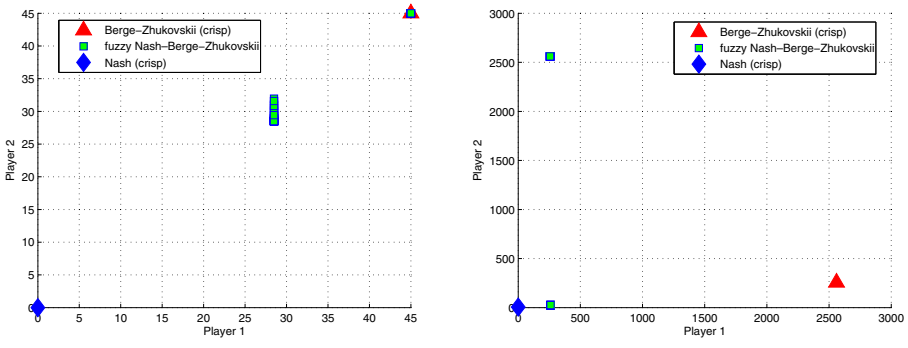
If no player stops the clock (and the game ends at time $t = T$), then the payoff for each of the n players is g, where $0 \leq g < (\lambda 2^{(T/\theta)})$.

The Nash equilibrium of the SRTT game is when all players stop the clock at zero seconds [6], so all players end up with minimal payoffs (zero for all players). However, studies on human players show that people do not play Nash equilibrium. The Berge-Zhukovskii equilibrium of the game is when all players wait until the last moment to press the button. In [6] an experiment based on the SRTT repeated game is presented. Results indicate that human players tend to stop the timer between 25 and 42 seconds (if a unique round is considered). Our aim is to find a fuzzy Nash–Berge-Zhukovskii equilibrium to model this situation.

In order to illustrate the Fuzzy Nash–Berge-Zhukovskii equilibrium we use the continuous centipede (or SRTT) game with the following parameter settings: $n = 2$, $T = 45$, $\theta = 5$, $\lambda = 5$, $\delta = 0.5$ and $g = 0$. Thus there are two players, the player who looses receives 10% of the winner's payoff and if no one stops the clock before 45 seconds the payoff for both players is zero.

Based on our experiments, the Fuzzy Nash-Pareto equilibrium fails to capture an intermediate equilibrium for the SRTT game. For any membership degree the fuzzy Nash-Pareto equilibria correspond to the Nash equilibrium.

Figures 1 and 2 depict the crisp Nash, the crisp Berge-Zhukovskii and the fuzzy N-BZ equilibrium of the SRTT game for various membership degrees. The Nash equilibrium of the game is when both players defect, meaning that they stop the clock at 0 seconds.



(a) Strategy space (b) Payoff space

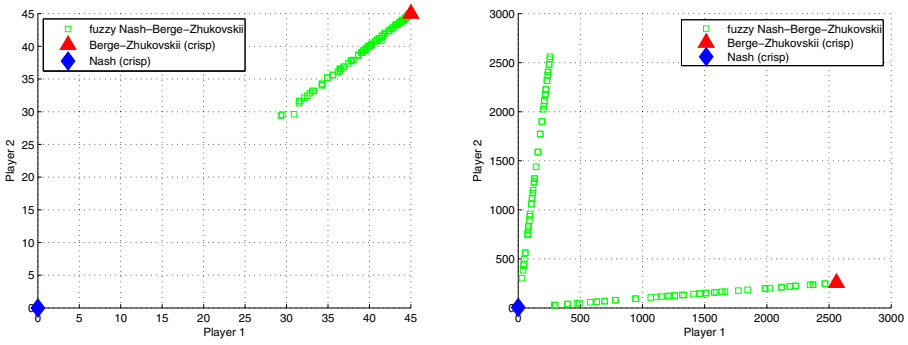
Fig. 1. The detected fuzzy N-BZ equilibrium for the SRTT game with membership degrees $A_N(1) = 0.4, A_{BZ}(1) = 0.6$ and $A_N(2) = 0.6, A_{BZ}(2) = 0.4$

Figure 1 depicts the case when Player 1 has a Nash membership degree of 0.4 (thus a Berge-Zhukovskii membership degree of 0.6) and Player 2 has a Nash membership degree of 0.6 (thus a Berge-Zhukovskii membership degree of 0.4). The fuzzy N-BZ equilibrium is when Player 1 stops the clock around 28 seconds, thus receives a higher payoff.

In cases where both players have equal memberships to both Nash and Berge-Zhukovskii equilibria ($A_N(1), A_N(2)$ and $A_{BZ}(1), A_{BZ}(2)$) the fuzzy N-BZ equilibrium converges either to crisp Nash or crisp Berge-Zhukovskii equilibria. For membership degrees $A_N(1), A_N(2) > 0.5$ (and $A_{BZ}(1), A_{BZ}(2) < 0.5$) the fuzzy N-BZ equilibrium is the same as the Nash equilibrium otherwise if $A_N(1), A_N(2) < 0.5$ (and $A_{BZ}(1), A_{BZ}(2) > 0.5$) the fuzzy N-BZ equilibrium is the same as the Berge-Zhukovskii equilibrium.

Figure 2 depicts the case when both players have equal membership degrees of 0.5. In this case the fuzzy N-BZ equilibrium consists of a set of points, meaning that both players stop the clock somewhere between 25 and 45 seconds. The payoffs for the players is between 256 and 2560 depending on the chosen strategy. Thus, both players end up with higher payoffs than in case of the Nash equilibrium. This equilibrium corresponds to the real-life results presented in [6].

Our numerical experiments show that the fuzzy Nash–Berge-Zhukovskii equilibria always lie between the crisp Nash and crisp Berge-Zhukovskii equilibria. In all the cases the payoffs for the fuzzy Nash–Berge-Zhukovskii equilibria is higher than for the Nash equilibrium. Moreover, when both players have equal biases to Nash and Berge-Zhukovskii equilibrium the detected fuzzy N-BZ equilibrium is suitable to model the human behavior presented in [6].



(a) Strategy space (b) Payoff space

Fig. 2. The detected fuzzy N-BZ equilibrium for the SRTT game with membership degrees: $A_N(1) = A_{BZ}(1) = 0.5$ and $A_N(2) = A_{BZ}(2) = 0.5$

5.2 Partnership Game

Partnership Game considers a firm with n partners. The profit of the firm depends on the partners effort expended on a certain job. The profit function is given by $p(x) = 4(\sum_{i=1}^{n} x_i + c\prod_{i=1}^{n} x_i)$, where x_i is the amount of the expended effort by partner i. The value c measures how complementary the tasks of partners are. Each partner i incur a personal cost x_i^2 of expending effort.

All partners select the level of their effort simultaneously and independently of the other partners. Each partner seeks to maximize their share of the firm's profit which is split equally. The payoff for partner i is given by $u_i(x) = \frac{p(x)}{n} - x_i^2$.

The Partnership game is used with the following parameter settings: $n = 2$, $x_1, x_2 \in [0, 4]$ and $c = 0.2$.

The fuzzy Nash-Pareto equilibrium does not offer promising results for the Partnership game. Similarly as for the SRTT game, for any membership degree the fuzzy Nash-Pareto equilibria correspond to the Nash equilibrium.

Figures 3 and 4 depict the crisp Nash, crisp Berge-Zhukovskii and the fuzzy N-BZ equilibrium for various membership degrees. The Nash equilibrium of the game ensures a smaller payoff for the players $(4, 4)$ while the Berge-Zhukovskii equilibrium offers a more favorable outcome, a payoff of 6.04 for both players.

Figure 3 depicts the detected fuzzy N-BZ equilibrium when player 1 is a pure Nash player ($A_N(2) = 0$, $A_{BZ}(2) = 1$) and player 2 is a pure Berge-Zhukovskii player ($A_N(2) = 0$, $A_{BZ}(2) = 1$).

When both players have equal membership degrees to Nash and Berge-Zhukovskii equilibria, the fuzzy N-BZ equilibrium is the same as the crisp Nash or crisp Berge-Zhukovskii equilibria, depending on the players' biases. If the Nash-bias for both players is higher than 0.5 ($A_N(1)$, $A_N(2) > 0.5$ and $A_{BZ}(1)$, $A_{BZ}(2) < 0.5$) then the corresponding fuzzy N-BZ equilibrium coincides with the Nash equilibrium. Otherwise, if both players are biased towards the Berge-Zhukovskii equilibrium ($A_N(1)$, $A_N(2) < 0.5$ and $A_{BZ}(1)$, $A_{BZ}(2) > 0.5$), the

corresponding fuzzy N-BZ equilibrium coincides with the crisp Berge-Zhukovskii equilibrium.

Figure 3 depicts the detected fuzzy N-BZ equilibrium when both players have equal membership degrees of 0.5 to both Nash and Berge-Zhukovskii equilibrium ($A_N(1) = A_{BZ}(1) = A_N(2) = A_{BZ}(2) = 0.5$). The detected fuzzy N-BZ front is very close to the Pareto-optimal front.

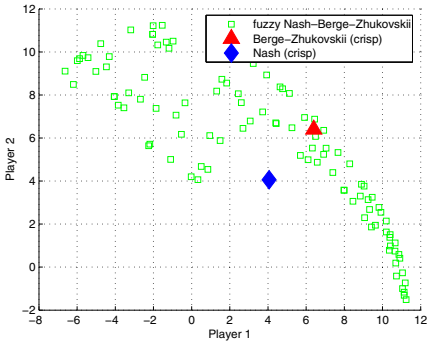


Fig. 3. The payoffs of the detected fuzzy N-BZ equilibria for the partnership game with membership degrees $A_N(1) = 1, A_{BZ}(1) = 0$ and $A_N(2) = 0, A_{BZ}(2) = 1$

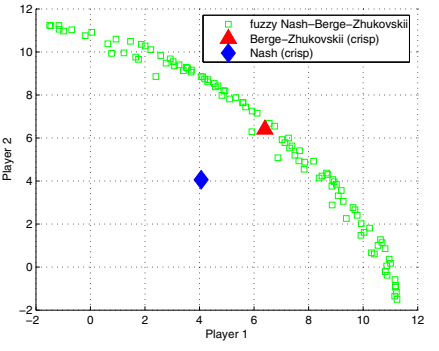


Fig. 4. The payoffs of the detected fuzzy N-BZ equilibria for the partnership game with membership degrees: $A_N(1) = A_{BZ}(1) = 0.5$ and $A_N(2) = A_{BZ}(2) = 0.5$

6 Conclusions

A new equilibrium concept, the fuzzy Nash–Berge-Zhukovskii equilibrium is proposed which is a fuzzy combination of the Nash and Berge-Zhukovskii equilibrium. Game equilibria may be described by generative relations. An evolutionary method for equilibria detection based on generative relations is considered.

Continuous trust games, the partnership game and a continuous version of the centipede game (the SRTT game), are investigated. In the case of the studied games the Nash equilibrium is mutual defection ensuring the lowest possible payoffs for all players. In contrary, the Berge-Zhukovskii equilibrium induces mutual cooperation which ensures higher payoffs.

Numerical results indicate that the proposed fuzzy Nash–Berge-Zhukovskii equilibrium offers a more realistic modeling of the real world. In case of the SRTT game fuzzy Nash–Berge-Zhukovskii equilibrium corresponds to the real life results presented in [6].

Acknowledgments. The first author wishes to thank for the financial support provided from pro- grams co-financed by the Sectoral Operational Programme Human Resources Development, Contract POSDRU/88/1.5/S/60185 Innovative Doctoral Studies in a Knowledge Based Society. The project was partially supported by CNCS-UEFISCDI (project number TE 252) and was made possible

through the support of a grant from the John Templeton Foundation. The opinions expressed in this publication are those of the authors and do not necessarily reflect the views of the John Templeton Foundation.

References

1. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II KanGAL Report No. 200001, Indian Institute of Tehnology Kanpur (2000)
2. Dumitrescu, D., Lung, R.I., Mihoc, T.D., Nagy, R.: Fuzzy Nash-Pareto Equilibrium: Concepts and Evolutionary Detection. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., Neri, F., Preuß, M., Togelius, J., Yannakakis, G.N. (eds.) EvoApplicatons 2010. LNCS, vol. 6024, pp. 71–79. Springer, Heidelberg (2010)
3. Dumitrescu, D., Lung, R.I., Nagy, R., Zaharie, D., Bartha, A., Logofătu, D.: Evolutionary Detection of New Classes of Equilibria: Application in Behavioral Games. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part II. LNCS, vol. 6239, pp. 432–441. Springer, Heidelberg (2010)
4. Gaskó, N., Dumitrescu, D., Lung, R.I.: Evolutionary detection of Berge and Nash equilibria. In: Nature Inspired Cooperative Strategies for Optimization, NICSO 2011, pp. 149–158 (2011)
5. Lung, R.I., Dumitrescu, D.: Computing Nash Equilibria by Means of Evolutionary Computation. Int. J. of Computers, Communications & Control, 364–368 (2008)
6. Murphy, R., Rapoport, A., Parco, J.: The breakdown of cooperation in iterative real-time trust dilemmas. Experimental Economics 9(2), 147–166 (2006)
7. Nash., J.F.: Non-cooperative games. Annals of Mathematics 54, 286–295 (1951)
8. Radner, R., Myerson, R., Maskin, E.: An Example of a Repeated Partnership Game with Discounting and with Uniformly Inefficient Equilibria. Review of Economic Studies 1, 59–69 (1986)
9. Rosenthal, Robert, W.: Games of perfect information, predatory pricing and the chain-store paradox. Journal of Economic Theory 25, 92–100 (1981)
10. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359 (1997)
11. Zhukovskii, V.I.: Linear Quadratic Differential Games, Naukova Doumka, Kiev (1994)

A Spanning Tree-Based Encoding of the MAX CUT Problem for Evolutionary Search

Kisung Seo[1], Soohwan Hyun[1], and Yong-Hyuk Kim[2,*]

[1] Dept. of Electronic Engineering, Seokyeong University, Seoul, Korea
ksseo@skuniv.ac.kr, xjavalov@shhyun.com
[2] Dept. of Computer Science and Engineering, Kwangwoon University, Seoul, Korea
yhdfly@kw.ac.kr

Abstract. Most of previous genetic algorithms for solving graph problems have used vertex-based encoding. In this paper, we introduce spanning tree-based encoding instead of vertex-based encoding for the well-known MAX CUT problem. We propose a new genetic algorithm based on this new type of encoding. We conducted experiments on benchmark graphs and could obtain performance improvement on sparse graphs, which appear in real-world applications such as social networks and systems biology, when the proposed methods are compared with ones using vertex-based encoding.

Keywords: Basis change, encoding, representation, genetic algorithm, MAX CUT, spanning tree, graph.

1 Introduction

In genetic algorithms (GAs), different encodings lead completely different search on solution space, and as a result, encoding can affect performance largely. There have been many studies of emphasizing the importance of encoding in GAs. Kim *et al.* [1] improved the performance of genetic algorithms on various problems by rearranging the related gene positions to be closely located. This gene rearrangement can be seen as a simply type of transformed encoding. There have also been more generalized studies of encoding transformation making the relation between genes be the most independent by applying invertible linear transformation [2, 3]. These studies just showed the importance of encoding transformation, but they failed to show the concrete transformation methods. As an extension of these studies, there has been a trial to find better encoding using a meta-GA [4]. However it also failed to give a good guideline about how we transform encoding in a given problem.

Most studies about graph problems such as graph partitioning and MAX CUT have been vertex-centric when dealing with partitions and representing them [5-14]. Intuitive techniques based on vertices, which are easy to manage, have been mainly used to solve the graph problems. However, when dealing with partitions, there have been

* Corresponding author.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 510–518, 2012.

studies [15-19] using methods based on edge, which is a dual of vertex. In particular, Armbruster *et al.* [20] and Yoon *et al.* [21] used an edge representation for solving graph partitioning, which maps a solution to an edge set, not a vertex set. In their representation, each location of an encoding is assigned to 1 if its corresponding edge is on the cut and 0 otherwise. This representation is well adapted to their integer programming formulation, but it is very crucial but difficult to check whether or not a given encoding forms a valid graph partition.

In this paper, we propose a new genetic algorithm based on not vertex-based encoding but spanning tree-based encoding [22, 23] as a kind of edge-based encoding. Contrary to general edge-based encoding, spanning tree-based encoding represents only feasible partitions. As a target problem, we adopted the MAX CUT problem, which is well known as a representative NP-hard problem, and examined the performance of the proposed genetic algorithms experimentally. The proposed method is expected to well perform on sparse graphs. In particular, if we consider that graphs appearing in real-world applications such as social networks and systems biology are sparse, the proposed method has great potential in real-world graph problems.

Section 2 discusses the MAX CUT problem and its previous work. Section 3 describes the proposed spanning tree-based encoding scheme. Section 4 presents experimental results on various graph sets, and Section 5 concludes the paper.

2 MAX CUT

It is important in combinatorial optimization to partition the vertices into two disjoint subsets of nearly equal size such that the sum of edge weights with two edge endpoints in different sets (cut size) is maximized or minimized. Given an undirected graph $G = (V, E)$ with edge weights, the MAX CUT problem (Fig. 1) is that of finding a subset $S \subset V$ which maximizes the sum of edge weights in the cut $(S, V - S)$.

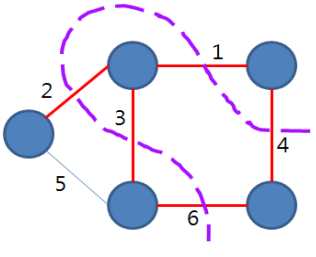


Fig. 1. Example of MAX CUT

Every graph has a finite number of cuts, so one can find the minimum or maximum weight cut in a graph by an exhaustive search that enumerates the sizes of all the cuts. This is not a practical approach for large graphs which arises in real-world applications since the number of cuts in a graph grows exponentially with the number of vertices. Although we can solve the min-cut problem without balance requirement in polynomial time using the maxflow-mincut algorithm [24], we have no such fortune

when it comes to the MAX CUT problem. There is no known way to solve the problem optimally other than by exhaustive enumeration. The MAX CUT problem is one of Karp's original NP-complete problems [25] and has been known to be NP-complete even if the problem is unweighted [26].

Since there is no algorithm that guarantees an optimal solution, a typical approach to solve such a problem is to find a ρ-approximation algorithm that delivers a solution at least ρ times the optimal value in polynomial time. Sahni and Gonzales [27] presented a 1/2-approximation algorithm for the MAX CUT problem. Their greedy approach iterates through the vertices and decides which placement (S or V - S) maximizes the cut of vertex v_i with respect to vertices v_1 to v_{i-1}. Since [27], many researchers have presented approximation algorithms for the MAX CUT problem [28-31], but little progress has been made. For more than twenty years a factor of 0.5 has been the best-known polynomial-time performance guarantee for the MAX CUT problem. An algorithm by Goemans and Williamson (GW) [32] guarantees a factor of 0.878 of the optimum. The significant improvements are due to the technique of positive semidefinite programming and randomized rounding. However, solving semidefinite programming is computationally expensive. Homer and Peinado [33] gave a parallelized version of GW. In [33], GW was improved by combining with simulated annealing (SA) [34]. Afterward, Kim et al. [35] successfully applied GAs to the MAX CUT problem. In practical, when the GA is combined with lock-gain-based local search [11], the hybrid GA could outperform GW (the best approximation algorithm) combined with SA. It is known the GAs have good performance when applied to the MAX CUT problem [35], we adopted the MAX CUT problem to test our new encoding scheme in this paper.

The MAX CUT problem has many applications in various fields, It has been observed that one of the phases (the layer assignment problem) in the design process for VLSI chips and printed circuit boards (PCB) can be reduced to the MAX CUT problem [36, 37]. One of the most famous applications of the problem comes from a classical application to statistical physics [36]. It is concerned with the exact determination of a minimal energy configuration of a spin glass under no exterior field and under a continuously varying exterior magnetic field. Poljak and Tuza [38] provided a comprehensive survey of the MAX CUT problem.

3 Encoding and Evaluation

Each solution is represented by a chromosome, which is a binary string. In this section, we consider two different types of binary encoding to represent solutions for the MAX CUT problem.

3.1 Vertex-Based Encoding

When we use vertex-based encoding in GAs, the number of genes in the chromosome equals n, which is the number of vertices in the graph. Each gene corresponds to a

vertex in the graph. A gene has value 0 if the corresponding vertex is in S, and has value 1 otherwise.

To evaluate the cut size of a solution, we should compute the number of cut edges, which is an edge whose end-vertices are in different sides. For each edge, to determine if it is a cut edge, we just check that the values of genes corresponding to its end-vertices are different, i.e., (0,1) or (1,0).

3.2 Spanning Tree-Based Encoding

If $\{V_1, V_2\}$ is a partition of V, the set $E(V_1, V_2)$ of all the edges of G crossing between V_1 and V_2 is called a *cut*. Cut space consisting of all the cuts is proven to be vector space [23]. It means that an arbitrary cut can be represented by a linear combination of basis elements of cut space.

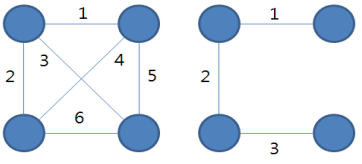


Fig. 2. Example of a graph (left) and its spanning tree (right)

In the case that the graph G is connected[1], we can derive a basis of cut space from a spanning tree of G. Finding basis of cut space based on spanning trees are known as nontrivial ones. General graph traversal algorithms such as depth-first search (DFS) and breadth-first search (BFS) can produce spanning trees of G. Let T be a spanning tree of G. For each edge e of the n - 1 edges in T, the graph T - e has exactly two components, and the set C_e of edges in G between the two components forms a cut. These n - 1 cuts are linearly independent and hence form a basis of cut space.

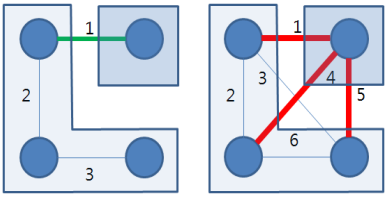


Fig. 3. Edge of spanning tree, sub-sets and cuts

When we use a spanning tree-based encoding in GAs, the number of genes in the chromosome equals n-1, the number of edges in T. Each gene corresponds to an edge in T. To evaluate the cut size of a solution, we should compute the number of cut edges. The set of cut edges is easily computed by summing C_e for each edge e in T whose gene value is 1. Then the cut size becomes the cardinality of the set of cut edges (see Fig. 3).

[1] In this paper, we assume that G is connected for convenience.

4 Simulation and Analysis

4.1 Experimental Environments

This section describes how we evaluated the proposed GA approach to develop spanning tree-based encoding for the MAX CUT problem. The GA parameters are shown in Table 1. The one-point crossover and random mutation were used for genetic recombination, and tournament selection is also adopted. The proposed algorithm was implemented using Open Beagle [39] with Boost Graph Library [40]. Three methods - Kruskal-like, DFS, and BFS - are used to find spanning trees. Every GA run is repeated 30 times for each case.

Table 1. GA Parameters

Parameters	*Values*
Max. # of generations	100
Population size	100
Crossover rate	0.9
Mutation rate	0.1
Tournament size	7

4.2 Experimental Results

The tabular results for various sets on a total of 31 graphs are provided in Table 2. The different classes of graphs that we tested our algorithms on are described below.

G*n.p*: a random graph on *n* vertices with edge probability *p*. E.g., G1000.01 is a 1,000-vertex graph with *p*=0.01.

U*n.d*: a geometric random graph on *n* vertices and expected degree *d*. E.g., U500.10 is a 500-vertex geometric graph with "expected degree" 10.

breg*n.b*: a regular random graph on *n* vertices in which each vertex has degree 3 and the optimal bisection size is *b* with high probability, i.e., probability approaches 1 as *n* approaches infinity.

cat.*n*: A caterpillar graph on *n* vertices, with each vertex having six legs. rcat.*n* is a caterpillar graph with *n* vertices, where each vertex on the spine has $\sqrt{n}$ legs.

grid*n.b*: A grid graph on *n* vertices and whose optimal minimum cut size is known to be *b*. w-grid*n.b* denotes the same grid but the boundaries are wrapped around.

Please see the reference [41] for details on how this class of graphs is generated.

Table 2. Comparison of encoding schemes for various graphs

Instances	Vertex / Edge ratio (%)	Vertex encoding		Edge encodings								
				Kruskal-like			DFS			BFS		
		Cut size	Std dev	Cut size	Std dev	Improvement(%)	Cut size	Std dev	Improvement(%)	Cut size	Std dev	Improvement(%)
G500.005	79.9%	375.0	3.7	391.3	3.0	4.35	**392.1**	**3.2**	**4.55**	388.9	3.8	3.71
G500.01	40.8%	700.1	4.5	706.1	4.6	0.87	**708.1**	**5.7**	**1.15**	707.0	4.4	1.00
G500.02	21.2%	**1303.1**	**7.2**	1293.3	7.4	-0.75	1294.7	7.9	-0.64	1295.7	7.5	-0.57
G500.04	9.7%	**2747.9**	**10.0**	2718.3	10.5	-1.08	2711.1	7.9	-1.34	2722.2	12.1	-0.94
G1000.005	40.0%	1373.4	8.4	**1383.6**	**5.4**	**0.75**	1383.5	6.9	0.74	1383.2	4.3	0.71
G1000.01	19.7%	**2711.0**	**8.5**	2699.2	7.7	-0.44	2695.9	9.7	-0.56	2702.0	7.8	-0.33
G1000.02	9.9%	**5307.4**	**10.9**	5271.3	13.6	-0.68	5265.6	15.0	-0.79	5280.5	15.2	-0.51
U500.05	38.9%	597.2	3.6	**606.3**	**3.8**	**1.52**	605.0	3.7	1.31	606.0	3.4	1.47
U500.10	20.3%	1276.1	4.8	1282.3	4.5	0.49	1279.3	4.0	0.25	**1283.8**	**6.5**	**0.61**
U500.20	11.0%	**2410.2**	**5.7**	2407.4	5.4	-0.11	2396.0	7.6	-0.59	2402.1	7.5	-0.34
U500.40	5.7%	**4575.6**	**7.1**	4563.2	8.2	-0.27	4547.1	6.6	-0.62	4557.5	7.2	-0.40
U1000.05	41.7%	1309.0	7.6	**1324.9**	**4.6**	**1.21**	1324.7	5.3	1.20	1323.1	5.0	1.08
U1000.20	10.7%	**4877.9**	**10.4**	4871.1	10.6	-0.14	4851.2	12.1	-0.55	4861.2	12.5	-0.34
U1000.40	5.5%	**9292.5**	**12.2**	9261.0	8.8	-0.34	9234.2	12.2	-0.63	9252.2	15.6	-0.43
breg500.12	66.5%	445.4	4.8	460.8	3.6	3.47	**461.8**	**3.3**	**3.67**	459.9	3.7	3.25
breg500.16	66.5%	444.5	3.4	**462.5**	**4.7**	**4.04**	462.3	3.8	4.01	460.0	3.5	3.49
breg500.20	66.5%	444.9	4.6	460.8	3.1	3.57	460.7	3.2	3.55	**461.2**	**4.3**	**3.66**
cat.352	100.0%	224.1	2.9	242.1	2.1	8.05	241.5	2.5	7.80	**242.7**	**2.8**	**8.33**
cat.702	100.0%	419.0	3.3	444.6	3.5	6.10	**446.2**	**3.9**	**6.49**	445.8	3.0	6.40
cat.1052	100.0%	606.8	5.0	**644.9**	**5.6**	**6.28**	**644.9**	**5.6**	**6.28**	**644.9**	**5.7**	**6.28**
cat.5252	100.0%	2804.5	10.0	2890.8	10.0	3.07	2885.3	10.7	2.88	**2891.2**	**10.2**	**3.09**
rcat.134	100.0%	99.4	1.5	**106.1**	**1.5**	**6.74**	**106.1**	**1.5**	**6.74**	**106.1**	**1.6**	**6.74**
rcat.554	100.0%	339.0	3.1	**360.2**	**2.5**	**6.26**	**360.2**	**2.5**	**6.26**	**360.2**	**2.6**	**6.26**
rcat.994	100.0%	577.9	5.2	**611.3**	**4.7**	**5.78**	**611.3**	**4.7**	**5.78**	**611.3**	**4.8**	**5.78**
rcat.5114	100.0%	2736.6	10.3	2817.4	10.0	2.95	**2817.6**	**9.7**	**2.96**	2816.1	9.5	2.91
grid100.10	55.2%	130.1	2.4	133.2	2.0	2.36	132.9	2.4	2.10	**133.7**	**1.8**	**2.77**
grid500.21	52.3%	556.9	4.3	569.4	4.3	2.24	**571.7**	**6.0**	**2.66**	570.2	5.5	2.38
grid1000.20	51.8%	1075.5	5.6	1094.6	7.2	1.77	**1096.3**	**5.1**	**1.93**	1095.9	7.7	1.89
w-grid100.20	49.5%	145.1	2.6	**146.8**	**1.9**	**1.22**	145.2	2.7	0.07	144.3	2.4	-0.55
w-grid500.42	49.9%	582.1	4.6	594.0	5.1	2.04	**594.5**	**5.4**	**2.12**	591.3	3.9	1.58
w-grid1000.40	50.0%	1113.8	9.0	**1133.2**	**7.0**	**1.74**	1130.2	5.1	1.47	1129.5	5.8	1.40

The ratio of vertices to edges in sparse graphs is greater than in dense graphs. For connected graphs, if n is the number of vertices, then the number of edges is between n-1 and $n(n-1)/2$. Therefore, the ratio of vertices to edges has a value between $2/(n-1)$ to $n/(n-1)$.

For the case of sparse graphs in which the ratio of vertices to edges is more than 38.9%, the performance of spanning tree-based encoding is superior to the results of vertex-based encoding. The improvement increases for greater ratios of vertices to edges, in general. Examples include breg*n*.*b*, grid*n*.*b*, cat.*n*, rcat.*n*, and w-grid*n*.*b*. The cat.*n* and rcat.*n* graph sets with 100% ratio of vertices to edges, show the average of approximately 6% improvement using spanning tree-based encoding.

On the other hand, the performance of vertex-based encoding is better than that of spanning tree-based encoding for dense graphs in which ratio of vertices to edges is less than 20%. Some G*n.p* and U*n.d* graph sets are examples of this category.

The efficiencies for graphs around with ratios of 20% are irregular. For example, the performance of spanning tree-based encoding is superior in U500.10, which has a 20.3% ratio, but the performance of vertex-based encoding is better in G500.02, which has a 21.2% ratio. Geometric random graphs are closer to real-world problems than random graphs, considering that the randomness of geometric random graphs is less than that of random graphs and the vertices connected with edges in geometric random graphs are locally clustered.

Therefore the superiority of spanning tree-based encoding will be expected for real-world problems which consist of sparse graphs having more than a 20% ratio of vertices to edges. In the paper, three methods - Kruskal-like, DFS, and BFS - are used to obtain spanning trees and the results differ slightly for each method, but they are not very different. The topic of what kind of algorithms for finding a spanning tree is efficient and how these algorithms influence the performance appears to be one of interests.

5 Conclusions

We proposed a new encoding method and investigated its performance comparing to a widely-used method for the MAX CUT problem. This study is the first trial of applying spanning tree-based encoding to optimization method for graph problems. To demonstrate the effectiveness of our proposed approach, experiments on three spanning tree-based encodings were conducted for benchmark graphs and could obtain performance improvement on sparse graphs.

We also found that the change of encoding method can make differences for optimization performance. In other words, edge-based encoding is advantageous for sparse graphs and vertex-based encoding is profitable for dense graphs.

The proposed approach can be applied to other graph partitioning problems, e.g., ratio-cut graph partitioning, which are similar to the MAX CUT problem. In particular, the proposed spanning tree-based encoding scheme has a merit on partitioning of sparse graphs, which appear widely in real-world applications. Further study will aim at the extension and refinement of the encoding schemes and their application to various graph sets.

Acknowledgments. This work was supported by National Research Foundation of Korea Grant funded by the Korea government (NRF-2011-0009958 and NRF-2009-0071419).

References

1. Kim, Y.-H., Kwon, Y.-K., Moon, B.-R.: Problem-independent Schema Synthesis for Genetic Algorithms. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 1112–1122. Springer, Heidelberg (2003)

2. Kim, Y.-H.: Linear transformation in pseudo-Boolean functions. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1117–1118 (2008)
3. Kim, Y.-H., Yoon, Y.: Effect of changing the basis in genetic algorithms using binary encoding. KSII Transactions on Internet and Information Systems 2(4), 184–193 (2008)
4. Kim, Y.-H., Yoon, Y.: Representation and recombination over nonsingular binary matrices. In: Proceedings of the World Summit on Genetic and Evolutionary Computation, pp. 855–858 (2009)
5. Alpert, C.J., Kahng, A.B.: Recent directions in netlist partitioning: a survey. Integration, the VLSI Journal 19(1-2), 1–81 (1995)
6. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Information Processing Letters 42(3), 153–159 (1992)
7. Clark, L.H., Shahrokhi, F., Székely, L.A.: A linear time algorithm for graph partition problems. Information Processing Letters 42(1), 19–24 (1992)
8. Feige, U., Karpinski, M., Langberg, M.: A note on approximating Max-Bisection on regular graphs. Information Processing Letters 79(4), 181–188 (2001)
9. Fjällström, P.O.: Algorithms for graph partitioning: a survey. Linkoping Electronic Articles in Computer and Information Science 3 (1998)
10. Hagen, L., Kahng, A.B.: New spectral methods for ratio cut partitioning and clustering. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 11(9), 1074–1085 (1992)
11. Kim, Y.-H., Moon, B.-R.: Lock-gain based graph partitioning. Journal of Heuristics 10(1), 37–57 (2004)
12. Kučera, L.: Expected complexity of graph partitioning problems. Discrete Applied Mathematics 57(2-3), 193–212 (1995)
13. Powers, D.L.: Graph partitioning by eigenvectors. Linear Algebra and its Applications 101, 121–133 (1988)
14. Yan, J.-T., Hsiao, P.-Y.: A fuzzy clustering algorithm for graph bisection. Information Processing Letters 52(5), 259–263 (1994)
15. Antonio, S.M., Abraham, D., Juan, J.P., Raúl, C.: High-performance VNS for the max-cut problem using commodity graphics hardware. In: Proceedings of the 18th Mini Euro Conference on VNS (2005)
16. Cong, J., Labio, W.J., Shivakumar, N.: Multiway VLSI circuit partitioning based on dual net representation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15(4), 396–409 (1996)
17. Guattery, S., Miller, G.L.: On the quality of spectral separators. SIAM Journal on Matrix Analysis and Applications 19(3), 701–719 (1998)
18. Michel, J., Pellegrini, F., Roman, J.: Unstructured Graph Partitioning for Sparse Linear System Solving. In: Lüling, R., Bilardi, G., Ferreira, A., Rolim, J.D.P. (eds.) IRREGULAR 1997. LNCS, vol. 1253, pp. 273–286. Springer, Heidelberg (1997)
19. Venkatakrishnan, V.: Parallel computation of Ax and A^Tx. International Journal of High Speed Computing 6, 325–342 (1994)
20. Armbruster, M., Fügenschuh, M., Helmberg, C., Jetchev, N., Martin, A.: Hybrid Genetic Algorithm Within Branch-and-Cut for the Minimum Graph Bisection Problem. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 1–12. Springer, Heidelberg (2006)
21. Yoon, Y., Kim, Y.-H., Moon, B.-R.: A note on edge-based graph partitioning and its linear algebraic structure. Journal of Mathematical Modelling and Algorithms 10(3), 269–276 (2011)
22. Biggs, N.: Algebraic Graph Theory, 2nd edn. Cambridge University Press (1994)

23. Diestel, R.: Graph Theory, 3rd edn. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (2005)
24. Ford Jr., L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press (1962)
25. Karp, R.M.: Reducibility Among Combinatorial Problems, pp. 85–103. Plenum Press, New York (1972)
26. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. Theoretical Computer Science 1(3), 237–267 (1976)
27. Sahni, S., Gonzalez, T.: P-complete approximation problems. Journal of the ACM 23(3), 555–565 (1976)
28. Vitányi, P.M.B.: How well can a graph be n-colored? Discrete Mathematics 34(1), 69–80 (1981)
29. Poljak, S., Tuza, Z.: A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem. Canadian Journal of Mathematics 34, 519–524 (1982)
30. Haglin, D.J., Venkatesan, S.M.: Approximation and intractability results for the maximum cut problem and its variants. IEEE Trans. on Computer 40, 110–113 (1991)
31. Hofmeister, T., Lefmann, H.: A combinatorial design approach to maxcut. In: Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (1995)
32. Goemans, M.X., Williamson, D.P.: Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. Journal of the Association for Computing Machinery 42(6), 1115–1145 (1995)
33. Homer, S., Peinado, M.: Design and Performance of Parallel and Distributed Approximation Algorithms for Maxcut. Journal of Parallel and Distributed Computing 46, 48–61 (1997)
34. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
35. Kim, S.-H., Kim, Y.-H., Moon, B.-R.: A hybrid genetic algorithm for the MAX CUT problem. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 416–423 (2001)
36. Barahona, F., Grotschel, M., Junger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Operational Research 36, 493–513 (1984)
37. Pinter, R.Y.: Optimal layer assignment for interconnect. Journal of VLSI Computing Systems 1, 123–137 (1984)
38. Poljak, S., Tuza, Z.: Maximum cuts and largest bipartite subgraphs, vol. 20. American Mathematical Society (1993)
39. Boost Library, `http://boost.org`
40. Open Beagle, `http://beagle.gel.ulaval.ca`
41. Bui, T.N., Moon, B.R.: Genetic algorithm and graph partitioning. IEEE Transactions on Computers 45(7), 841–855 (1996)

A Hybrid Approach to Piecewise Modelling of Biochemical Systems

Zujian Wu, Shengxiang Yang, and David Gilbert

School of Information Systems, Computing and Mathematics
Brunel University, Uxbridge, Middlesex UB8 3PH, UK
{zujian.wu,shengxiang.yang,david.gilbert}@brunel.ac.uk

Abstract. Modelling biochemical systems has received considerable attention over the last decade from scientists and engineers across a number of fields, including biochemistry, computer science, and mathematics. Due to the complexity of biochemical systems, it is natural to construct models of the biochemical systems incrementally in a piecewise manner. This paper proposes a hybrid approach which applies an evolutionary algorithm to select and compose pre-defined building blocks from a library of atomic models, mutating their products, thus generating complex systems in terms of topology, and employs a global optimization algorithm to fit the kinetic rates. Experiments using two signalling pathways show that given target behaviours it is feasible to explore the model space by this hybrid approach, generating a set of synthetic models with alternative structures and similar behaviours to the desired ones.

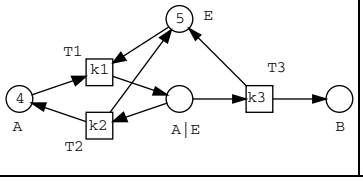
1 Introduction

Models of biochemical systems can be used in systems biology to predict and explain behaviour, or as templates for designing novel biological systems in synthetic biology. It is still an open question regarding how to build and verify models of biochemical systems, involving intelligent methods and tractable computational tools. Traditionally the structures of models are inferred from various experimental observations, and the kinetic rates are estimated computationally by considering kinetic laws [3,9].

Much previous research has focused on how to fit the kinetic rates of an existing biochemical model so that its behaviour coincides with the observations of a given physical system [7,14,13]. However, another research line is to identify alternative topologies and optimize the topologies [8,20]. Moreover, a model of a biochemical system can be engineered by modifying and piecewise constructing its network topology, using biological building blocks. As the kinetic rates (parameters) associated with biochemical reactions (forming the structure) are crucial for biochemical systems exhibiting observed behaviours, it is necessary to model the systems in terms of both the topology and kinetic rates by a hybrid method. The challenging aim of our research is the development of a robust method for the automated construction of models from descriptions of the

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 519–528, 2012.

Table 1. An enzymatic reaction and its components

Enzymatic Reaction	Petri net	Components
$A + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} A\|E \xrightarrow{k_3} B + E$ $[A] = 4$ $[E] = 5$ $[A\|E] = [B] = 0$		$A + E \xrightarrow{k_1} A\|E$ $A\|E \xrightarrow{k_2} A + E$ $A\|E \xrightarrow{k_3} B + E$

observed or desired behaviours of the biochemical systems, by the manipulation of both the topology and kinetic rates.

Some recent research applying evolutionary methodologies to model biological systems can be found in [18,19,2]. Evolutionary computation and functional Petri nets have been applied to infer metabolic pathways by Kitagawa and Iba [10]; however their approach relies on starting with an existing network model which is then modified, whereas our approach is to incrementally piecewise construct a network from a single node. Previously [21] we have developed a method to piecewise construct the topology of networks using simulated annealing (SA); in the research reported here we use a hybrid approach which employs evolution strategy (ES) to derive the topology and SA for the kinetic rates.

2 Components and Composition Rules

2.1 Pre-defined Components

Components are defined according to the semantics and syntax of the biological building blocks in [21]. There are two patterns for generating the reusable components in a library: (1) binding pattern $P_1 + P_2 \xrightarrow{k_i} P_3$; (2) unbinding pattern $P_3 \xrightarrow{k_j} P_1 + P_2$. The parameters k_i and k_j are the kinetic rates of binding and unbinding reactions, and usually $k_i \gg k_j$. The two patterns illustrate how a complex can be synthesized from substrates or broken down into substrates. A basic enzymatic reaction can be represented by one instantiation of the binding pattern and two instantiations of the unbinding pattern, as shown in Table 1. The concentrations of substrates in the enzymatic reaction are indicated with labels within square brackets, such as '[A]' and '[A|E]', where the symbol '|' means that the biochemical complex 'A|E' is made from two substrates A and E.

2.2 Composition Rules

Given an existing biochemical network model *BioN* and a library of biological components *CompLib*, the operation of piecewise composition can be the addition of one component C_a from *CompLib* to *BioN*, or the subtraction of one component C_s from *BioN*. We have further developed the original composition

rules proposed in [21] to permit component subtraction and greater flexibility in composition. The rules developed are performed by comparing and replacing parts of the labels of the added component. In this paper, L_i $(i = 1, 2, 3)$ is the label of places P_i from the added component C_a, and L_B is the label of a place P_B of a component C_B in *BioN*. The details of composition rules are as follows.

1. Given a binding component $P_1 + P_2 \xrightarrow{k1} P_3$ or an unbinding component $P_3 \xrightarrow{k2} P_1 + P_2$, where L_1, L_2 and L_3 are labels of P_1, P_2 and P_3, respectively.
   (a) If $L_B = L_1$ or $L_B = L_2$ or $L_B = L_1|L_2$, the component C_a is added to the existing network *BioN* by adding its reaction equations directly;
   (b) If $L_B \neq L_1$ or $L_B \neq L_2$, all L_1 (L_2) in C_a are replaced by L_B in C_a and the modified reaction equations are added to *BioN*;
   (c) If $L_B \neq L_3$ and P_B is a complex, L_3 in C_a is replaced by L_B, L_1 is replaced by L_{B1}, and L_2 is replaced by L_{B2} where $\{L_{B1}, L_{B2}|L_{B1} \cap L_{B2} = 0$ and $L_{B1} \cup L_{B2} = L_B\}$. The corresponding modified reaction equations of C_a are added to *BioN*;
   (d) If $L_B \neq L_3$ and P_B is not a complex, the reaction equations of C_a are added into *BioN*, and a new component C'_a is created by binding P_3 with P_B to produce $P_B|P_3$ $(P_3 + P_B \xrightarrow{k1} P_B|P_3)$, and reaction equations of C'_a are added to *BioN*.
2. A component C_s is selected randomly from *BioN* for subtraction.
   (a) If C_s is the only component in *BioN*, no subtraction is applied to *BioN*;
   (b) If C_s is not the only component in *BioN*, the transition and its incident arcs in C_s are removed directly. The *BioN* is checked for connectivity. Non-connected parts of *BioN* are linked by creating a binding component with species selected randomly from the non-connected parts.

3 Hybrid Piecewise Modelling

Current research has focused on generating topologies [16] and fitting kinetic rates [17], or both [5]. In this paper, we aim to solve a topology optimization problem by iteratively piecewise assembling components represented by quantitative Petri nets from a user pre-defined library, combined with optimizing the kinetic rates.

A hybrid evolutionary and heuristic approach has been developed using a two layer framework: firstly, this hybrid approach evolves the topology of the model representing the target system by performing ES at the outer layer, and then SA is applied at the inner layer to optimize the kinetic rates of the evolved model. The piecewise modelling stops after a pre-defined number of generations and returns a set of the best synthetic models, offering alternative topologies with similar behaviours to the target system. The pseudo-code of the hybrid framework between ES and SA is shown in Algorithm 1. The details of evolving the topology by the ES layer and optimizing kinetic rates by the SA layer are described in Section 3.1 and Section 3.2, respectively.

Algorithm 1. A hybrid piecewise modelling framework

Require: CompLib, Composition Rules
Ensure: $BioN_{best}$
Initiate the population;
while Not reached maximum generation (ES layer) **do**
  for Each individual in the population **do**
    Mutate the topology of individual by Addition or Subtraction;
    Check the mutated topology of the individual;
    Evaluate the mutated individual;
    if The kinetic rates are required to be optimized **then**
      while Not reach minimum temperature (SA layer) **do**
        Optimize the kinetic rates of individual by Gaussian distribution;
        Evaluate the mutated kinetic rates;
      end while
    end if
  end for
  Crossover the individuals;
  Select offspring for next generation;
end while
Return $BioN_{best}$

3.1 Evolution Strategy Based Topology Optimization

The (μ+λ)-ES is utilized to iteratively piecewise assemble the components for the model construction, where μ and λ are the number of parents and children respectively. The (μ+λ)-ES starts from an initial population of individuals and each individual is a single component selected randomly from the library. The individuals are mutated by genetic operators adapted from evolutionary algorithms: *Addition* ($\oplus$), *Subtraction* ($\ominus$) and *Crossover* ($\otimes$). The individuals with the best fitness are selected to generate offspring for the next generation.

The three genetic operators are concepts taken from genetic algorithms, and the implementation of these operators in this paper is inspired by nature. The addition operator is used to integrate a component to an existing topology of model. The subtraction operator is used to remove the transition with incident arcs in a component selected randomly from the model for a removal. The crossover operator is used to apply a 'cut and splice' method to reproduce offspring from two models under construction. The set of composition rules has been introduced in Section 2.2 for the components composition carried out by the three genetic operators.

3.2 Simulated Annealing Based Kinetic Rates Fitting

SA is a heuristic optimization algorithm for searching a global optimum solution in a very large solutions space, avoiding local optimum solutions. In our previous work [21] we have applied the SA to piecewise construct and explore the topologies of the biological systems. In this paper, the SA layer is integrated

within the ES layer to estimate the kinetic rates of the synthetic models. The topologies of these models are fixed while in the SA layer, having been passed down from the ES based outer layer after mutating their structures.

The rates of reactions in a given model are coded as follows: a vector $K(M) = (k_1^t, k_2^t, ..., k_l^t)$ is used to record the rate values in a model, where l is the number of reactions, t is the current cooling temperature, and k_i^t is a constant rate of the ith chemical reaction r_i $(i = 1, 2, ..., l)$. The vector $K(M)$ is mutated by the *Gaussian* distribution $N(\mu, \sigma)$ by N iteration times at each cooling temperature. The mutated $K(M)$ of the model is evaluated after each iteration, by comparing the behaviour of the synthetic model and the target system.

Due to the probabilistic behaviour of the random procedure of SA [1], a mutated vector $K(M)$ could be generated which causes a bad estimated fitness of the model. This is because there is a chance that the model with a fixed topology and optimized kinetic rates returned from the SA layer to the ES layer could be worse than the one passed into the SA layer.

3.3 Model Evaluation

A synthetic model is evaluated by comparing its behaviours with target behaviours of a biochemical system. The behaviours are represented by time series data of the concentrations of species, e.g. enzymes, other proteins, and complexes. The behaviours of the species in the target system can be obtained from a reference model or by observations of a biochemical system from the wet-lab.

Given a set of reference data for the target system M_T, there are N generated time series $X_T = (X_1, X_2, ..., X_N)$ which represent the behaviours of N species, $N \geq 1$. There are P data points in each time series $X_i = (x_i^1, x_i^2, ..., x_i^P)^T$, $i = 1, ..., N$. There are M time series $X_G = (\hat{X}_1, \hat{X}_2, ..., \hat{X}_M)$ describing the behaviours of M species in a constructed model M_G, with P data points for each time series $\hat{X}_j = (\hat{x}_j^1, \hat{x}_j^2, ..., \hat{x}_j^P)^T$, $j = 1, ..., M$. The intersection between M_T and M_G of species is defined by $X_C = X_T \cap X_G = (X_1, X_2, ..., X_n)$, $1 \leq n \leq N$. The difference between the behaviours of M_T and M_G is calculated by averaging the difference of behaviours of each species in X_C by a paired comparison of the P data points. As shown in Eq. (1), the difference of behaviours for one species $X_k \in X_C$ is measured by the Euclidean distance function, where η is the total number of compared substrates in X_C:

$$d_{M_T, M_G}(X_k) = \frac{1}{\eta} \sum_{k=1}^{\eta} \sqrt{\sum_{t=1}^{P} (x_k^t - \hat{x}_k^t)^2}. \tag{1}$$

While evaluating the generated model, the species for behaviour comparison can be specified by the user and are stored in X'_C ($|X'_C| = n'$). In this scenario, there could be some synthetic substrates in M_G which do not exist in M_T. Therefore, if a substrate is specified for comparison in M_G but does not exist in M_T, then M_G should be punished. If a species for comparison exists both in M_T and M_G, a reward can be given to M_G. A *Reward and Penalty* function $\Phi(X_k)$ is used

to improve the objective function as a complement of the Euclidean distance function: $\Phi(X_k) = -\varepsilon_1$ if $X_k \in X_G \wedge X_k \notin X_T$, where ε_1 is a non-negative real value for the reward; $\Phi(X_k) = \varepsilon_2$ if $X_k \in X_G \wedge X_k \in X_T$, where ε_2 is a non-negative real value for the punishment. The reward and penalty can be defined by the user at the initial stage. The return result of $\Phi(X_k)$ will partly contribute to the fitness evaluation of a generated model M_G by an objective function $f(M_G)$ in Eq. (2):

$$f(M_G) = d_{M_T,M_G}(X_k) + \frac{1}{\eta}\sum_{k=1}^{\eta}\Phi(X_k) \tag{2}$$

where $\eta = n$ if the compared substrates are from the intersection X_C, and $\eta = n'$ if the compared substrates are from the specific X'_C. In this paper, modelling is a minimization problem, therefore the smaller the fitness value, the better the generated model.

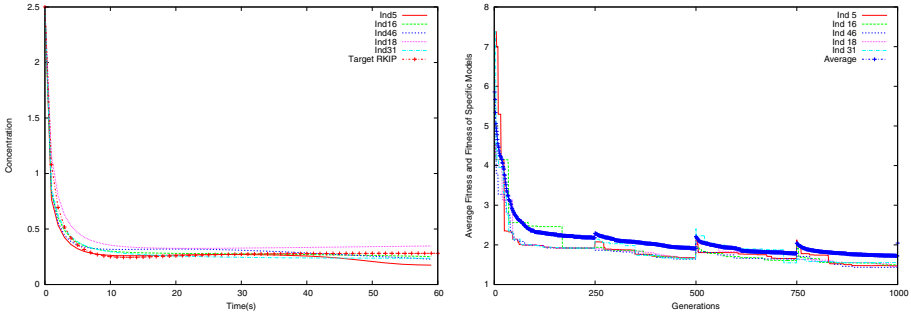
4 Experimental Study

In this section, we present simulation results for the implementation of the hybrid modelling approach on two signalling pathways: (1) the RKIP pathway, which is a mathematical model taken from Cho et al [6] for representing the fragment of the mitogen-activated protein kinase (MAPK) signal transduction pathway concerned with the inhibition of the extracellular signal regulated kinase (ERK) by the Raf1 kinase inhibitor protein (RKIP); (2) the Levchenko pathway [11] for quantitatively analyzing the signal propagation regulated by the formation of scaffold kinase complexes in the core MAPK cascade. ERK is one of the MAP Kinases (mitogen activated protein kinases), and can also be referred to as 'MAPK'; it is a player in both the Cho et al model of the RKIP fragment of the MAPK cascade as well as in the Levchenko model of the MAPK cascade.

4.1 Generation of Similar Behaviours

The main aim of our approach is to construct models with similar behaviours to the target biochemical systems. The RKIP pathway transfers the mitogenic signals from the cell membrane to the nucleus. The hypothesis is that RKIP can inhibit activation of Raf1 by binding to it, disrupting the interaction between Raf1 and MEK, thus playing a part in regulating the activity of the ERK.

Figure 1a shows that the behaviours of substrates in the generated models are similar to the target behaviour in terms of Euclidean distance. Because the behaviours of RKIP in the 50 synthetic models are similar both to each other and also to the target behaviour, we only illustrate the behaviours of RKIP from the five best generated models (obtained in a single run). The construction of the models can be driven to approach to that of the target pathway by increasing the fitness in terms of reducing the Euclidean distance between behaviours employed to evaluate the models. As shown in Fig. 1b, the fitness of each model converges



(a) Behaviours of RKIP (b) Fitness of synthetic models

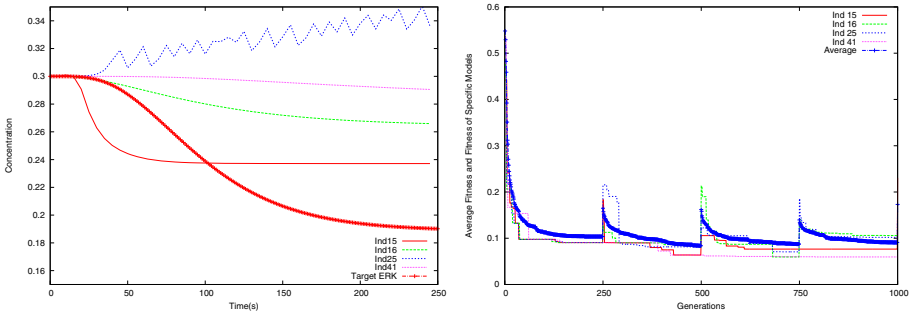
Fig. 1. (a) Behaviours of the RKIP from five best synthetic models and target RKIP pathway; (b) Average fitness of all 50 synthetic models of RKIP pathway, and fitness of the five best synthetic models.

to a minimum value with the increased number of generations in the simulation. In our current implementation, the hybrid modelling process is set to call the SA layer to optimize the kinetic rates of each model at every 250 generations; different settings are under study in ongoing research. Due to the probabilistic mechanism of accepting a worse solution by the SA, there is a jump of fitness convergence for most models. These fitness values converge again after move back to the ES layer, following a traditional evolutionary process, see Fig. 1b.

Our results for the Levchenko pathway given in Figure 2a show that models can be generated with unexpected behaviours regarding ERK which are similar in terms of Euclidean distance to the target in the MAPK cascade [11]. Although the target system does not exhibit oscillations, there is an oscillating substrate behaviour from one of the synthetic models, as supported by Kholodenko's model [12]. This suggests that feedbacks could exist in solution space, and are indeed incorporated in many MAPK models, e.g. [4], although missing in our target Levchenko model. Again, the fitness of constructed models converges with the increased number of generations in the simulation as shown in Fig. 2b.

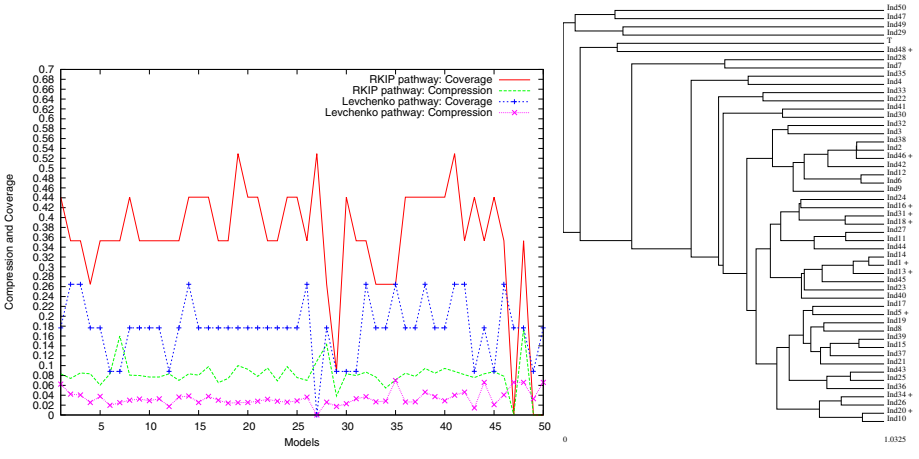
4.2 Exploration of Alternative Topologies

One of main aims of modelling biochemical systems is to explore alternative topologies, for understanding the relationships among the compounds in wet-lab. Our approach can search the model space and suggest a set of alternative topologies with similar behaviours to the target. The results can be analysed in terms of the structural difference between models, using *Compression* and *Coverage* measures. *Compression* (adapted from [21]) is a metric which computes the distance between two networks in terms of the proportion of matched (common) arcs (between the generated and target model) with respect to the maximum number of arcs in the generated or target model. *Coverage* computes inclusion in terms of the ratio of arcs in the target model which are matched in the generated



(a) Behaviours of ERK (b) Fitness of synthetic models

Fig. 2. (a) Behaviours of ERK from four best and interesting synthetic models and target Levchenko pathway; (b) Average fitness of all 50 synthetic models of Levchenko pathway, and fitness of four best and interesting synthetic models in terms of ERK behaviours.



(a) RKIP and Levchenko pathways (b) Clusters

Fig. 3. (a) Compression and coverage of RKIP and Levchenko pathways; (b) A clustering of 50 synthetic models and target RKIP pathway (T).

model. Both measures vary from 0 (worst) to 1 (best). If either compression or coverage is low for a particular model, then its topology is very different to the target, even if their behaviours are similar.

Figure 3a illustrates the compression and coverage of two signalling pathways. Most coverage of synthetic models of RKIP and Levchenko pathways is in the ranges of [0, 0.53] and [0, 0.27], respectively. Compression for both the RKIP and the Levchenko generated models is very poor, ranging over [0, 0.18], indicating that the generated models are very different to the target ones in terms of topologies. Figure 3b is a dendrogram of hierarchical pairwise clustering based on similarity and complete linkage over compression among 50 generated models

and the target RKIP pathway, and illustrates the generation of a wide range of alternative topologies by our hybrid approach; the closest 10 models in terms of fitness are shown with a '+'. Although none of the generated topologies are close to the target one, the nearest being individual 48 which is 10th closest regarding fitness, there are 9 other models which are closer in terms of behaviour despite being poorly related to the target structurally, and are also fairly widely scattered over structural space. Thus our approach is able to search model space for networks which have similar behaviours to the target, even though they may differ quite significantly in terms of structure.

5 Conclusions and Future Work

Our study addresses the evolution of quantitative Petri nets and could thus be applied to stochastic and hybrid Petri nets as well as the continuous Petri nets, which can benefit mathematical modelling. We have applied the proposed approach to two signalling pathways. The experiments show that it is feasible to iteratively piecewise model biochemical systems using our hybrid approach and explore the solution space of alternative models with different topologies but similar behaviours to the target ones.

One important issue to be investigated in our future research is to study the switching policy between ES and SA layers, in order to obtain models with good quality in terms of both topology and kinetic rates. Furthermore, implementation of the genetic operators can result in different model sizes, and thus one of our future aims is to exploit the potential tradeoff of the combinatorial application of the genetic operators. More biological constraints will be considered for defining the components and the composition rules, thus improving the biological relevance of the synthetic models. Finally, the generated models can be used as design templates to guide the construction of synthetic biological systems which may have quite different topologies from existing natural systems.

Acknowledgements ZW is partially supported by a grant from the China Scholarship Council and a grant from Brunel University. We would like to thank Crina Grosan for her helpful comments on the text.

References

1. Anily, S., Federgruen, A.: Simulated annealing methods with general acceptance probabilities. J. Appl. Prob. 24(3), 657–667 (1987)
2. Balsa-Canto, E., Banga, J.R., Egea, J.A., Fernandez-Villaverde, A., de Hijas-Liste, G.M.: Global optimization in systems biology: stochastic methods and their applications. In: Goryanin, I.I., Goryachev, A.B. (eds.) Advances in Systems Biology, Adv. Exp. Med. Biol., vol. 736, pp. 409–424 (2012)
3. Breitling, R., Gilbert, D., Heiner, M., Orton, R.: A structured approach for the engineering of biochemical network models, illustrated for signalling pathways. Brief Bioinform. 9(5), 404–422 (2008)

4. Brightman, F.A., Fell, D.A.: Differential feedback regulation of the MAPK cascade underlies the quantitative differences in EGF and NGF signalling in PC12 cells. FEBS Letters 482(3), 169–174 (2000)
5. Cao, H., Romero-Campero, F., Heeb, S., Camara, M., Krasnogor, N.: Evolving cell models for systems and synthetic biology. Syst. Synth. Biol. 4(1), 55–84 (2010)
6. Cho, K.H., Shin, S.Y., Kim, H.W., Wolkenhauer, O., Mcferran, B., Kolch, W.: Mathematical Modeling of the Influence of RKIP on the ERK Signaling Pathway. In: Priami, C. (ed.) CMSB 2003. LNCS, vol. 2602, pp. 127–141. Springer, Heidelberg (2003)
7. Feng, X.J., Hooshangi, S., Chen, D., Li, G., Weiss, R., Rabitz, H.: Optimizing genetic circuits by global sensitivity analysis. Biophys 87(4), 2195–2202 (2004)
8. Francois, P., Hakim, V.: Design of genetic networks with specified functions by evolution in silico. PNAS 101(2), 580–585 (2004)
9. Gilbert, D., Breitling, R., Heiner, M., Donaldson, R.: An Introduction to BioModel Engineering, Illustrated for Signal Transduction Pathways. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 13–28. Springer, Heidelberg (2009)
10. Kitagawa, J., Iba, H.: Identifying metabolic pathways and gene regulation networks with evolutionary algorithms. In: Fogel, G.B., Corne, D.W. (eds.) Evolutionary Computation in Bioinformatics, pp. 255–278 (2003)
11. Levchenko, A., Bruck, J., Sternberg, P.W.: Scaffold proteins biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. Proc. of the National Academy of Sciences of the United States of America 97(11), 5818–5823 (2000)
12. Kholodenko, B.N.: Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. Eur. J. Biochem. 267, 1583–1588 (2000)
13. Manca, V., Marchetti, L.: Log-Gain stoichiometric stepwise regression for MP systems. J. Found. Comput. Sci. 22(1), 97–106 (2011)
14. Maria, G.: A review of algorithms and trends in kinetic model identification for chemical and biochemical systems. Chem. Biochem. Eng. Q. 18(3), 195–222 (2004)
15. Murata, T.: Petri Nets: properties, analysis and applications. Proc. of the IEEE 77(4), 541–580 (1989)
16. Rodrigo, G., Carrera, J., Jaramillo, A.: Genetdes: automatic design of transcriptional networks. Bioinformatics 23(14), 1857–1858 (2007)
17. Schulz, M., Bakker, B.M., Klipp, E.: TIde: a software for the systematic scanning of drug targets in kinetic network models. BMC Bioinformatics 10(1), 344–353 (2009)
18. Sendin, J.O.H., Exler, O., Banga, J.R.: Multi-objective mixed integer strategy for the optimisation of biological networks. Systems Biology, IET 4(3), 236–248 (2010)
19. Sun, J., Garibaldi, J.M., Hodgman, C.: Parameter estimation using metaheuristics in systems biology: a comprehensive review. IEEE/ACM Trans. Comput. Biol. Bioinformatics 9(1), 185–202 (2012)
20. Vyshemirsky, V., Girolami, M.: Bayesian ranking of biochemical system models. BMC Bioinformatics 24(6), 833–839 (2008)
21. Wu, Z., Gao, Q., Gilbert, D.: Target driven biochemical network reconstruction based on petri nets and simulated annealing. In: Quaglia, P. (ed.) CMSB 2010, pp. 33–42. ACM (2010)

An Empirical Comparison of CMA-ES in Dynamic Environments

Chun-Kit Au and Ho-Fung Leung

Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin, Hong Kong
{auchunkit,lhf}@cuhk.edu.hk

Abstract. This paper empirically investigates the behavior of three variants of covariance matrix adaptation evolution strategies (CMA-ES) for dynamic optimization. These three strategies include the elitist (1+1)-CMA-ES, the non-elitist (μ, λ)-CMA-ES and sep-CMA-ES. To better understand the influence of covariance matrix adaptation methods and of the selection methods to the strategies in dynamic environments, we use the state-of-art dynamic optimization benchmark problems to evaluate the performance. We compare these CMA-ES variants with the traditional (1+1)-ES with the one-fifth success rule. Our experimental results show that the simple elitist strategies including the (1+1)-ES and the (1+1)-CMA-ES generally outperform those non-elitist CMA-ES variants on one out of the six dynamic functions. We also investigate the performance when the dynamic environments change with different severity and when the problems are in higher dimensions. The elitist strategies are robust to different severity of dynamic changes, but the performance is worse when the problem dimensions are increased. In high dimensions, the performance of the elitist and the non-elitist versions of CMA-ES are marginally the same.

Keywords: Dynamic Optimization, Evolution Strategies, Covariance Matrix Adaptation.

1 Introduction

In recent years, there has been a fair amount of research works that have contributed to the state-of-art covariance matrix adaptation evolution strategies (CMA-ES) [1,2,3,4] that is used to solve many black-box optimization problems. CMA-ES usually optimizes the real-valued objective functions $f : \mathbb{R}^n \to \mathbb{R}$ in the continuous domain. On ill-conditioned problems covariance matrix adaptation can accelerate the rate of convergence of evolution strategies by orders of magnitude. For example, a successful covariance matrix adaptation can enable strategies to generate candidate solutions predominantly in the direction of narrow valleys. The CMA-ES is able to learn the appropriate covariance matrix from the successful steps that the algorithm has taken. The covariance matrix is updated such that variances in the directions of the search space that have

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 529–538, 2012.

previously been successful are increased while those in other directions are decreased. Even for a small population, the accumulation of information over a number of successful steps can reliably adapt the covariance matrix.

However, the problem classes that have been considered in most of these works are of a static nature. In contrast, many problems in engineering, computational and biological domains are dynamic in that the objective functions are not constant but vary with time. Examples of dynamic optimization problems arise in the context of online job scheduling, where new jobs arrive in the course of optimization. A complete list of survey and works on the evolutionary algorithms for dynamic optimization has been reviewed by [5,6].

There are a few works that focus on evolution strategies in dynamic optimization. The early work [7] has empirically studied the family of evolution strategies in dynamic rotating problems. It investigated the performance when evolution strategies employed different forms of mutation step size adaptation. The experimental results show that a simple mutation step size adaptation achieves the best results compared to other complicated adaptation mechanisms including covariance matrix adaptation. It also suggested to use small populations in evolution strategies because using large populations implies a higher degree of dynamism and this is undesirable in dynamic optimization. Another work [8] studies evolution strategies for the number of mutation step sizes required when the optima move in one or all n coordinates with different severity. The results showed that adapting all n mutation step sizes achieves a better performance than adapting a single mutation step size. The work [9] compares different variants of mutative self-adaptation and shows that the lognormal self-adaptation used in evolution strategies performs better than the variants of self-adaptation commonly used in evolutionary programming. Obviously all these works demonstrate the difficulty of understanding the behavior of evolution strategies and their operators and parameters for dynamic environments.

In this paper, we will empirically investigate the state-of-art CMA-ES variants in the literature and study their performance for dynamic optimization. In the next two sections, we will briefly recall the CMA-ES variants and the dynamic optimization benchmark problems. The empirical comparison is described in Section 4 followed by the conclusion in the last section.

2 CMA-ES Variants

The standard (μ, λ)-CMA-ES: In the standard (μ, λ)-CMA-ES [1,2], in each iteration g, λ number of candidate solutions are generated by sampling a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{C})$ with mean $\mathbf{0}$ and $n \times n$ covariance matrix $\mathbf{C}$. The μ best solutions are selected to update the distribution parameters for the next iteration step $g + 1$. The standard CMA-ES employs the concept of cumulative step adaptation (CSA). There are two evolution paths $\mathbf{p}_\sigma$ and $\mathbf{p}_c$ and they are two n-dimensional vectors that are used to accumulate the information about the recent steps of the strategy. The learning of the accumulating information is controlled by three independent learning rates c_σ, c_1 and c_c that

change the global step size σ and the covariance matrix $\mathbf{C}$. The standard (μ, λ)-CMA-ES in this paper is identical to that described by [2] and is summarized in Table 1.

Table 1. Update equations in the standard (μ, λ)-CMA-ES with iteration index $g = 1, 2, 3, \ldots$. The symbol $\mathbf{x}_{i:\lambda}$ represents the i-th best of the candidate solutions $\mathbf{x}_1, \ldots, \mathbf{x}_\lambda$. The values of learning parameters $c_1, c_c, c_\mu, c_\sigma$ are set to the same values as in [1,2].

Given $g \in \mathbb{N} \cup \{0\}, \mathbf{m}^g \in \mathbb{R}^n, \sigma^g \in \mathbb{R}, \mathbf{C}^g \in \mathbb{R}^{n \times n}, \mathbf{p}_\sigma^g, \mathbf{p}_c^g \in \mathbb{R}^n$ and
$\mathbf{p}_\sigma^{g=0} = \mathbf{p}_c^{g=0} = \mathbf{0}, \mathbf{C}^{g=0} = \mathbf{I}$

$\mathbf{x}_i \sim \mathbf{m}^g + \sigma^g \times \boldsymbol{\mathcal{N}}_i(\mathbf{0}, \mathbf{C}^g)$ is normally distributed for $i = 1, \ldots, \lambda$

$\mathbf{m}^{g+1} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}$ where $f(\mathbf{x}_{1:\lambda}) \leq \cdots \leq f(\mathbf{x}_{\lambda:\lambda})$

$\mathbf{p}_\sigma^{g+1} = (1 - c_\sigma)\mathbf{p}_\sigma^g + \sqrt{c_\sigma(2 - c_\sigma)\mu_w}\mathbf{C}^{g-\frac{1}{2}}\frac{\mathbf{m}^{g+1} - \mathbf{m}^g}{\sigma^g}$
for $\mathbf{C}^{g-\frac{1}{2}} = \mathbf{B}\mathbf{D}^{-\frac{1}{2}}\mathbf{B}^T, \mathbf{B}\mathbf{D}\mathbf{B}^T = \mathbf{C}$

$h_\sigma = \begin{cases} 1 & \text{if } ||\mathbf{p}_\sigma^{g+1}|| < \sqrt{1 - (1 - c_\sigma)^{2(g+1)}}(1.4 + \frac{2}{n+1})\mathrm{E}||\boldsymbol{\mathcal{N}}(\mathbf{0}, \mathbf{I})|| \\ 0 & \text{otherwise} \end{cases}$

$\mathbf{p}_c^{g+1} = (1 - c_c)\mathbf{p}_c^g + h_\sigma\sqrt{c_c(2 - c_c)\mu_w}\frac{\mathbf{m}^{g+1} - \mathbf{m}^g}{\sigma^g}$

$\mathbf{C}_\mu = \sum_{i=1}^{\mu} w_i \frac{\mathbf{x}_{i:\lambda} - \mathbf{m}^g}{\sigma_g} \times \frac{(\mathbf{x}_{i:\lambda} - \mathbf{m}^g)^T}{\sigma_g}$

$\mathbf{C}^{g+1} = (1 - c_1 - c_\mu)\mathbf{C}^g + c_1\mathbf{p}_c^{g+1}\mathbf{p}_c^{g+1^T} + c_\mu\mathbf{C}_\mu$

$\sigma^{g+1} = \sigma^g \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||\mathbf{p}_\sigma^{g+1}||}{\mathrm{E}||\boldsymbol{\mathcal{N}}(\mathbf{0},\mathbf{I})||} - 1\right)\right)$ where $\mathrm{E}||\boldsymbol{\mathcal{N}}(\mathbf{0}, \mathbf{I})||$
is the expectation of the n-dimensional normal distributed vector.

The sep-CMA-ES: In the standard CMA-ES, the full learning task scales roughly with n^2 and can dominate most of the search cost. This is one of the major limitations in the standard CMA-ES because of the high degree of freedom $\frac{n^2+n}{2}$ in the covariance matrix. One of the solutions to this is to reduce the degree of freedom from $\frac{n^2+n}{2}$ to n where only the diagonal of the covariance matrix is adapted. The resulting algorithm is called "sep-CMA-ES" [3]. There are two simple changes undertaken in sep-CMA-ES. First, the covariance matrix $\mathbf{C}$ is constrained to be diagonal. Second, the learning rate c_μ is increased. This means that the mutation distribution is sampled independently in the given coordinate system using n individual variances. For sep-CMA-ES, the changes to the update equations in Table 1 are:

1. $\mathbf{D} = \sqrt{\mathrm{diag}(\mathbf{C})}$ where $\mathrm{diag}(\mathbf{C})$ is a diagonal matrix with the same diagonal elements as $\mathbf{C}$. The matrix $\mathbf{B}$ remains $\mathbf{I}$ for all iterations.

2. $c_\mu^{\text{sep-CMA-ES}} = \frac{n+2}{3} \cdot c_\mu$

(1+1)-CMA-ES: The (1+1)-CMA-ES is a new variant that has been recently proposed by [4] as an extension of (1+1)-ES with the one-fifth success rule [10]. It differs from the standard CMA-ES variant in that: (1) it is an elitist algorithm, and (2) not only the step size but also a covariance matrix associated to the search distribution is adapted. The experimental results in [4] shows that it is about 1.5 times faster than the standard CMA-ES on unimodal functions. We follow the principles introduced in [4] and the (1+1)-CMA-ES is summarized in Table 2. A candidate solution $\mathbf{y}^g$ is sampled by perturbing the current solution $\mathbf{x}^g$ by adding a normal distributed vector with mean vector $\mathbf{0}$ and covariance matrix $\mathbf{C}^g$ and scaled by the mutation step-size σ^g. This candidate solution is accepted only if $f(\mathbf{y}^g) < f(\mathbf{x}^g)$. The mutation step-size is adapted using the averaged success rate p_{succ} such that it is increased if the success rate is strictly larger than the target probability p_{succ}^{target}, and decreased if it is strictly smaller. If $f(\mathbf{y}^g) < f(\mathbf{x}^g)$, the covariance matrix is adapted by adding to a multiple of $\mathbf{C}^g$ the rank-one update matrix $\mathbf{p}^{g+1}\mathbf{p}^{g+1^T}$ where $\mathbf{p}^{g+1^T}$ is the transpose of $\mathbf{p}^{g+1}$. We will use the same default settings as in [4] for all strategy parameters.

Table 2. Update equations in the $(1+1)$-CMA-ES with iteration index $g = 1, 2, 3, \ldots$

Given $g \in \mathbb{N} \cup \{0\}\,, \mathbf{x}^g \in \mathbb{R}^n, \sigma^g \in \mathbb{R}, \mathbf{C}^g \in \mathbb{R}^{n\times n}, p_{\text{succ}}^g \in \mathbb{R}, \mathbf{p}^{g=0} = \mathbf{0}, \mathbf{C}^{g=0} = \mathbf{I}$ and $p_{\text{succ}}^{g=0} = p_{succ}^{target} = \frac{2}{11}, c_p = \frac{1}{12}, c_c = \frac{2}{N+2}, c_\mu = \frac{2}{N^2+6}, p_{thresh} = 0.44$
$\mathbf{A}^g = \text{chol}(\mathbf{C}^g)$ where $\text{chol}(\cdot)$ is the Cholesky decompositions such that $\mathbf{C} = \mathbf{A}\mathbf{A}^T$
$\mathbf{z}^g \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$\mathbf{y}^g \sim \mathbf{x}^g + \sigma^g \mathbf{A}^g \mathbf{z}^g$
$p_{\text{succ}}^{g+1} = (1 - c_p) p_{\text{succ}}^g + c_p \mathbf{1}_{\{f(\mathbf{y}^g) \le f(\mathbf{x}^g)\}}$
$\sigma^{g+1} = \sigma^g \exp\left(\frac{p_{\text{succ}}^{g+1} - p_{succ}^{target}}{n \cdot (1 - p_{succ}^{target})}\right)$
$\mathbf{p}^{g+1} = \begin{cases} (1 - c_c)\mathbf{p}^g + \mathbf{1}_{\{p_{succ}^{g+1} < p_{thresh}\}} \sqrt{c_c(2 - c_c)}\, \mathbf{A}^g \mathbf{z}^g & \text{if } f(\mathbf{y}^g) \le f(\mathbf{x}^g) \\ \mathbf{p}^g & \text{otherwise} \end{cases}$
$\mathbf{C}^{g+1} = \begin{cases} \left(1 - c_\mu + c_\mu \mathbf{1}_{\{p_{succ}^{g+1} > p_{thresh}\}} c_c(2 - c_c)\right) \mathbf{C}^g & \\ \quad + c_\mu \mathbf{p}^{g+1} \mathbf{p}^{g+1^T} & \text{if } f(\mathbf{y}^g) \le f(\mathbf{x}^g) \\ \mathbf{C}^g & \text{otherwise} \end{cases}$
$\mathbf{x}^{g+1} = \begin{cases} \mathbf{y}^g & \text{if } f(\mathbf{y}^g) \le f(\mathbf{x}^g) \\ \mathbf{x}^g & \text{otherwise} \end{cases}$

3 Dynamic Optimization Benchmark

In dynamic optimization, the objective function changes during the course of optimization. At any given time $t \in \mathbb{T}$, one needs to find the solutions $\mathbf{x}^*$ such that $\forall \mathbf{x}, \mathbf{x}^* \in \mathbb{R}^n, f(\mathbf{x}^*, t) \le f(\mathbf{x}, t)$ where $f : \mathbb{R}^n \times \mathbb{T} \to \mathbb{R}$ is the objective function of a minimization problem and n is the problem dimension. For dynamic optimization problems, the fitness functions, design variables and environmental conditions change from time to time. The simplest way to solve dynamic

optimization problems is to consider each change as an arrival of a new static optimization problem if the time and computational resources are sufficient. However, time and resources given are always limited and the explicit restart approach may not be feasible.

In this paper, we use the generalized dynamic benchmark generator (GDBG)[1]. [11] to evaluate the performance of the CMA-ES variants. This benchmark is a problem generator that can construct dynamic environments in the continuous space. It differs from the benchmarks in the literature that it uses the rotation method instead of shifting the positions of the peaks. Using the rotation method can prevent the unequal challenge per change for the algorithms when the positions of the peaks bounce back from the boundary of the search space.

Types of Environment Changes: We focus on the non-dimensional changes in GDBG. In non-dimensional changes, the values of variables within the problem constraints are changed. One of the examples is to increase or decrease the number of peaks during the course of optimization. Formally, we can describe dynamic changes as: $\phi(t+1) = \phi(t) \oplus \Delta\phi$ where $\phi(t)$ is the system control parameters and $\Delta\phi$ is a deviation from the current system control parameters. At time t, the new environment at time $t+1$ can be expressed as: $f(x, \phi, t+1) = f(x, \phi(t) \oplus \Delta\phi, t)$. There are six types of the non-dimensional changes, including the small step change, the large step change, the random change, the chaotic change, the recurrent change, and the recurrent change with noisy. We name them C_1 to C_6 for our easy reference:

- C_1 Small step change: $\Delta\phi = \alpha \cdot ||\phi|| \cdot r \cdot \phi_{severity}$
- C_2 Large step change: $\Delta\phi = ||\phi|| \cdot (\alpha \cdot sgn(r) + (\alpha_{max} - \alpha) \cdot r) \cdot \phi_{severity}$
- C_3 Random step change: $\Delta\phi = \mathcal{N}(0,1) \cdot \phi_{severity}$
- C_4 Chaotic change: $\phi(t+1) = A \cdot \phi(t) \cdot (1 - \frac{\phi(t)}{||\phi||})$
- C_5 Recurrent change: $\phi(t+1) = \phi_{min} + ||\phi|| \cdot \frac{\left(sin(\frac{2\pi t}{P} + \varphi) + 1\right)}{2}$
- C_6 Recurrent change step with noise:

$$\phi(t+1) = \phi_{min} + ||\phi|| \cdot \frac{\left(sin(\frac{2\pi t}{P} + \varphi) + 1\right)}{2} + \mathcal{N}(0,1) \cdot \phi_{noisyseverity}$$

where $||\phi||$ is the range of ϕ, $\phi_{severity} \in (0,1)$ is the change severity of ϕ, ϕ_{min} is the minimum value of ϕ, $\phi_{noisyseverity} \in (0,1)$ is the noisy severity in the recurrent change change with noise. The parameters $\alpha \in (0,1)$ and $\alpha_{max} \in (0,1)$ are constant values in C_1 Small step change and C_2 Large step change. A logistics function is used in the C_4 Chaotic change, where A is a positive constant between $(1.0, 4.0)$, if ϕ is a vector, the initial values of the items in ϕ should be different within $||\phi||$ in C_4 chaotic change. P is the period in the C_5 Recurrent step change and the C_6 Recurrent step change with noise, φ is the initial phase, r is a random

[1] Due to pages constraints, we outline the key equations of GDBG. For complete details, please read [11].

number drawn uniformly from -1 and 1. The function $sgn(x)$ returns 1 when x is greater than 0, returns -1 when x is less than 0, otherwise returns 0. Finally, $\mathcal{N}(0,1)$ returns a normal distributed one dimensional random number with mean zero and standard derivation one.

Rotation DBG F_1: There are two instances in the GBDB benchmark: Rotation DBG and Composition DBG. In the Rotation DBG, the fitness landscape consists of multiple peaks that can be controlled by tuning the system control parameters. The height, the width and the position of each peak are changed in the six change types described above. If the dynamic problem is $f(x,\phi,t)$, then the set of system control parameters is $\phi = (\mathbf{H},\mathbf{W},\mathbf{X})$, where $\mathbf{H}$,$\mathbf{W}$ and $\mathbf{X}$ are the peak height, the peak width and the peak position respectively. Formally, the function $f(x,\phi,t)$ is

$$f(x,\phi,t) = \min\left\{\mathbf{H}_i(t)+\mathbf{W}_i(t)\left(\exp\left(\sqrt{\sum_{j=1}^{n}\frac{(x_j-\mathbf{X}_j^i(t))^2}{n}}\right)-1\right)\right\}_{i=1}^{m}$$

where m is the number of peaks, n is the problem dimension. The height and the width of the peaks are changed: $\mathbf{H}(t+1) = \text{DynamicChanges}\left(\mathbf{H}(t),\phi_{h_{severity}},||\phi_h||\right)$ and $\mathbf{W}(t+1) = \text{DynamicChanges}\left(\mathbf{W}(t),\phi_{h_{severity}},||\phi_h||\right)$ where the change severity of the height and the width are $\phi_{h_{severity}}$ and $\phi_{w_{severity}}$ respectively. The ranges of the height and the width are denoted by $||\phi_h||$ and $||\phi_w|$.

Composition DBG F_2 to F_6: Another instance of the GDBG benchmark is the Composition DBG. The basic idea is to construct more challenging benchmark functions with randomly located global and local optima. By shifting, rotating and composing the global optima of the standard functions, more challenging test functions that possesses many desirable properties can be obtained. Formally, the Composition DBG can be described as follows:

$$F(x,\phi,t) = \sum_{i=1}^{m}\left\{w_i^{'}\cdot\left(f_i^{'}\left(\frac{(x-\mathbf{O}_i(t)+\mathbf{O}_{iold})\cdot\mathbf{M}_i}{\lambda_i}\right)+\mathbf{H}_i(t)\right)\right\}$$

where the system control parameter $\phi = (\mathbf{O},\mathbf{M},\mathbf{H})$, $F(x)$ is the composition function, $f_i(\mathbf{x})$ is the i-th basic function used to construct the composition function, m is the number of the basic functions, $\mathbf{M}_i$ is the orthogonal rotation matrix for each $f_i(\mathbf{x})$, $\mathbf{O}_i$ and $\mathbf{O}_{iold}$ are the shifted and the old optimum position respectively for each basic function $f_i(\mathbf{x})$.

4 Experimental Study

Setup: In our setup, we use the same set of problems in [11]. A total of six dynamic problems F_1 to F_6 are tested[2] . All six problems are multi-modal, scalable, rotated and have a large number of local optima. Unless stated otherwise,

[2] For details of functions please reference to [11].

a change will occur only after $1e^2 \cdot n$ number of functions evaluations are used. 50 independent runs are executed per each problem and per each change. All problems have the global optimum within the given bounds and there is no need to perform search outside of the given bounds for these problems. All algorithms will be terminated when the number of changes reaches 60. To evaluate the performance of the algorithms for maximization problems, we record the relative function error value $E^{last}(t) = \frac{f(\mathbf{x}_{best}(t))}{f(\mathbf{x}^*(t))}$ after each change. The vector $\mathbf{x}_{best}(t)$ is the best solutions found by the algorithm at time t and the vector $\mathbf{x}^*(t)$ is the location of the global optimum at time t. In our experiments, all three variants of CMA-ES and the (1+1)-ES with the one-fifth success rule are compared on all six benchmark problems. All strategy parameters of evolution strategies are set to the default values in their original works [10,1,2,3,4]. No parameters tuning has been conducted.

Results. The experimental results are shown in Figure 1. The graphs show the relative function error values against the change types. The whiskers in the graphs mark the 10th and 90th percentiles. We first take a look at the first problem F_1 Rotation Peak Problem. The performances of the (1+1)-ES and the (1+1)-CMA-ES generally outperform the standard CMA-ES and the sep-CMA-ES. The results are consistent for all 6 types of dynamic changes. An elitist evolution strategies using a small population size leads to the best results compared to all other strategies that are non-elitist and are in large population sizes. Both the (1+1)-ES and the (1+1)-CMA-ES are statistically indistinguishable for all 6 types of dynamic changes. Comparing the standard CMA-ES with the sep-CMA-ES, their performance are also statistically equivalent. None of our statistical tests are able to show any significance in these two strategies. Obviously the simple adaptation technique like the one-fifth success rule can adapt quickly to the dynamically changing environments. The more complicated mechanisms that produce very good results in static optimization, are not adapting very well for dynamic optimization. If we look into the graph for F_2, the results are consistent with those in F_1. The (1+1)-ES and the (1+1)-CMA-ES achieve the best performance. From functions F_3 to F_6, the performance of all strategies becomes worser since the fitness landscapes are more rugged than the functions F_1 and F_2. However, the performance differences between the elitist and non-elitist versions become smaller. In some of the cases in functions F_3 and F_5, all four variants are statistically equivalent. In functions F_4 and F_6, there are a few cases where the CMA-ES and the sep-CMA-ES outperform the (1+1) variants. Overall all strategies in most of the cases are indistinguishable in functions F_4 and F_6.

We next investigate how these strategies are robust to dynamic changes with different severity and to the problem dimensions. Figure 2 shows the median numbers of relative function errors against the severity when the underlying function is F_1 and the change type is C_3. The severity $\phi_{severity}$ is normalized such that severity is equal to $\frac{\phi_{severity}}{|\phi|}$ where $|\phi|$ is the range of the system control parameters. When we increase the severity, the performance generally becomes

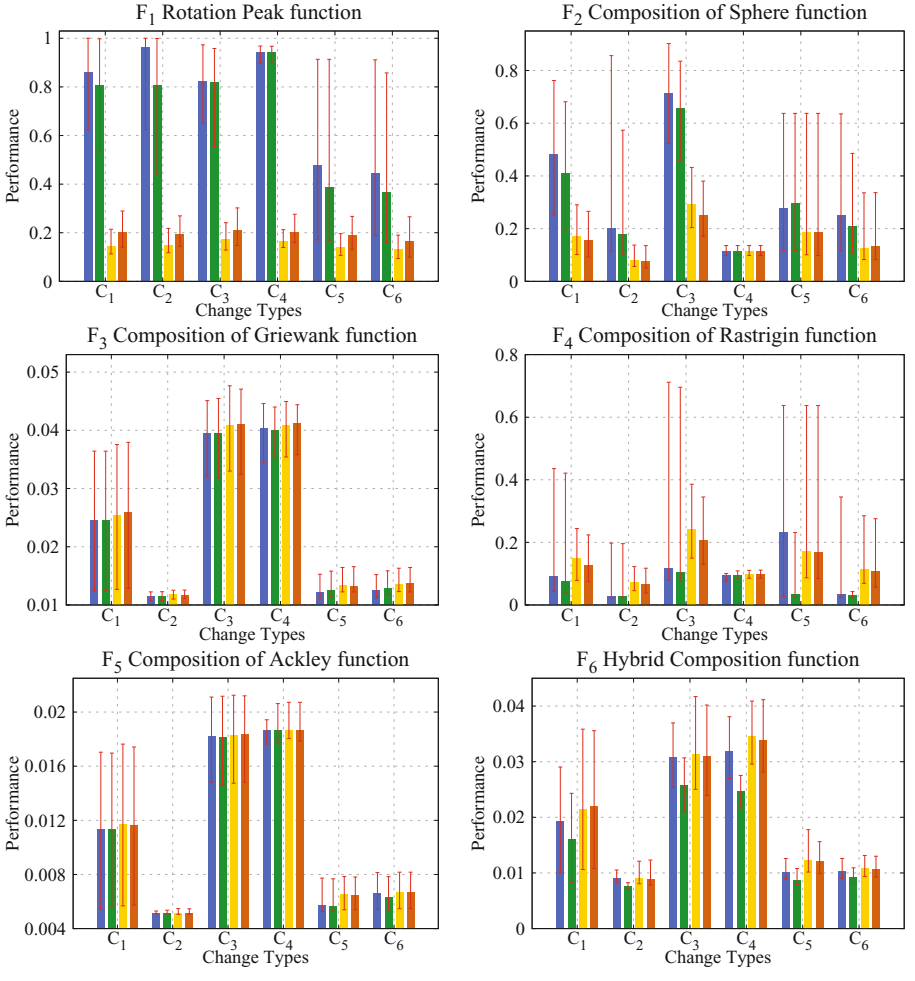


Fig. 1. The median performance of the CMA-ES variants and the (1+1)-ES with the one-fifth success rule on F_1 to F_6 over the trials of 50. The dynamic change types are C_1 Small step change, C_2 Large step change, C_3 Random step change, C_4 Chaotic change, C_5 Recurrent change and C_6 Recurrent change step with noise. For each change type, from left to right, the bars represent the (1+1)-ES with the one-fifth success rule (—), (1+1)-CMA-ES (—), the standard CMA-ES (—) and the sep-CMA-ES (—).

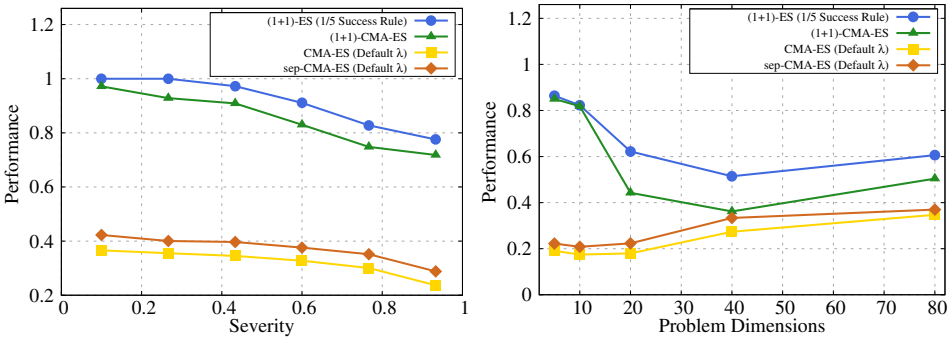


Fig. 2. The median performance of the CMA-ES variants and the (1+1)-ES with the one-fifth success rule on F_1 when the change type is C_3 random step change. The left graph shows the performance against the severity of the dynamic changes while the right graph shows the performance against the problem dimensions.

worser. The elitist (1+1)-ES and the (1+1)-CMA-ES are generally better than the non-elitist strategies in dynamic changes with different severity. Lastly we investigate the performance of the strategies when the dimensions are increased. The right graph in Figure 2 shows the median numbers against the problem dimensions. The performance of elitist strategies becomes worser when the problem dimensions are scaled up. We believe this is due to the small population sizes of these point-based (1+1) strategies. In contrast to the elitist strategies, the non-elitist version of the CMA-ES that are population-based improves gradually in higher dimensions. The performance gap between the elitist and non-elitist CMA-ES is getting smaller. Obviously when we increase the problem dimension, the dynamic problems become more challenging and using the population-based strategies are necessary in order to achieve a reasonable performance.

5 Conclusion

In this paper, we investigate the state-of-art CMA-ES variants for dynamic optimization and they include the elitist (1+1)-CMA-ES, the standard (μ, λ)-CMA-ES and the sep-(μ, λ)-CMA-ES. We first briefly review the CMA-ES variants in the context of static optimization and then we discuss the latest dynamic optimization benchmark problems that are used in our simulations. On one out of the six dynamic functions, the elitist (1+1)-ES with the one-fifth rule and the (1+1)-CMA-ES achieve the best performance. In most of our simulations, these two elitist strategies are statistically equivalent. The non-elitist strategies, including the standard (μ, λ)-CMA-ES and the sep-CMA-ES, are outperformed by the elitist variants. The results are consistent for dynamic changes with different severity. However, the performance of the elitist strategies, which are pointed-based search algorithms, becomes worser for higher dimensional problems. Using the population-based strategies like the standard (μ, λ)-CMA-ES and the sep-CMA-ES can achieve the equivalent performance as what the elitist (1+1) variants do.

In the future work, it would be interesting to introduce additional diversity into the CMA-ES variants. Concentrating the search near the the current optima in a dynamic environment could make the strategies missing the important changes in different region of the search space. Adding predictions mechanism and diversity control methods can be a promising way for CMA-ES to optimize the dynamic functions.

References

1. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
2. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). Evolutionary Computation 11(1), 1–18 (2003)
3. Ros, R., Hansen, N.: A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 296–305. Springer, Heidelberg (2008)
4. Igel, C., Suttorp, T., Hansen, N.: A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO 2006, pp. 453–460 (2006)
5. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments-a survey. IEEE Transactions on Evolutionary Computation
6. Branke, J.: Evolutionary Optimization in Dynamic Environments (2001)
7. Weicker, K., Weicker, N.: On evolution strategy optimization in dynamic environments. In: Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999 vol.3., 3 vol. xxxvii+2348 (1999)
8. Schönemann, L.: Evolution Strategies in Dynamic Environments. In: Yang, S., Ong, Y.S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments. SCI, vol. 51, pp. 51–77. Springer, Heidelberg (2007)
9. Back, T.: On the behavior of evolutionary algorithms in dynamic environments. In: IEEE World Congress on Computational Intelligence, The 1998 IEEE International Conference on Evolutionary Computation Proceedings, pp. 446–451 (May 1998)
10. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. PhD thesis, TU Berlin (1971)
11. Li, C., Yang, S., Nguyen, T.T., Yu, E.L., Yao, X., Jin, Y., Beyer, H.G., Suganthan, P.N.: Benchmark generator for cec 2009 competition on dynamic optimization (2008)

Author Index