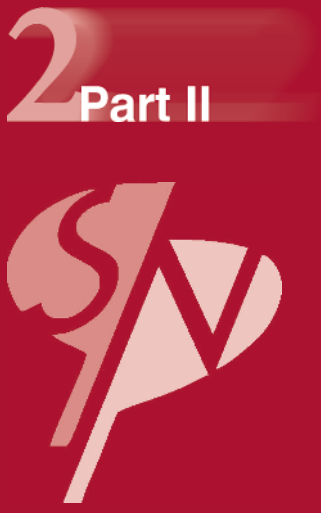
Anne Auger · Carlos M. Fonseca
Nuno Lourenço · Penousal Machado
Luís Paquete · Darrell Whitley (Eds.)

LNCS 11102

Parallel Problem Solving from Nature – PPSN XV

15th International Conference
Coimbra, Portugal, September 8–12, 2018
Proceedings, Part II



Springer

Lecture Notes in Computer Science 11102

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

Editorial Board

More information about this series at http://www.springer.com/series/7407

Anne Auger · Carlos M. Fonseca
Nuno Lourenço · Penousal Machado
Luís Paquete · Darrell Whitley (Eds.)

Parallel Problem Solving from Nature – PPSN XV

15th International Conference
Coimbra, Portugal, September 8–12, 2018
Proceedings, Part II

Springer

Editors
Anne Auger
Inria Saclay
Palaiseau
France

Carlos M. Fonseca
University of Coimbra
Coimbra
Portugal

Nuno Lourenço
University of Coimbra
Coimbra
Portugal

Penousal Machado
University of Coimbra
Coimbra
Portugal

Luís Paquete
University of Coimbra
Coimbra
Portugal

Darrell Whitley
Colorado State University
Fort Collins, CO
USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-99258-7 ISBN 978-3-319-99259-4 (eBook)
https://doi.org/10.1007/978-3-319-99259-4

Library of Congress Control Number: 2018951432

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

During September 8–12, 2018, researchers from all over the world gathered in Coimbra, Portugal, for the 15th International Conference on Parallel Problem Solving from Nature (PPSN XV). Far more than a European event, this biennial meeting has established itself among the most important and highly respected international conferences in nature-inspired computation worldwide since its first edition in Dortmund in 1990. These two LNCS volumes contain the proceedings of the conference.

We received 205 submissions from 44 countries. An extensive review process involved over 200 reviewers, who evaluated and reported on the manuscripts. All papers were assigned to at least three Program Committee members for review. A total of 745 review reports were received, or over 3.6 reviews on average per manuscript. All review reports were analyzed in detail by the Program Chairs. Where there was disagreement among reviewers, the Program Chairs also evaluated the papers themselves. In some cases, discussion among reviewers with conflicting reviews was promoted with the aim of making as accurate and fair a decision as possible. Overall, 79 manuscripts were selected for presentation and inclusion in the proceedings, which represents an acceptance rate just below 38.6%. This makes PPSN 2018 the most selective PPSN conference of the past 12 years, and reinforces its position as a major, high-quality evolutionary computation scientific event.

The meeting began with an extensive program of 23 tutorials and six workshops covering a wide range of topics in evolutionary computation and related areas, including machine learning, statistics, and mathematical programming. Tutorials offered participants the opportunity to learn more about well-established, as well as more recent, research, while workshops provided a friendly environment where new ideas could be presented and discussed by participants with similar interests.

In addition, three distinguished invited speakers delivered keynote addresses at the conference. Ahmed Elgammal (Rutgers University, USA), Francis Heylighen (Vrije Universiteit Brussel, Belgium), and Kurt Mehlhorn (Max Planck Institute for Informatics, Saarbrücken, Germany) spoke on advances in the area of artificial intelligence and art, foundational concepts and mechanisms that underlie parallel problem solving in nature, and models of computation by living organisms, respectively.

We thank the authors of all submitted manuscripts, and express our appreciation to all the members of the Program Committee and external reviewers who provided thorough evaluations of those submissions. We thank the keynote speakers, tutorial speakers, and workshop organizers for significantly enriching the scientific program with their participation. To all members of the Organizing Committee and local organizers, we extend our deep gratitude for their dedication in preparing and running the conference. Special thanks are due to the University of Coimbra for hosting the conference and, in particular, to INESC Coimbra, CISUC, the Department of Informatics Engineering, the Department of Mathematics, and the International Relations Unit, for their invaluable contribution to the organization of this event, and to the

sponsoring institutions for their generosity. Finally, we wish to personally thank Carlos Henggeler Antunes for his unconditional support.

September 2018

Anne Auger
Carlos M. Fonseca
Nuno Lourenço
Penousal Machado
Luís Paquete
Darrell Whitley

Organization

PPSN 2018 was organized by INESC Coimbra and CISUC, and was hosted by the University of Coimbra, Portugal. Established in 1290, the University of Coimbra is the oldest university in the country and among the oldest in the world. It is a UNESCO World Heritage site since 2013.

Organizing Committee

General Chairs

Carlos M. Fonseca	University of Coimbra, Portugal
Penousal Machado	University of Coimbra, Portugal

Honorary Chair

Hans-Paul Schwefel	TU Dortmund University, Germany

Program Chairs

Anne Auger	Inria Saclay, France
Luís Paquete	University of Coimbra, Portugal
Darrell Whitley	Colorado State University, USA

Workshop Chairs

Robin C. Purshouse	University of Sheffield, UK
Christine Zarges	Aberystwyth University, UK

Tutorial Chairs

Michael T. M. Emmerich	Leiden University, The Netherlands
Gisele L. Pappa	Federal University of Minas Gerais, Brazil

Publications Chair

Nuno Lourenço	University of Coimbra, Portugal

Local Organization Chair

Pedro Martins	University of Coimbra, Portugal

Webmasters

Catarina Maçãs	University of Coimbra, Portugal
Evgheni Polisciuc	University of Coimbra, Portugal

Steering Committee

David W. Corne	Heriot-Watt University Edinburgh, UK
Carlos Cotta	Universidad de Malaga, Spain
Kenneth De Jong	George Mason University, USA
Agoston E. Eiben	Vrije Universiteit Amsterdam, The Netherlands
Bogdan Filipič	Jožef Stefan Institute, Slovenia
Emma Hart	Edinburgh Napier University, UK
Juan Julián Merelo Guervós	Universidad de Granada, Spain
Günter Rudolph	TU Dortmund University, Germany
Thomas P. Runarsson	University of Iceland, Iceland
Robert Schaefer	University of Krakow, Poland
Marc Schoenauer	Inria, France
Xin Yao	University of Birmingham, UK

Keynote Speakers

Ahmed Elgammal	Rutgers University, USA
Francis Heylighen	Vrije Universiteit Brussel, Belgium
Kurt Mehlhorn	Max Planck Institute for Informatics, Germany

Program Committee

Youhei Akimoto	Shinshu University, Japan
Richard Allmendinger	University of Manchester, UK
Dirk Arnold	Dalhousie University, Canada
Asma Atamna	Inria, France
Anne Auger	Inria, France
Dogan Aydin	Dumlupinar University, Turkey
Jaume Bacardit	Newcastle University, UK
Helio Barbosa	Laboratório Nacional de Computação Científica, Brasil
Thomas Bartz-Beielstein	Cologne University of Applied Sciences, Germany
Heder Bernardino	Universidade Federal de Juiz de Fora, Brasil
Hans-Georg Beyer	Vorarlberg University of Applied Sciences, Austria
Mauro Birattari	Université Libre de Bruxelles, Belgium
Christian Blum	Spanish National Research Council, Spain
Peter Bosman	Centrum Wiskunde & Informatica, The Netherlands
Pascal Bouvry	University of Luxembourg, Luxembourg
Juergen Branke	University of Warwick, UK
Dimo Brockhoff	Inria and Ecole Polytechnique, France
Will Browne	Victoria University of Wellington, New Zealand
Alexander Brownlee	University of Stirling, Scotland
Larry Bull	University of the West of England, England
Arina Buzdalova	ITMO University, Russia
Maxim Buzdalov	ITMO University, Russia
Stefano Cagnoni	University of Parma, Italy
David Cairns	University of Stirling, Scotland

Mauro Castelli	Universidade Nova de Lisboa, Portugal
Wenxiang Chen	Colorado State University, USA
Ying-Ping Chen	National Chiao Tung University, Taiwan
Marco Chiarandini	University of Southern Denmark, Denmark
Francisco Chicano	University of Málaga, Spain
Miroslav Chlebik	University of Sussex, UK
Sung-Bae Cho	Yonsei University, South Korea
Alexandre Chotard	Inria, France
Carlos Coello Coello	CINVESTAV-IPN, Mexico
Dogan Corus	University of Nottingham, UK
Ernesto Costa	University of Coimbra, Portugal
Carlos Cotta	University of Málaga, Spain
Kenneth De Jong	George Mason University, USA
Antonio Della Cioppa	University of Salerno, Italy
Bilel Derbel	University of Lille, France
Benjamin Doerr	École Polytechnique, France
Carola Doerr	Sorbonne University, Paris, France
Marco Dorigo	Université Libre de Bruxelles, Belgium
Johann Dréo	Thales Research & Technology, France
Rafal Drezewski	AGH University of Science and Technology, Poland
Michael Emmerich	Leiden University, The Netherlands
Andries Engelbrecht	University of Pretoria, South Africa
Anton Eremeev	Omsk Branch of Sobolev Institute of Mathematics, Russia
Katti Faceli	Universidade Federal de São Carlos, Brasil
João Paulo Fernandes	University of Coimbra, Portugal
Pedro Ferreira	University of Lisbon, Portugal
José Rui Figueira	University of Lisbon, Portugal
Bogdan Filipic	Jožef Stefan Institute, Slovenia
Steffen Finck	Vorarlberg University of Applied Sciences, Austria
Andreas Fischbach	Cologne University of Applied Sciences, Germany
Peter Fleming	University of Sheffield, UK
Carlos M. Fonseca	University of Coimbra, Portugal
Martina Friese	Cologne University of Applied Sciences, Germany
Marcus Gallagher	University of Queensland, Australia
José García-Nieto	University of Málaga, Spain
Antonio Gaspar-Cunha	University of Minho, Portugal
Mario Giacobini	University of Torino, Italy
Tobias Glasmachers	Institut für Neuroinformatik, Germany
Roderich Gross	University of Sheffield, UK
Andreia Guerreiro	University of Coimbra, Portugal
Jussi Hakanen	University of Jyväskylä, Finland
Hisashi Handa	Kindai University, Japan
Julia Handl	University of Manchester, UK
Jin-Kao Hao	University of Angers, France
Emma Hart	Napier University, UK
Nikolaus Hansen	Inria, France

Verena Heidrich-Meisner	Christian-Albrechts-Universität zu Kiel, Germany
Carlos Henggeler Antunes	University of Coimbra, Portugal
Hisao Ishibuchi	Southern University of Science and Technology, China
Christian Jacob	University of Calgary, Canada
Domagoj Jakobovic	University of Zagreb, Croatia
Thomas Jansen	Aberystwyth University, Wales
Yaochu Jin	University of Surrey, England
Laetitia Jourdan	University of Lille, France
Bryant Julstrom	St. Cloud State University, USA
George Karakostas	McMaster University, Canada
Graham Kendall	University of Nottingham, UK
Timo Kötzing	Hasso-Plattner-Institut, Germany
Krzysztof Krawiec	Poznan University of Technology, Poland
Martin Krejca	Hasso-Plattner-Institut, Germany
Algirdas Lančinskas	Vilnius University, Lithuania
William Langdon	University College London, England
Frederic Lardeux	University of Angers, France
Jörg Lässig	University of Applied Sciences Zittau/Görlitz, Germany
Per Kristian Lehre	University of Birmingham, UK
Johannes Lengler	ETH Zurich, Switzerland
Arnaud Liefooghe	University of Lille, France
Andrei Lissovoi	University of Sheffield, UK
Giosuè Lo Bosco	Università di Palermo, Italy
Fernando Lobo	University of Algarve, Portugal
Daniele Loiacono	Politecnico di Milano, Italy
Manuel López-Ibáñez	University of Manchester, UK
Nuno Lourenço	University of Coimbra, Portugal
Jose A. Lozano	University of the Basque Country, Spain
Gabriel Luque	University of Málaga, Spain
Thibaut Lust	Sorbonne University, France
Penousal Machado	University of Coimbra, Portugal
Jacek Mańdziuk	Warsaw University of Technology, Poland
Vittorio Maniezzo	University of Bologna, Italy
Elena Marchiori	Radboud University, The Netherlands
Giancarlo Mauri	University of Milano-Bicocca, Italy
James McDermott	University College Dublin, Republic of Ireland
Alexander Melkozerov	Tomsk State University of Control Systems and Radioelectronics, Russia
J. J. Merelo	University of Granada, Spain
Marjan Mernik	University of Maribor, Slovenia
Silja Meyer-Nieberg	Universität der Bundeswehr München, Germany
Martin Middendorf	University of Leipzig, Germany
Kaisa Miettinen	University of Jyväskylä, Finland
Edmondo Minisci	University of Strathclyde, Scotland
Gara Miranda	University of La Laguna, Spain
Marco A. Montes De Oca	"clypd, Inc.", USA

Sanaz Mostaghim	Otto von Guericke University Magdeburg, Germany
Boris Naujoks	Cologne University of Applied Sciences, Germany
Antonio J. Nebro	University of Málaga, Spain
Ferrante Neri	De Montfort University, England
Frank Neumann	University of Adelaide, Australia
Phan Nguyen	University of Birmingham, UK
Miguel Nicolau	University College Dublin, Republic of Ireland
Kouhei Nishida	Shinshu University, Japan
Michael O' Neill	University College Dublin, Republic of Ireland
Gabriela Ochoa	University of Stirling, Scotland
Pietro S Oliveto	University of Sheffield, UK
José Carlos Ortiz-Bayliss	Tecnológico de Monterrey, Mexico
Ben Paechter	Napier University, UK
Gregor Papa	Jožef Stefan Institute, Slovenia
Gisele Pappa	Universidade Federal de Minas Gerais, Brasil
Luis Paquete	University of Coimbra, Portugal
Andrew J. Parkes	University of Nottingham, UK
Margarida Pato	Universidade de Lisboa, Portugal
Mario Pavone	University of Catania, Italy
David Pelta	University of Granada, Spain
Martin Pilat	Charles University in Prague, Czech Republic
Petr Pošík	Czech Technical University in Prague, Czech Republic
Mike Preuss	University of Münster, Germany
Robin Purshouse	University of Sheffield, UK
Günther Raidl	Vienna University of Technology, Austria
William Rand	North Carolina State University, USA
Khaled Rasheed	University of Georgia, USA
Tapabrata Ray	University of New South Wales, Australia
Eduardo Rodriguez-Tello	CINVESTAV-Tamaulipas, Mexico
Günter Rudolph	TU Dortmund University, Germany
Andrea Roli	University of Bologna, Italy
Agostinho Rosa	University of Lisbon, Portugal
Jonathan Rowe	University of Birmingham, UK
Thomas Runarsson	University of Iceland, Iceland
Thomas A. Runkler	Siemens Corporate Technology, Germany
Conor Ryan	University of Limerick, Republic of Ireland
Frédéric Saubion	University of Angers, France
Robert Schaefer	AGH University of Science and Technology, Poland
Andrea Schaerf	University of Udine, Italy
Manuel Schmitt	ALYN Woldenberg Family Hospital, Israel
Marc Schoenauer	Inria, France
Oliver Schuetze	CINVESTAV-IPN, Mexico
Eduardo Segredo	Napier University, UK
Martin Serpell	University of the West of England, England
Roberto Serra	University of Modena and Reggio Emilia, Italy
Marc Sevaux	Université de Bretagne-Sud, France
Shinichi Shirakawa	Yokohama National University, Japan

Kevin Sim	Napier University, UK
Moshe Sipper	Ben-Gurion University of the Negev, Israel
Jim Smith	University of the West of England, England
Christine Solnon	Institut National des Sciences Appliquées de Lyon, France
Sebastian Stich	EPFL, Switzerland
Catalin Stoean	University of Craiova, Romania
Jörg Stork	Cologne University of Applied Sciences, Germany
Thomas Stützle	Université Libre de Bruxelles, Belgium
Dirk Sudholt	University of Sheffield, UK
Andrew Sutton	University of Minnesota Duluth, USA
Jerry Swan	University of York, UK
Ricardo H. C. Takahashi	Universidade Federal de Minas Gerais, Brasil
El-Ghazali Talbi	University of Lille, France
Daniel Tauritz	Missouri University of Science and Technology, USA
Jorge Tavares	Microsoft, Germany
Hugo Terashima	Tecnológico de Monterrey, Mexico
German Terrazas Angulo	University of Nottingham, UK
Andrea Tettamanzi	University Nice Sophia Antipolis, France
Lothar Thiele	ETH Zurich, Switzerland
Dirk Thierens	Utrecht University, The Netherlands
Renato Tinós	University of São Paulo, Brasil
Alberto Tonda	Institut National de la Recherche Agronomique, France
Heike Trautmann	University of Münster, Germany
Leonardo Trujillo	Instituto Tecnológico de TIjuana, Mexico
Tea Tusar	Jožef Stefan Institute, Slovenia
Nadarajen Veerapen	University of Stirling, UK
Sébastien Verel	Université du Littoral Côte d'Opale, France
Markus Wagner	University of Adelaide, Australia
Elizabeth Wanner	Aston University, UK
Carsten Witt	Technical University of Denmark, Denmark
Man Leung Wong	Lingnan University, Hong Kong
John Woodward	Queen Mary University of London, UK
Ning Xiong	Mälardalen University, Sweden
Shengxiang Yang	De Montfort University, UK
Gary Yen	Oklahoma State University, USA
Martin Zaefferer	Cologne University of Applied Sciences, Germany
Ales Zamuda	University of Maribor, Slovenia
Christine Zarges	Aberystwyth University, UK

Additional Reviewers

Matthew Doyle
Yue Gu
Stefano Mauceri
Aníl Özdemir
Isaac Vandermuelen

Contents – Part II

Runtime Analysis and Approximation Results

Contents – Part I

Multi-objective Optimization

Parallel and Distributed Frameworks

Applications

Runtime Analysis and Approximation Results

A General Dichotomy of Evolutionary Algorithms on Monotone Functions

Johannes Lengler(✉)

Department of Computer Science, ETH Zürich, Zürich, Switzerland
johannes.lengler@inf.ethz.ch

Abstract. It is known that the $(1 + 1)$-EA with mutation rate c/n optimises every monotone function efficiently if $c < 1$, and needs exponential time on some monotone functions (HotTopic functions) if $c > c_0 = 2.13692...$ We study the same question for a large variety of algorithms, particularly for $(1 + \lambda)$-EA, $(\mu + 1)$-EA, $(\mu + 1)$-GA, their fast counterparts like fast $(1 + 1)$-EA, and for $(1 + (\lambda, \lambda))$-GA. We prove that all considered mutation-based algorithms show a similar dichotomy for HotTopic functions, or even for all monotone functions. For the $(1 + (\lambda, \lambda))$-GA, this dichotomy is in the parameter $c\gamma$, which is the expected number of bit flips in an individual after mutation and crossover, neglecting selection. For the fast algorithms, the dichotomy is in m_2/m_1, where m_1 and m_2 are the first and second falling moment of the number of bit flips. Surprisingly, the range of efficient parameters is not affected by either population size μ nor by the offspring population size λ.

The picture changes completely if crossover is allowed. The genetic algorithms $(\mu + 1)$-GA and $(\mu + 1)$-fGA are efficient for arbitrary mutations strengths if μ is large enough.

1 Introduction

For evolutionary algorithms (EAs), choosing a good mutation strength is a delicate matter that is subject to conflicting goals. For example, consider a pseudo-Boolean fitness function $f : \{0, 1\}^n \to \mathbb{R}$ and standard bit mutation, i.e., all bits are flipped independently. On the one hand, if the mutation strength is too low, then progress is also slow, and the algorithm will be susceptible to local optima. On the other hand, if the mutation rate is too high and the parent is already close to a global optimum, then typically the offspring, even if it has a "good" mutation in it, will also have a large number of detrimental mutations. A well-known example of this tradeoff are linear functions (e.g., OneMax), for which there is an optimal mutation rate $1/n$ [16]. This rate minimises the expected runtime, i.e., the expected number of function evaluations before the optimum is hit. Any deviation from this mutation rate to either direction decreases performance.

Extended Abstract. All proofs and further details are available on arxiv [12].

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 3–15, 2018.
https://doi.org/10.1007/978-3-319-99259-4_1

A different, more extreme example are strictly monotone pseudo-Boolean functions.[1] A function $f : \{0,1\}^n \to \mathbb{R}$ is *(strictly) monotone* if for every $x, y \in \{0,1\}^n$ with $x \neq y$ and such that $x_i \geq y_i$ for all $1 \leq i \leq n$ it holds $f(x) > f(y)$. In particular, every monotone function has a unique global optimum at $(1 \dots 1)$. Moreover, every such function is efficiently optimised by *random local search* (RLS), which is the (1+1) algorithm that flips in each round exactly one random bit. From any starting point, RLS finds the optimum after at most n improving steps, and by a coupon collector argument it will optimise any monotone function in time $O(n \log n)$. Thus, monotone functions might be regarded as trivial to optimise, and we might expect every standard EA to solve them efficiently.

However, this is not so. Doerr et al. showed [7,8] that even the $(1 + 1)$-EA, which flips each bit independently with mutation rate c/n, may have problems. More precisely, for small mutation rate, $c < 1$, the $(1 + 1)$-EA has expected runtime $O(n \log n)$, as desired, but for large mutation rate, $c > 16$, there are monotone functions for which the (1+1)-EA needs exponential time. Lengler and Steger [13] gave a simpler construction of such "hard" monotone functions, which we call HotTopic (they didn't provide a name), and which yield exponential runtime for $c > c_0 := 2.13692...$ The basic idea of this construction is that at every point in time there is some subset of bits which form a"hot topic", i.e., the algorithm considers them much more important than the other bits. An algorithm with a large mutation rate that focuses too much on the current hot topic tends to deteriorate the quality of the remaining bits. If the hot topic changes often, then the algorithm stagnates.

Since both low and high mutation rates have their disadvantages, many different strategies have been developed to gain the best of two worlds. In this paper we pick a collection of either traditional or particularly promising methods, and analyse whether they can overcome the detrimental effect of the HotTopic functions for larger mutation rates. In particular, we consider (for constant μ, λ) the classical $(1 + \lambda)$-EA, $(\mu + 1)$-EA, and $(\mu + 1)$-GA, the $(1 + (\lambda, \lambda))$-GA by Doerr et al. [6], and the recently proposed fast $(1 + \lambda)$-EA, fast $(\mu + 1)$-EA, and fast $(\mu + 1)$-GA [9], which we abbreviate by $(1 + 1)$-fEA, $(1 + \lambda)$-fEA and $(\mu + 1)$-fGA, respectively. Surprisingly, for mutation-based algorithms neither μ nor λ have any effect on the results. While we do obtain a fine-grained landscape of results (see below), one major trend is prevailing: crossover helps!

Results. In this section we collect our results for the different algorithms. Note that, unless explicitly otherwise stated, we always assume that the parameters μ, λ, c, γ of the algorithms are constant.

Classical EAs. For the classical evolutionary algorithm $(1 + \lambda)$-EA, we show a dichotomy: if the mutation parameter c is sufficiently small, then the algorithms optimise all monotone functions in time $O(n \log n)$, while for large c the algorithm needs exponential time on some HotTopic functions. The interesting question is: how does the threshold for c depend on the parameters λ? It may

[1] We will be sloppy and drop the term "strictly" outside of theorems, but throughout the paper we always mean strictly monotone functions.

seem that a large λ bears some similarity with an increased mutation rate. After all, the total number of mutations in each generation is increased by a factor of λ. Thus, we might expect that the $(1+\lambda)$-EA has difficulties with monotone functions for even smaller values of c. However, this is not so. The bounds on the mutation rate, $c < 1$ and $c > c_0$, do *not* depend on λ. In fact, for the HotTopic functions we can show that this is tight: if $c < c_0$, then the $(1+\lambda)$-EA and the $(\mu+1)$-GA optimise all HotTopic functions in time $O(n \log n)$, while for $c > c_0$ it is exponentially slow on some HotTopic instances. The same result on HotTopic holds for the $(\mu+1)$-EA. In particular, the threshold on c is also independent of μ. For the $(\mu+1)$-EA, we could not show an upper runtime bound for *all* monotone functions in the case $c < 1$, so currently we can not exclude that the situation might get even *worse* for larger μ, as there may still be other monotone functions which are hard for the $(\mu+1)$-EA with $c < 1$.

$(\mu+1)$-GA. It has been observed before that some algorithms may be sped up by crossover. In particular, Sudholt [15] showed that the $(\mu+1)$-GA is by a constant factor faster than the $(\mu+1)$-EA on OneMax. For monotone functions we also observe a change, but in extremis. We show that for the HotTopic functions crossover extends the range of mutation rate arbitrarily. For every $c > 0$, if μ is a sufficiently large constant, then the $(\mu+1)$-GA finds the optimum of HotTopic in time $O(n \log n)$. At present, there are no monotone functions known on which the $(\mu+1)$-GA with arbitrary c and large $\mu = \mu(c)$ is slow. Thus it remains an intriguing open question whether the $(\mu+1)$-GA with large μ is fast on *every* monotone function.

$(1+(\lambda,\lambda))$-GA. This algorithm creates λ offspring, and uses the best of them to perform λ biased crossovers with the parent, see Sect. 2. The best crossover offspring is then compared with the parent. This algorithm has been derived by Doerr et al. [5,6] from a theoretical understanding of so-called *black-box complexity*, and has been intensively studied thereafter [1–4]. Most remarkably, it gives an asymptotic improvement on the runtime of the most intensively studied test function OneMax, on which it has runtime roughly $n\sqrt{\log n}$ for static settings (up to $\log \log n$ terms), and linear runtime $O(n)$ for dynamic parameter settings. These runtimes are achieved with a non-constant $\lambda = \lambda(n)$. The $(1+(\lambda,\lambda))$-GA is arguably the only known natural unbiased evolutionary algorithm that can optimise OneMax faster than $\Theta(n \log n)$.

The algorithm comes with three parameters, the offspring population size λ, the mutation rate c/n by which the offspring are created, and a crossover bias γ, which is the probability to take the offspring's genes in the crossover. Again we find a dichotomy between weak and strong mutation, but this time not in c, but rather in the product $c\gamma$. In [4] it is suggested to choose c, γ in such a way that $c\gamma = 1$. Note that this makes sense, because $c\gamma$ is (neglecting possible biases by the selection process) the expected number of mutations in the crossover child. Thus it is plausible that it plays a similar role as the parameter c in classical algorithms. Indeed we find that for $c\gamma < 1$ the runtime is small for every monotone function, while for $c\gamma > c_0$ it is exponential on HotTopic functions. As before, the bound

is tight for HotTopic, i.e. for $c\gamma < c_0$ the $(1 + (\lambda, \lambda))$-GA needs time $O(n \log n)$ to optimise HotTopic.

Notably, the runtime benefits on OneMax carry over, at least to the HotTopic function. Since the benefits on OneMax in previous work have been achieved for non-constant parameter choices, we relax our assumption on constant parameters for the $(1 + (\lambda, \lambda))$-GA. In particular, we show that for the optimal static parameter and adaptive parameter settings in [9], the algorithm achieves the same asymptotic runtime on HotTopic as on OneMax, in particular runtime $O(n)$ in the adaptive setup.

Unfortunately, it seems unlikely that the runtimes of $o(n \log n)$ for OneMax carry over to arbitrary monotone functions, because they are achieved by increasing c and λ with n (although $c\gamma$ is left constant). For OneMax, if there is a zero-bit that is flipped in one of the mutations, then this mutation is always selected for crossovers. In the most relevant regime, where the expected number of flipped zero-bits in *any* mutation is small (say, at most one), the probability of being selected increases by a factor of $\Theta(\lambda)$ (from $1/\lambda$ to $\Theta(1)$) if a zero-bit is flipped. For monotone functions the probability to be selected does increase with the number of flipped zero-bits. However, there is no apparent reason that it should increase by a factor of $\Theta(\lambda)$, or by any significant factor at all.

Fast (1+1)-EA, **Fast (1+λ)-EA**, **Fast (μ+1)-EA.** These algorithms, which we abbreviate by $(1 + 1)$-fEA, $(1 + \lambda)$-fEA, and $(\mu + 1)$-fEA have recently been proposed by Doerr et al. [9], and they have immediately attracted considerable attention (e.g, [14]). The idea is to replace the standard bit mutation, in which each bit is flipped independently, by a *heavy-tailed* distribution $\mathcal{D}$. That is, in each round we draw a number s from some heavy-tailed distribution (for example, a *power-law distribution* with $\Pr[s = k] \sim k^{-\kappa}$ for some $\kappa > 1$, also called *Zipf distribution*). Then the mutation is generated from the parent by flipping exactly s bits. In this way, most mutations are generated by flipping only a small number of bits, but there is a substantially increased probability to flip many bits. This approach has given some hope to unify the best of the two worlds: of small mutation rate and of large mutation rate.

For monotone functions, our results are rather discouraging. This is not completely unexpected since the algorithms build on the very idea of increasing the probability of large mutation rates. We show a dichotomy for the $(1 + 1)$-fEA with respect to m_2/m_1, where $m_1 := \mathbb{E}[s]$ and $m_2 := \mathbb{E}[s(s-1)]$ are the first and second falling moment of the distribution $\mathcal{D}$, although the results are subject to some technical conditions.[2] As before, if $m_2/m_1 < 1$, then the runtime is $O(n \log n)$ for all monotone functions. On the other hand, if $m_2/m_1 > c_0$ and additionally $p_1 := \Pr[\mathcal{D} = 1]$ is sufficiently small, then the runtime on some HotTopic instances is exponential. As for the other functions, we get a sharp threshold for the parameter regime that is efficient on HotTopic, so we can decide for each distribution whether it leads to fast or to exponential runtimes on HotTopic. Due to a correction term related to p_1 (Eq. (5) on page 9), it

[2] Note that a heavy tail generally increases m_2 much stronger than m_1, so it increases the quotient m_2/m_1.

is possible to construct heavy-tail distributions which are efficient on all HotTopic functions, but they must be chosen with great care. For example, no power-law distribution with exponent $\kappa \in (1,2)$ is efficient, which includes the choice $\kappa = 1.5$ that is used for experiments in [9,14]. Also, no distribution with $p_1 < \frac{4}{9}\Pr[\mathcal{D} = 3]$ is efficient on HotTopic. In general, our findings contrast the results in [9], where larger tails (smaller κ) lead to faster runtimes.

As before, without crossover larger values of λ and μ do not seem to have any effect. For the $(1+\lambda)$-fEA and $(\mu+1)$-fEA, we show exactly the same results as for the $(1+1)$-fEA, except that we could not show runtime bounds for *all* monotone functions if $m_2/m_1 < 1$. Rather, we only show them for HotTopic. Thus it is still possible that larger values of λ, μ make things even worse.

Fast $(\mu+1)$-GA. As for the classical algorithms, crossover tremendously improves the situation. For every distribution $\mathcal{D}$ with $\Pr[\mathcal{D} = 1] = \Omega(1)$, if μ is a sufficiently large constant, then the $(\mu+1)$-fGA optimises HotTopic in time $O(n \log n)$. As for the $(\mu+1)$-GA, it is an open question whether the same result carries over to *all* monotone functions.

Further Results. For all algorithms, the regime of exponential runtime does not just mean that it is hard to find the optimum, but rather the algorithms do not even come close. More precisely, in all these cases there is an $\varepsilon > 0$ (depending only on c or on the other dichotomy parameters) such that the probability that any of the EAs or GAs finds a search point with at least $(1-\varepsilon)n$ correct bits within a subexponential time is exponentially small as $n \to \infty$. The size of ε can be quite considerable if the parameter c is much larger than c_0. For example, simulations suggest for the $(1+1)$-EA that $\varepsilon \approx 0.15$ for $c = 4$. On the other hand, starting close to the optimum does not help either: for every $\varepsilon > 0$ there are monotone function such that if the EAs or GAs are initialised with random search points with εn incorrect bits, then still the runtime is exponential.

Summary. It appears that increasing the number of offspring λ or the population size μ does not help at all to overcome the detrimental effects of large mutation rate in evolutionary algorithms. All EAs are highly vulnerable even to a very moderate increase of the mutation rate. Using heavy tails as in the fEAs seems to make things even worse, although the picture gets more complicated. On the other hand, using crossover can remedy the effect of large mutation rates, and can extend the range of good mutation rates arbitrarily.

Intuition on HotTopic. We conclude the introduction with an intuition why the HotTopic functions are hard to optimise for large mutation rates. Note that a monotone function, by its very definition, has a local "gradient" that always points into the same corner of the hypercube, in the sense that for each bit individually, in all situations we prefer a one-bit over a zero-bit. The construction by Lengler and Steger [13] distorts the gradient by assigning different positive weights to the components. Such a distortion cannot alter the direction of the gradient by too much. In particular, following the gradient will always decrease the distance from the optimum. This is why algorithms with small mutation rate may find the optimum; they follow the gradient relatively closely. However, the weights in [13] are

chosen such that there is always a"hot topic", i.e., a subdirection of the gradient which is highly preferred over all other directions. Focusing too much on this hot topic will lead to a behaviour that is very good at optimising this particular aspect – but all other aspects will deteriorate a little because they are out of focus. Thus if the hot topic is rather narrow and changes often, then advances in this aspect will be overcompensated by a decline in the neglected parts, which leads overall to stagnation.

This last sentence is not just a pessimistic allegory on scientific progress, but it also describes evolutionary algorithms with large mutation rates. They rank the currently preferred direction above everything else, and accept any mutation that makes progress in that direction, regardless of the harm that such a mutation may cause on other bits. This may lead to a drift away from the optimum, since random walk steps naturally tend to increase the distance from the optimum. For the fEAs or fGAs, this effect is amplified if the algorithm is close to the optimum. In this case, the probability to find any improvement at all is small, and improvements often occur in aggressive steps in which many bits are flipped. Then the same step may cause many errors among the low-priority bits. For the same reason, an adaptive choice of the mutation strength c may be harmful if it increases the mutation parameter in phases of stagnation: close to the optimum, most steps are stagnating, so an adaptive algorithm might react by increasing the mutation parameter. This indeed increases the probability to find a better search point in the hot topic direction (though not the probability to make *any* improvement), and may thus lead fatally to a large mutation parameter.

2 Preliminaries and Definitions

Notation. Throughout the paper we will assume that $f : \{0,1\}^n \to \mathbb{R}$ is a monotone function, i.e., for every $x, y \in \{0,1\}^n$ with $x \neq y$ and such that $x_i \geq y_i$ for all $1 \leq i \leq n$ it holds $f(x) > f(y)$.[3] We will consider maximisation algorithms, and we will mostly focus on the *runtime* of an algorithm, i.e., the number of function evaluations before the algorithm evaluates the global maximum of f for the first time. We say that an EA or GA is *elitist* [10] if the selection operator greedily chooses the fittest individuals to form the next generation. We call an EA or GA *unbiased* [11] if the mutation and crossover are invariant under the isomorphisms of $\{0,1\}^n$, i.e., if mutation and crossover are symmetric with respect to the ordering of the bits, and with respect to exchange of the values 0 and 1. All algorithms considered in this paper are unbiased.

For $n \in \mathbb{N}$, we denote $[n] := \{1, \dots, n\}$. We will use n for the dimension of the search space, μ for the population size, λ for the offspring population size, c for the mutation parameter, γ for the crossover parameter of the $(1+(\lambda,\lambda))$-GA, and $\mathcal{D}, m_1, m_2$ for the bit flip distribution of the fast EAs and GAs and its first

[3] Note that this property might more correctly be called *strictly monotone*, but in this paper we will stick with the shorter, slightly less precise term *monotone*. In all other cases we use the standard terminology, e.g. the term *increasing sequence* has the same meaning as *non-decreasing sequence*.

and second falling moment $\mathbb{E}[s \mid s \sim \mathcal{D}]$ and $\mathbb{E}[s(s-1) \mid s \sim \mathcal{D}]$, respectively. Unless otherwise stated, we will assume that $\mu, \lambda, c, \gamma = \Theta(1)$ and $m_1 = \Omega(1)$.

Algorithms. Most algorithms that we consider fall into the class of $(\mu + \lambda)$ evolutionary algorithms, $(\mu + \lambda)$-*EAs*, or $(\mu + \lambda)$ genetic algorithms, $(\mu + \lambda)$-*GAs*. They can be described as follows. They maintain a population of size μ. In each *generation*, λ additional offspring are created by *mutation* and possibly *crossover*, and the μ search points of highest fitness among the $\mu + \lambda$ individuals form the next generation. Thus we use an *elitist selection* scheme. In EAs, the offspring are only created by *mutation*, in GAs they are either created by mutation or by crossover. For mutation we use standard bit mutation as a default, in which each bit is independently flipped with probability c/n, where c is the *mutation parameter*. The only exception are the *fast* EAs and GAs, in which first the number s of bit mutations is drawn from some distribution $\mathcal{D} = \mathcal{D}(n)$, and then exactly s bits are flipped, chosen uniformly at random. We will always assume that $\mu, \lambda, c = \Theta(1)$.

An exception to the above scheme is the $(1 + (\lambda, \lambda))$-GA [5]. Here the population consists of a single search point x. Then in each round, we pick $s \sim \textsc{Bin}(n, c/n)$, and create λ offspring from x by flipping exactly s bits in x uniformly at random. Then we select the fittest offspring y among them, and we perform λ independent biased crossover between x and y, where for each bit we take the parent gene from y with probability γ, and the gene from x otherwise. If the best of these crossover offspring is at least as fit as x, then it replaces x. We will usually assume that $\lambda, c, \gamma = \Theta(1)$, unless otherwise mentioned.

Hard Monotone Functions: HotTopic. In this section we give the construction of hard monotone functions by Lengler and Steger [13], following closely their exposition. The functions come with four parameters $\alpha, \beta, \rho, \varepsilon$, and they are given by a randomised construction. We call the corresponding function $\textsc{HotTopic}_{\alpha,\beta,\rho,\varepsilon} = \mathrm{HT}_{\alpha,\beta,\rho,\varepsilon} = \mathrm{HT}$. The hard regime of the parameters is

$$1 > \alpha \gg \varepsilon \gg \beta \gg \rho > 0, \tag{1}$$

by which we mean that $\alpha \in (0,1)$ is a constant, $\varepsilon = \varepsilon(\alpha)$ is a sufficiently small constant, $\beta = \beta(\alpha, \varepsilon)$ is a sufficiently small constant, and $\rho = \rho(\alpha, \varepsilon, \beta)$ is a sufficiently small constant.

Now we come to the construction. For $1 \le i \le e^{\rho n}$ we choose sets $A_i \subseteq [n]$ of size αn independently and uniformly at random, and we choose subsets $B_i \subseteq A_i$ of size βn uniformly at random. We define the *level* $\ell(x)$ of a search point $x \in \{0,1\}^n$ by $\ell(x) := \max\{\ell' \in [e^{\rho n}] : |\{j \in B_{\ell'} : x_j = 0\}| \le \varepsilon\beta n\}$, where we set $\ell(x) = 0$, if no such ℓ' exists). Then we define $f : \{0,1\}^n \to \mathbb{R}$ as follows:

$$\mathrm{HT}(x) := \ell(x) \cdot n^2 + \sum\nolimits_{i \in A_{\ell(x)+1}} x_i \cdot n + \sum\nolimits_{i \notin A_{\ell(x)+1}} x_i, \tag{2}$$

where for $\ell = e^{\rho n}$ we set $A_{\ell+1} := B_{\ell+1} := \emptyset$.

So the set $A_{\ell+1}$ defines the hot topic while the algorithm is at level ℓ, where the level is determined by the sets B_i. It was shown in [13] that whp[4] the $(1+1)$-EA with $c > c_0$ needs exponential time to find the optimum. One easily checks that this function is monotone: the (monotone) term $\ell(x)n^2$ dominates the rest, and for constant values of ℓ the remaining terms just give a linear function.

3 Results

We first give a generic result for strong dichotomies, i.e., we specify circumstances under which an algorithm optimises every monotone function in time $O(n \log n)$.

Theorem 1 (Generic Easiness Proof). *Consider an elitist algorithm $\mathcal{A}$ with population size one that in each round generates an offspring by an arbitrary method, and replaces the parent if and only if the offspring has at least the same fitness. Let s_{01} denote the number of zero-bits in the parent that are one-bits in the offspring, and vice versa for s_{10}. Assume that there is a constant $\delta > 0$ such that for all $x \in \{0,1\}^n$,*

$$\mathbb{E}[s_{10} \mid \textit{parent} = x \textit{ and } s_{01} > 0] \leq 1 - \delta, \tag{3}$$

and

$$\Pr[s_{01} > 0 \mid \textit{parent} = x] = \Omega(\tfrac{1}{n}(n - \textsc{OneMax}(x))). \tag{4}$$

Then whp the runtime of $\mathcal{A}$ on any (strictly) monotone functions is $O(n \log n)$.

The $(1+\lambda)$-EA, the $(1+1)$-fEA, and the $(1+(\lambda,\lambda))$-GA all fit the generic description in Theorem 1, modulo Condition (3). For the $(1+(\lambda,\lambda))$-GA, the procedure to generate the offspring is rather complicated, and involves several intermediate mutation and crossover steps. Nevertheless, the procedure ultimately produces a single offspring (the fittest of the crossover offspring) which competes with the parent. The crucial step is in all cases to show that these settings satisfy (3). For the $(1+(\lambda,\lambda))$-GA with $c\gamma < 1$ and non-constant parameters we cannot apply Theorem 1 directly. However, the proof is similar, and the conditional expectation in (3) is still the crucial object to study.

Theorem 2. *Let $\delta > 0$. The following algorithms need whp $O(n \log n)$ generations on any (strictly) monotone function.*

- *The $(1+\lambda)$-EA with $c \leq 1-\delta$, $c = \Omega(1)$ and $\lambda = O(1)$;*
- *the $(1+1)$-fEA with $m_2/m_1 \leq 1-\delta$ and $m_1 = \Omega(1)$;*
- *the $(1+(\lambda,\lambda))$-GA with $c\gamma \leq 1-\delta$ and $c\gamma = \Omega(1)$.*

[4] With high probability, i.e. with probability tending to one as $n \to \infty$.

Moreover, if the $(1 + (\lambda, \lambda))$*-GA with* $c\gamma < 1 - \delta$ *uses the optimal static or adaptive parameter choice from [4][5], then whp the runtime on* HotTopic *is up to a factor* $\Theta(1)$ *the same as the runtime for* OneMax.

We remark that the optimal runtime of the $(1 + (\lambda, \lambda))$-GA on OneMax is $O(n\sqrt{\log(n)\log\log\log(n)/\log\log n})$ for static parameters, and $O(n)$ for adaptive parameter choices [4,6].

Our next theorem gives upper bounds on the runtime of the $(1 + \lambda)$-fEA on any monotone function, provided that $m_2/m_1 < 1$, where m_1 and m_2 are the first and second falling moments of the distribution $\mathcal{D}$. We need to make the assumption that the algorithm starts at most in distance εn to the optimum. It is unclear whether this assumption is necessary, or an artefact of our proof.

Theorem 3. *Let* $\delta > 0$ *be a constant, let* $\lambda = O(1)$*, and consider the* $(1 + \lambda)$*-fEA with distribution* $\mathcal{D} = \mathcal{D}(n)$*, whose falling moments* m_1, m_2 *satisfy* $m_2/m_1 \leq 1-\delta$ *and* $m_1 = \Omega(1)$*. Then there is* $\varepsilon > 0$ *such that the* $(1+\lambda)$*-fEA starting with any search point with at most* εn *zero-bits finds the optimum of every (strictly) monotone functions in time* $O(n \log n)$ *whp.*

Next we analyse the behaviour of a generic algorithm on HotTopic, which will later serve as basis for all of our results on HotTopic for concrete algorithms. The generic algorithm uses population size one, but we will show that, surprisingly, $(\mu + 1)$ algorithm can be described by the same framework.

Theorem 4 (HotTopic, Generic Runtime). *Let* $0 < \alpha < 1$*. Consider an elitist, unbiased optimisation algorithm* $\mathcal{A}$ *with population size one that starts with a random search point* x *and in each round generates an offspring* y *by an arbitrary (unbiased) method, and replaces the parent* x *by* y *if* $\mathrm{HT}(y) > \mathrm{HT}(x)$*. For equal fitness, it may decide arbitrarily whether it replaces the parent. Let* s *be the random variable that denotes the total number of bits in which parent and offspring differ, and note that the distribution of* s *may depend on the parent. For parent* x*, we define*

$$\Phi(x) := \frac{\mathbb{E}[s(s-1)(1-\alpha)^{s-1}]}{\mathbb{E}[s(1-\alpha)^{s-1}]} - \frac{((1-\alpha)/\alpha) \cdot \Pr[s=1]}{\mathbb{E}[s(1-\alpha)^{s-1}]}. \tag{5}$$

(a) If there are constants $\zeta, \zeta' > 0$ *such that*

$$\Phi(x) \geq 1 + \zeta' \tag{6}$$

holds for all $x \in \{0,1\}^n$ *with at most* ζn *zero-bits, then whp* $\mathcal{A}$ *has exponential runtime on* $\mathrm{HotTopic}_{\alpha,\beta,\rho,\varepsilon}$ *with parameters* β, ρ, ε *as in* (1)*.*

[5] In fact, the suggested parameter choice in [4,6] satisfies $c\gamma = 1$ instead of $c\gamma < 1$. However, the runtime analysis in [6] only changes by constant factors if γ is decreased by a constant factor. Thus Theorem 2 applies to the parameter choices from [4,6], except that γ is decreased by a constant factor.

(b) If there are constants $\zeta, \zeta' > 0$ such that

$$\Phi(x) \leq 1 - \zeta' \tag{7}$$

holds for all $x \in \{0,1\}^n$ with at most ζn zero-bits, and if moreover $\Pr[s = 1] \geq \zeta$ and $\mathbb{E}[s(s-1)] \leq 1/\zeta$ for all parents x, then whp $\mathcal{A}$ has runtime $O(n \log n)$ on $\textsc{HotTopic}_{\alpha,\beta,\rho,\varepsilon}$ with parameters β, ρ, ε as in (1).

(c) The statements in (a) and (b) remain true for algorithms that are only unbiased conditioned on an improving step, if in (b) we have $\Pr[\text{improving step}] \geq \zeta \cdot d([n], x)$ as well. Moreover, (b) remains true for algorithms that are only unbiased if x has more than ζn zero-bits, and possibly biased for at most ζn zero bits, if we replace (7) by the condition $\mathbb{E}[s \mid \mathrm{HT}(y) > \mathrm{HT}(x)] \leq 2 - \zeta$.

Finally, there is a constant $\eta = \eta(\zeta', \alpha) > 0$ independent of ζ such that (a), (b), and (c) remain true in the presence of the following adversary A. Whenever an offspring x' is created from x that satisfies $f(x') > f(x)$, then A flips a coin. With probability $1 - \eta$, she does nothing. Otherwise, she draws an integer $\tau \in \mathbb{N}$ with expectation $O(1)$ and she may change up to τ bits in the current search point. For (a) we additionally require $\Pr[\tau \geq \tau'] = e^{-\Omega(\tau')}$, while for (b) and (c) we only require $\Pr[\tau \geq n^{1-\eta}] = o(1/(n \log n))$.

We remark that (b) and (c) require parameters as in (1), and thus do not exclude a large runtime on HotTopic for atypical parameters, e.g., for large ε.

It turns out that Theorem 4 suffices to classify the behaviour on HotTopic for all algorithms that we study. On the first glance, this may seem surprising, since some of them are population-based, while Theorem 4 explicitly requires population size one. However, for small ε the populations typically collapse to μ identical copies of the same search point, and the other cases can be attributed to the adversary. In this way Theorem 4 implies the following theorem.

Theorem 5 (HotTopic, Concrete Results). *Let $\delta > 0$. We assume that $\mu, \lambda, c = \Theta(1)$ and $\Pr[\mathcal{D} = 1] = \Omega(1)$, except for the $(1 + (\lambda, \lambda))$-GA, for which we replace the condition on c by $c\gamma = \Theta(1)$. Let $c_0 = 2.13692..$ be the smallest constant for which the function $c_0 x - e^{-c_0(1-x)} - x/(1-x)$ has a solution $\alpha \in [0,1]$. For all $\alpha \in (0,1)$, whp each of the following algorithms optimises the function $\textsc{HotTopic}_{\alpha,\beta,\rho,\varepsilon}$ with parameters β, ρ, ε as in (1) in time $O(n \log n)$.*

- *The $(1+\lambda)$-EA with $c \leq c_0 - \delta$.*
- *The $(\mu+1)$-EA with $c \leq c_0 - \delta$.*
- *The $(\mu+1)$-GA with arbitrary $c = \Theta(1)$ if $\mu = \mu(c)$ is sufficiently large.*
- *The $(1 + (\lambda, \lambda))$-GA with $c\gamma \leq c_0 - \delta$.*
- *The $(1+\lambda)$-fEA with $m_2/m_1 \leq 1 - \delta$; more generally, the $(1+\lambda)$-fEA with any distribution that satisfies (7) for $s \sim \mathcal{D}$, as well as $\Pr[\mathcal{D} = 1] = \Omega(1)$.*[6]

[6] Note that this is not a trivial consequence of Theorem 4, since (6), (7) are conditions on the distribution for the best of λ offspring, while the condition here is on the distribution $\mathcal{D}$ for generating a single offspring.

- *The* $(\mu+1)$*-fEA in the preceding case, if additionally* $\Pr[\mathcal{D}=0]=\Omega(1)$.
- *The* $(\mu+1)$*-fGA with arbitrary* $\mathcal{D}$ *with* $\Pr[\mathcal{D}=0]=\Omega(1)$*, if* $\mu=\mu(\mathcal{D})$ *is sufficiently large.*

On the other hand, for $\alpha_0 = 0.237134..$*, whp each of the following algorithms needs exponential time to optimise the function* $\textsc{HotTopic}_{\alpha_0,\beta,\rho,\varepsilon}$ *with parameters* β,ρ,ε *as in* (1).

- *The* $(1+\lambda)$*-EA with* $c \geq c_0+\delta$.
- *The* $(\mu+1)$*-EA with* $c \geq c_0+\delta$.
- *The* $(\mu+1)$*-GA with* $c \geq c_0+\delta$ *if* $\mu=\mu(c)$ *is sufficiently small.*[7]
- *The* $(1+(\lambda,\lambda))$*-GA with* $c\gamma \geq c_0+\delta$.
- *The* $(1+\lambda)$*-fEA with any distribution satisfying* (6) *for* $s\sim\mathcal{D}$.[6] *In particular, this includes the following cases.*
  - *The* $(1+\lambda)$*-fEA with* $m_2/m_1 \geq 1+\delta$*, if* $\Pr[\mathcal{D}=1] \geq C/s_0$ *for a sufficiently large constant* $C>0$*, where* $s_0 := \min\{\sigma\in\mathbb{N} \mid m_{2,\leq\sigma} \geq (1+\delta/2)m_1\}$*, with* $m_{2,\leq\sigma} := \sum_{i=1}^{\sigma}\Pr[\mathcal{D}=i]i(i-1)$.
  - *The* $(1+\lambda)$*-fEA with any power law distribution with exponent* $\kappa\in(1,2)$*, i.e.* $\Pr[\mathcal{D}\geq\sigma]=\Omega(\sigma^{-\kappa})$.
  - *The* $(1+\lambda)$*-fEA with* $\Pr[\mathcal{D}=1] \leq \frac{4}{9}\cdot\Pr[\mathcal{D}\geq 3]-\delta$.
- *The* $(\mu+1)$*-fEA in all preceding cases for* $(1+\lambda)$*-fEA, if* $\Pr[\mathcal{D}=0]=\Omega(1)$.
- *The* $(\mu+1)$*-fGA in all preceding cases for* $(1+\lambda)$*-fEA if* $\mu=\mu(\mathcal{D})$ *is sufficiently small.*[7]

Remark 1. For the fEAs we remark that the interesting regime $\kappa\in[2,3)$ is not excluded by the negative results in Theorem 5, if $\Pr[\mathcal{D}]$ is sufficiently large. In particular, a calculation with Mathematica$^{\text{TM}}$ shows that the Zipf distribution[8] with exponent $\kappa\geq 2$ satisfies (7) for all $\alpha\in(0,1)$. However, note that this holds only if the distribution is *exactly* the Zipf distribution; changing any probability even by a constant factor may lead to exponential runtimes.

4 Conclusions

We have studied a large set of algorithms, and we have shown that in all cases without crossover, there is a dichotomy with respect to a parameter (c, $c\gamma$, or Φ, where the latter one is related to m_2/m_1) for optimising the monotone function family HotTopic. If the parameter is small, then the algorithms need time $O(n\log n)$; if the parameter is large, then they need exponential time on some instances. In the cases of $(1+\lambda)$-EA, $(1+1)$-fEA $(1+(\lambda,\lambda))$-GA, and for good start points also of $(1+\lambda)$-fEA, if the parameter is small, then we could show that the algorithms are actually fast on *all* monotone functions. However, there are many open problems left, and we conclude the paper by a selection of those.

[7] This statement follows trivially from the other results by setting $\mu=1$, and it is listed only for completeness.

[8] i.e., $\Pr[\mathcal{D}=k]=k^{-\kappa}/\zeta(\kappa)$, where ζ is the Riemann ζ function.

- We have analysed the algorithms theoretically for the case $n \to \infty$. Experiments are sorely needed to understand the effects for small finite n.
- In some cases our runtime bounds for small parameter values hold only for HotTopic, but the general status of monotone functions remains unclear ($(\mu+1)$-EA, $(\mu+1)$-fEA). So does a small mutation parameter guarantee a small runtime on *all* monotone functions?
- We could show that genetic algorithms are superior to evolutionary algorithms on the HotTopic functions. However, is the same true in general for monotone functions? Is it true that the $(\mu+1)$-GA and the $(\mu+1)$-fGA are fast for all monotone functions if μ is large enough?
- It seems important to understand more precisely how large μ should be in GAs to cope with larger mutation parameters. For example, for the $(\mu+1)$-GA with mutation parameter c, how large does m need to be so that it is still fast on all HotTopic instances?
- By now a classical question is: are there monotone functions which are hard for the parameter range $[1, c_0)$? Most intriguingly: are there hard monotone instances for the $(1+1)$-EA for every $c > 1$? For $c = 1$ it is known that the runtime is polynomial, but is it always $O(n \log n)$?
- Our proofs for population sizes $\mu > 1$ rely on the fact that in all considered algorithms diversity tends to be lost close to the optimum. Do the results stay the same if diversity is actively maintained, for example by duplication avoidance or by genotypical or phenotypical niching?
- How is the performance of algorithms that change the mutation strength dynamically, e.g., with the 1/5-th rule? The introduction gives an intuition why this might be bad, but intuition has failed before on monotone functions.
- While HotTopic is defined in a discrete setting, the underlying intuition is related to continuous optimisation. Is there a continuous analogue of HotTopic, and what is the performance of optimisation algorithms like the CMA-ES or particle swarm optimisation?

References

1. Doerr, B.: Optimal parameter settings for the $(1 + \lambda, \lambda)$ genetic algorithm. In: GECCO (2016)
2. Doerr, B., Doerr, C.: Optimal parameter choices through self-adjustment: applying the 1/5-th rule in discrete settings. In: GECCO (2015)
3. Doerr, B., Doerr, C.: A tight runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on OneMax. In: GECCO (2015)
4. Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1+(\lambda, \lambda))$ genetic algorithm. Algorithmica **80**, 1–52 (2017)
5. Doerr, B., Doerr, C., Ebel, F.: Lessons from the black-box: fast crossover-based genetic algorithms. In: GECCO (2013)
6. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theor. Comput. Sci. **567**, 87–104 (2015)
7. Doerr, B., Jansen, T., Sudholt, D., Winzen, C., Zarges, C.: Optimizing monotone functions can be difficult. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN 2010. LNCS, vol. 6238, pp. 42–51. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_5

8. Doerr, B., Jansen, T., Sudholt, D., Winzen, C., Zarges, C.: Mutation rate matters even when optimizing monotonic functions. Evol. Comput. **21**(1), 1–27 (2013)
9. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: GECCO (2017)
10. Doerr, C., Lengler, J.: Introducing elitist black-box models: when does elitist behavior weaken the performance of evolutionary algorithms? Evol. Comput. **25**(4), 587–606 (2017)
11. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**, 623–642 (2012)
12. Lengler, J.: A general dichotomy of evolutionary algorithms on monotone functions. arXiv e-prints (2018)
13. Lengler, J., Steger, A.: Drift analysis and evolutionary algorithms revisited. arXiv e-prints (2016)
14. Mironovich, V., Buzdalov, M.: Evaluation of heavy-tailed mutation operator on maximum flow test generation problem. In: GECCO (2017)
15. Sudholt, D.: How crossover speeds up building block assembly in genetic algorithms. Evol. Comput. **25**(2), 237–274 (2017)
16. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Comb. Probab. Comput. **22**(2), 294–318 (2013)

Artificial Immune Systems Can Find Arbitrarily Good Approximations for the NP-Hard Partition Problem

Dogan Corus(✉), Pietro S. Oliveto, and Donya Yazdani

Rigorous Research, University of Sheffield, Sheffield, UK
{d.corus,p.oliveto,dyazdani1}@sheffield.ac.uk

Abstract. Typical Artificial Immune System (AIS) operators such as hypermutations with mutation potential and ageing allow to efficiently overcome local optima from which Evolutionary Algorithms (EAs) struggle to escape. Such behaviour has been shown for artificial example functions such as Jump, Cliff or Trap constructed especially to show difficulties that EAs may encounter during the optimisation process. However, no evidence is available indicating that similar effects may also occur in more realistic problems. In this paper we perform an analysis for the standard NP-Hard Partition problem from combinatorial optimisation and rigorously show that hypermutations and ageing allow AISs to efficiently escape from local optima where standard EAs require exponential time. As a result we prove that while EAs and Random Local Search may get trapped on 4/3 approximations, AISs find arbitrarily good approximate solutions of ratio $(1+\epsilon)$ for any constant ϵ within a time that is polynomial in the problem size and exponential only in $1/\epsilon$.

1 Introduction

Artificial Immune Systems (AIS) take inspiration from the immune system of vertebrates to solve complex computational problems. Given the role of the natural immune system to recognise and protect the organism from viruses and bacteria, natural applications of AIS have been pattern recognition, computer security, virus detection and anomaly detection [1–3]. Various AIS, inspired by Burnet's clonal selection principle, have been devised for solving optimisation problems. Amongst these, the most popular are Clonalg [4], the B-Cell algorithm [5] and Opt-IA [6].

AIS for optimisation are very similar to evolutionary algorithms (EAs) since they essentially use the same Darwinian evolutionary principles to evolve populations of solutions (here called antibodies). In particular, they use the same natural selection principles to gradually evolve high quality solutions. The main distinguishing feature of AIS to more classical EAs is their use of variation operators that typically have higher mutation rates compared to the standard bit mutations (SBM) of EAs (such as the contiguous somatic mutations of the B-Cell algorithm and the hypermutations with mutation potential of Opt-IA) and

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 16–28, 2018.
https://doi.org/10.1007/978-3-319-99259-4_2

their use of ageing operators that remove old solutions i.e., that have spent a long time without improving (local optima). Despite their popularity it is still largely unclear on what problems an AIS will have better performance to that of EAs. Also very little guidance is available on when a class of AIS should be applied rather than another. Amongst the few available results, it has been proven that there exist instance classes of both vertex cover [7] and the longest common subsequence [8] NP-Hard problems that are hard for EAs equipped with SBM and crossover for which the B-Cell algorithm is efficient The superior performance is due to the ability of the contiguous somatic mutations of the B-Cell algorithm to efficiently escape the local optima of these instances while SBM require exponential expected time in the size of the instance.

Apart from these results, the theoretical understanding of AIS relies on analyses of their behaviour for artificially constructed toy problems. Recently it has been shown how both the hypermutations with mutation potential and the ageing operator of Opt-IA can lead to considerable speed-ups compared to the performance of well-studied EAs using SBM for standard benchmark functions used in evolutionary computation such as Jump, Cliff or Trap [9]. While the performance of hypermutation operators to escape the local optima of these functions is comparable to that of the EAs with high mutation rates that have been increasingly gaining popularity since 2009 [10–14], ageing allows the optimisation of hard instances of Cliff in $O(n \log n)$, a runtime that is required by the SBM of typical EAs to optimise any function with unique optimum i.e., SBM require $\Theta(n \log n)$ to optimise its easiest function with unique optimum - OneMax [15]. Although some of these speed-ups over standard EA performance are particularly impressive, no similar evidence of superior performance of these operators is available for more realistic problems.

In this paper we perform an analysis for Partition, a classical NP-Hard combinatorial optimisation problem for which the performance of Random Local Search (RLS) and of the $(1+1)$ EA is understood [16–18]. Since Partition is essentially a makespan scheduling problem with two machines, its relevance to practical applications can be easily seen. It is well understood that both RLS and the $(1+1)$ EA may get stuck on local optima which lead to a worst case approximation ratio of $4/3$. In order to achieve a $(1+\epsilon)$ approximation for arbitrary ϵ a clever restart strategy has to be put in place. Herein, we first show the power of hypermutations and ageing by proving that each of them solve to optimality instances that are hard for RLS and SBM EAs, by efficiently escaping local optima for which the EAs struggle. Afterwards we prove that AIS using hypermutations with mutation potential guarantee arbitrarily good solutions of approximation ratio $(1+\epsilon)$ in expected time $n(\epsilon^{-(2/\epsilon)-1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon} + n^3 2^{2/\epsilon} + n^3$, which reduces to $O(n^3)$ for any constant ϵ without requiring any restarts. On the other hand, we prove that an AIS with SBM and ageing can efficiently achieve the same approximation ratio in $O(n^2)$, automatically restarting the optimisation process by implicitly detecting when it is stuck on a local optimum. To the best of our knowledge this is the first time either hypermutations or ageing have been theoretically analysed for a standard problem from combinatorial optimi-

Algorithm 1. (1+1) IA^{hyp} [9] for minimisation

1: Set each bit in x to 1 with probability 1/2 and to 0 otherwise, then evaluate $f(x)$.
2: **while** termination condition not satisfied **do**
3: $y := x$; $Flip := \{1, \ldots, n\}$;
4: **while** $Flip \neq \emptyset$ and $f(y) \geq f(x)$ **do**
5: $i :=$ Sample $Flip$ u. a. r.; $Flip := Flip \setminus i$; flip y_i; evaluate $f(y)$;
6: **end while**
7: If $f(y) \leq f(x)$, then $x := y$.
8: **end while**

sation and the first time performance guarantees of any AIS are proven in such a setting.

Due to space limitations some proofs are omitted for this extended abstract[1].

2 Preliminaries

Hypermutation with mutation potential operators are inspired by the high mutation rates occurring in the natural immune system [6]. For the purpose of optimisation, these high mutation rates may allow the algorithm to escape local optima by identifying promising search areas far away from the current ones.

In this paper we will analyse the static hypermutation operator considered in [9] for benchmark functions, where the maximum number of bits to be flipped is fixed to $M = cn$ throughout the optimisation process. Two variants of static hypermutations have been proposed in the literature. A straightforward version, where in each mutation exactly cn bits are flipped, and another one called *stop at first constructive mutation* (FCM) where the solution quality is evaluated after each of the cn bit-flips and the operator is halted once a *constructive mutation* occurs. Since Corus et al. [9] proved that the straightforward version requires exponential time to optimise any function with a polynomial number of optima, we will consider the version with FCM. We define a mutation to be *constructive* if the solution is strictly better than the original parent and we set $c = 1$ such that all bits will flip if no constructive mutation is found before. For the sake of understanding the potentiality of the operator, we embed it into a minimal AIS framework that uses only one antibody (or individual) and creates a new one in each iteration via hypermutation as done previously in the literature [9]. The algorithm is essentially a $(1 + 1)$ EA [20,21] that uses hypermutations instead of SBM. The simple AIS for the minimisation of objective functions, called $(1 + 1)$ IA^{hyp} for consistency with the literature, is formally described in Algorithm 1.

Another popular operator used in AIS is *Ageing*. The idea behind the operator is to remove antibodies which have not improved for a long time. Intuitively, these antibodies are not improving because they are trapped on some local optimum, and they may be obstructing the algorithm from progressing in more promising areas of the search space (i.e., the population of antibodies may

[1] A complete version of the paper including all the proofs is available on arXiv [19].

Algorithm 2. $(\mu + 1)$ EA^{ageing} [22] for minimisation

1: Create population $P := \{x_1, \cdots, x_\mu\}$ with each bit in x_i set to 1 with probability $1/2$ and to 0 otherwise;
2: For all $x \in P$ evaluate $f(x)$ and set $x[age] := 0$.
3: **while** termination condition not satisfied **do**
4: For all $x \in P$ set $x[age] := x[age] + 1$.
5: Select $x \in P$ uniformly at random;
6: $y := x$ and flip each bit in y with probability $1/n$. Add y to P.
7: If $f(y) < f(x)$, then $y[age] := 0$. Else, $y[age] := x[age]$;
8: For all $x \in P$ if $x[age] \geq \tau$ then remove x from P;
9: If $|P| > \mu$ then remove the individual from P with the highest $f(x)$;
10: If $|P| < \mu$ then add enough number of randomly created individuals to P until $|P| = \mu$;
11: **end while**

quickly be taken over by a high quality antibody on a local optimum). The antibodies that have been removed by the ageing operator are replaced by new ones initialised at random.

Ageing operators have been proven to be very effective at automatically restarting the AIS, without having to set up a restart strategy in advance, once it has converged on a local optimum [23]. Stochastic versions have been shown to also allow antibodies to escape from local optima [9,22]. As in previous analyses we incorporate the ageing operator in a simple $(\mu+1)$ EA algorithmic framework and for simplicity consider the static variant where antibodies are removed from the population with probability 1 if they have not improved for τ generations. The algorithm is formally defined in Algorithm 2. We will compare the performance of the AIS with the standard $(1 + 1)$ EA [21,24] and RLS for which the performance for Partition is known. The former uses standard bit mutation i.e., it flips each bit of the parent with probability $1/n$ in each iteration, while the latter flips exactly one bit.

Given n jobs with processing times $p_1, \ldots, p_n$ and $p_i > 0$, the Partition problem is that of scheduling the jobs on two identical machines, M_1 and M_2, in a way that the overall completion time (i.e., the *makespan*) is minimised. This simple to define scheduling problem is well studied in theoretical computer science and is known to be NP-Hard [25]. Hence, it cannot be expected that any algorithm finds exact solutions to all instances in polynomial time. However, there exist efficient problem specific algorithms which guarantee solutions with approximation ratio $(1 + \epsilon)$ in time $O(n^3/\epsilon)$, in classical complexity measures, which is polynomial both in n and $1/\epsilon$ [26]. Let f_A be the solution quality guaranteed by algorithm A and f_{opt} be the value of the optimal solution. Then the approximation ratio for a minimisation problem is defined as f_A/f_{opt}.

For the application of randomised search heuristics a solution may easily be represented with a bitstring $x \in \{0,1\}^n$ where each bit i represents job i and if the bit is set to 0 it is assigned to the first machine (M_1) and otherwise to the second machine (M_2). Hence, the goal is to minimise

$f(x) := \max\left\{\sum_{i=1}^{n} p_i x_i, \sum_{i=1}^{n} p_i(1-x_i)\right\}$, i.e., the processing time of the last machine to terminate. Both RLS and the (1 + 1) EA have 4/3 worst case expected approximation ratios for the problem (i.e. there exist instances where they can get stuck on solutions by a factor of 4/3 worse than the optimal one) [16,17]. However if an appropriate restart strategy is setup in advance, both algorithms may be made into polynomial randomised approximation schemes (PRAS, i.e., algorithms that compute a $(1+\epsilon)$ approximation in polynomial time in the problem size with probability at least 3/4) with a runtime bounded by $O(n\ln(1/\epsilon)) \cdot 2^{(e\log e+e)\lceil 2/\epsilon\rceil \ln(4/\epsilon)+O(1/\epsilon)}$ [16,17]. Hence, as long as ϵ does not depend on the problem size, the algorithms can achieve arbitrarily good approximations with an appropriate restart strategy.

In this paper we will show that AIS can achieve stronger results. Firstly, we will show that both ageing and hypermutations can efficiently solve to optimality the worst-case instances for RLS and the (1 + 1) EA. More importantly, we will prove that ageing automatically achieves the necessary restart strategy to guarantee the $(1+\epsilon)$ approximation while hypermutations guarantee it in a single run in polynomial expected runtime and with overwhelming probability.

3 Generalised Worst-Case Instance

The instance from the literature P^*_ϵ leading to a 4/3 worst-case expected approximation ratio for RLS and the (1 + 1) EA consists of two large jobs with long processing times $p_1 := p_2 := 1/3 - \epsilon/4$ and the remaining $n-2$ jobs with short processing times $p_i := (1/3+\epsilon/2)/(n-2)$ for $3 \leq i \leq n$ (the sum of the processing times are normalised to 1 for cosmetic reasons) [16]. Any partition where one large job and half of the small jobs are placed on each machine is naturally a global optimum of the instance (i.e., the makespan is 1/2). Note that the sum of all the processing times of the small jobs is slightly larger than the processing time of a large job. The local optimum leading to the 4/3 expected approximation consists of the two large jobs on one machine and all the small jobs on the other. The makespan of the local optimum is $p_1 + p_2 = 2/3 - \epsilon/2$ and, in order to decrease it, a large job has to be moved to the fuller machine and at the same time at least $\Omega(n)$ small jobs have to be moved to the emptier machine. Since this requires to flip $\Omega(n)$ bits, which cannot happen with RLS and only happens with exponentially small probability $n^{-\Omega(n)}$ with the SBM of EAs, their expected runtime to escape from such a configuration is at least exponential.

In this section, to highlight the power of the AIS to overcome hard local optima for EAs, we generalise the instance P^*_ϵ to contain an arbitrary number $s = \Theta(1)$ of large jobs and show how the considered AIS efficiently solve the instance class to optimality by escaping from local optima efficiently while RLS and the (1 + 1) EA cannot. Hence, hypermutations and ageing are efficient on a vast number of instances where SBM and local mutations alone fail. The generalised instance class G^*_ϵ is defined as follows.

Definition 1. *The class of* PARTITION *instances G^*_ϵ are characterised by an even number of jobs n, an even number of large jobs $s = \Theta(1)$ and the following processing times:* $p_i = \begin{cases} \frac{1}{2s-1} - \frac{\epsilon}{2s} & \text{if } i \leq s \\ \frac{s-1}{n-s} \cdot (\frac{1}{2s-1} + \frac{\epsilon}{2(s-1)}) & \text{otherwise} \end{cases}$ *where $0 < \epsilon < 1/(2s-1)$ is an arbitrarily small constant.*

The instance class has the same property as the instance P^*_ϵ. If all the large jobs are placed on one machine and all the small jobs on the other, then at least $\Omega(n)$ small jobs need to be moved in exchange for a large job to decrease the discrepancy (the difference between the loads of the machines). Such a local optimum allows to derive a lower bound on the worst-case expected approximation ratio of algorithms that get stuck there. Obviously the greater the number of large jobs, the smaller the bound on the approximation ratio will be. We now prove that the $(1+1)$ EA has exponential expected runtime on G^*_ϵ. The proof is similar to that of the 4/3 approximation [16]. It essentially shows that with constant probability the algorithm gets trapped on the local optimum. Then the statement will follow by showing that exponential expected time is required to escape from there. That RLS also fails is essentially a corollary.

Theorem 1. *The $(1+1)$ EA needs at least $n^{\Omega(n)}$ fitness function evaluations in expectation to optimise any instance of G^*_ϵ.*

In the following subsection we will show that hypermutations are efficient. Afterwards we will do the same for ageing.

3.1 Hypermutations

We will start this subsection by proving some general statements about hypermutations. The hypermutation operator flips all the bits in a bitstring successively and evaluates the fitness after each step. Unless an improvement is found first, each hypermutation operation results in a sequence of n solutions sampled in the search space. This sequence of solutions contains exactly one bitstring for each Hamming distance $d \in [n]$ to the initial bitstring. Moreover, the string which will be sampled at distance d is uniformly distributed among all the bitstrings of distance d to the input solution. Thus, as we approach the middle, the number of possible outcomes grows exponentially large but the first and the last few strings sampled are picked among a polynomially large subset of the search space. We will now provide two lemmata regarding the probability of particular outcomes in the first and the last m bitstrings of the sequence.

Lemma 1. *Given that no improvement is found, the probability that k specific bit positions are flipped by the hypermutation operator in the first (or last) $m \geq k$ mutation steps is at least $\left(\frac{m-k+1}{n-k+1}\right)^k$.*

Lemma 2. *Let x^i be the ith bitstring sampled by the hypermutation and s^i the substring of x^i which consists of bit positions $S \subset [n]$. For any given target*

substring s^* *and integer* $m > |S|$*, the probability that* $\forall i \in \{m, m+1, \ldots, n-m-1, n-m\}$*,* $s^i = s^*$ *is at least* $\left(\frac{m-|S|+1}{n-|S|+1}\right)^{|S|}$.

We can observe that the probability bounds we get from these lemmata are polynomial if k (or $|S|$) is a constant. Moreover, if both $k = \Theta(1)$ and $m = \Omega(n)$ we obtain probabilities in the order of $\Omega(1)$.

For G^*_ϵ, we would like to distribute two kinds of jobs evenly among machines. While one type of job is constant in number, the other is in the order of $\Theta(n)$. So the previous lemmata would provide reasonable probability bounds only for the large jobs. For the small jobs we will make use of the fact that the configuration we want is not any configuration, but an exact half and half split. Intuitively, it is obvious that if all the jobs start on one machine, at exactly the $n/2$th bit flip, the split will be half and half. However, as the starting distribution gets further away from the extremes, it becomes less clear when the split will exactly happen. Fortunately, the fact that the number of small jobs is large, will work in our favor to predict the time of the split more precisely. Whilst the previous lemmata provide bounds for the first and last bitflips of hypermutation, we will use Serfling's concentration bound [27] on hypergeometric distributions to prove the following theorem about the bitflips in the middle.

Theorem 2. *If the input bitstring of hypermutation has* $\left(\frac{1}{2}+a\right)n$ *1-bits for some constant* $a > 0$*, then the probability that any solution sampled after the* $n\frac{a}{2a-c}$*th mutation step for any* $a > c > 0$ *to have more than n/2 1-bits is in the order of* $e^{-\Omega(nc^2)}$.

Corollary 1. *If the input bitstring of hypermutation has* $\left(\frac{1}{2}+a\right)n$ *1-bits for some constant* $a > 0$*, then with probability* $1 - e^{-\Omega(nc^2)}$*, there exists a* $k \in \{n\frac{a-c}{2a-c}, \ldots, n\frac{a}{2a-c}\}$ *for any positive* $a > c = \omega(1/\sqrt{n})$*, such that the number of 1-bits in the kth solution sampled by hypermutation has exactly n/2 1-bits.*

Now we have all the necessary tools to prove that the $(1+1)$ IA^{hyp} can solve G^*_ϵ efficiently. The heart of the proof of the following theorem is to show that from any local optimum, hypermutations identify the global optimum with constant probability unless an improvement is found first. The proof then follows because there are at most $O(n)$ different fitness levels.

Theorem 3. *The* $(1+1)$ IA^{hyp} *optimises the* G^*_ϵ *class of instances in* $O(n^2)$ *expected function evaluations.*

Since the worst case instance for the $(1+1)$ EA [16] is an instance of G^*_ϵ with $s = 2$, the following corollary holds.

Corollary 2. *The* $(1+1)$ IA^{hyp} *optimises* P^*_ϵ *in* $O(n^2)$ *expected function evaluations.*

3.2 Ageing

In this section we will show that the $(\mu + 1)$ EA^{ageing} can optimise the G^*_ϵ instances efficiently. Our approach to prove the following theorem is to first show that if a solution where the large jobs are equally distributed is sampled in the initialisation, then it takes over the population and the $(\mu + 1)$ EA^{ageing} quickly finds the optimum. The contribution of ageing is considered afterwards to handle the case when no such initial solution is sampled. Then, we will show that whenever the algorithm gets stuck at a local optima, it will reinitialise the whole population after τ iterations and sample a new population.

Theorem 4. *The* $(\mu + 1)$ *EA*^{ageing} *optimises the* G^*_ϵ *class of instances in* $O(\mu n^2 + \tau)$ *steps in expectation for* $\tau = \Omega(n^2)$ *and* $\mu = O(\log n)$.

Clearly the following corollary holds as P^*_ϵ is an instance of G^*_ϵ.

Corollary 3. *The* $(\mu+1)$ *EA*^{ageing} *optimises* P^*_ϵ *in* $O(\mu n^2+\tau)$ *steps in expectation for* $\tau = \Omega(n^2)$ *and* $\mu = O(\log n)$.

4 $(1 + \epsilon)$ Approximation Ratios

4.1 Hypermutations

In the next theorem we will show that the $(1 + 1)$ IA^{hyp} can efficiently find arbitrarily good constant approximations to any PARTITION instance. Before we state our main theorem, we will require the following helper lemma.

Lemma 3. *Let* x^i *be the ith bitstring sampled by the hypermutation,* s^i *the substring of* x^i *which consists of bit positions* $S \subset [n]$*, and,* $f(x) := \sum_{j \in S} x_j w_j$ *a linear function defined on the substring for some non-negative weights* w_j*. Given that the input bitstring of the hypermutation is* 0^n*, the expected value of* $f(x^i)$ *is* $\frac{i}{n} \sum_{j \in S} w_j$.

In the proof of the following theorem we first divide the search space into $2^{2/\epsilon}$ subspaces according to the distribution of the largest $2/\epsilon$ jobs and then further divide each subspace into the same n fitness levels used for the proof of the $(1 + 1)$ EA in [16]. However, we show that in each fitness level, hypermutations have a good probability of either finding a $(1 + \epsilon)$ approximation or leaving the fitness level for good. The statement then follows by summing over the $n2^{2/\epsilon}$ fitness levels.

Theorem 5. *The* $(1+1)$ *IA*^{hyp} *finds a* $(1+\epsilon)$ *approximation to any instance of* PARTITION *in at most* $n(\epsilon^{-(2/\epsilon)-1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon} + n^3 2^{2/\epsilon} + n^3$ *fitness function evaluations in expectation for any* $\epsilon = \omega(n^{-1/2})$.

Proof. In order to prove our upper bound on the runtime, we will divide the run of the algorithm into at most $2^{2/\epsilon}$ phases and find a bound on the expected time that is valid for all phases.

Following the proof of Theorem 3 in [16], we refer to the $s := \lceil 2/\epsilon \rceil - 1 \leq n/2$ jobs with the largest processing times as *large jobs* and to the rest as the *small jobs*. Let P be the sum of the processing times of all jobs. Since $p_1 \geq p_2 \geq \ldots \geq p_n$ w.l.o.g., we know that $p_i \leq \epsilon P/2$ for all $i > s$ because otherwise the sum of the first $s+1$ jobs would be larger than P.

Consider the 2^s partitions of only the large objects. We sort these 2^s partitions according to non-increasing makespan and denote the ith partition in the sequence as y_i. Now, we can divide the solution space of the original problem into subspaces A_i for $i \in [2^s]$, where A_i consists of all the solutions where the large jobs are distributed as in y_i. Let x_i^* denote the solution with the best fitness value in A_i.

Let $d(x)$ denotes the discrepancy of solution x. We define k as the smallest index i that satisfies $d(y_i) \leq \sum_{j=s+1}^{n} p_j$. Thus, for any large jobs configuration y_j for $j < k$, if all the small jobs are assigned to the emptier machine of y_j, then the load of the obtained solution is the same as the load of y_j itself because the discrepancy is larger than the sum of the processing times of the small jobs by definition. Since the makespan of y_j is a lower bound on the makespan of any solution $x \in A_j$, the solution where all small jobs are assigned to the emptier machine of y_i is the best solution in A_i for all $i < k$. Again for $i < k$, we can say that once the $(1+1)$ IA^{hyp} finds a solution with better fitness value than x_i^*, any solution that belongs to any set A_j for $j \leq i$ will be rejected by the algorithm due to its inferior fitness value. This allows us to divide the time until a solution better than x_{k-1}^* is found for the first time into $k-1$ distinct phases where during phase i the fitness of the current solution is between $f(x_{i-1}^*)$ and $f(x_i^*)$ (we define $f(x_0^*) := P$ for completeness). We will now further divide the expected length of phase i, into the expected time until x_i^* is found given an arbitrary solution x that satisfies $f(x_{i-1}^*) > f(x) > f(x_i^*)$, and the expected time until an improvement is found given that the current solution is $f(x_i^*)$.

We start with the expected time until x_i^* is found given an initial solution x such that $f(x_{i-1}^*) > f(x) > f(x_i^*)$ holds. Since $f(x_{i-1}^*) > f(x)$, the makespan of the underlying large job configuration of x is at least as good as the makespan of y_i and thus the makespan of x can be upper bounded by the load obtained when all small jobs are assigned to the fuller machine of y_i. Since $i < k$ we also know that the fuller machine of x_i^* has no small jobs on it. Thus, during phase i we can bound the fitness in the interval $[f(y_i), f(y_i) + \sum_{j=s+1}^{n} p_j]$. We further divide this interval into the following levels, $L_\ell := \left\{ x \mid f(y_i) + \sum_{j=s+\ell}^{n} p_j \geq f(x) > f(y_i) + \sum_{j=s+\ell+1}^{n} p_j \right\}$, implying that for any solution at level L_ℓ there is at least one job with processing time at least $p_{s+\ell}$ which can be moved to the emptier machine with probability $1/n$ as the first bit-flip and yield a solution at level $L_{\ell+1}$. Since there are at most n levels for the phase i, the expected number of iterations until the level L_{n-s+1}, which contains the solution x_i^* is discovered is at most n^2.

Secondly, we will bound the expected time until an improvement is found given that the current solution is $f(x_i^*)$. In order to do this we will bound the probability that a $(1+\epsilon)$ approximation is obtained in a single iteration given that no other improvements are found in the previous mutation steps of the hypermutation operator by some value $p_{\approx}$. This will allow us to claim that in expected $p_{\approx}^{-1}$ generations we will either find the approximation we sought or at least leave the subspace A_i for good.

Consider the optimal configuration of large jobs y_{2^s} and denote its makespan as L. Since both y_{2^s} and its complementary bitstring have the same makespan, w.l.o.g., we will assume the fuller machines of y_i and y_{2^s} are both M_1. According to Lemma 2, the $s \leq 2/\epsilon$ large jobs are assigned identically to y_{2^s} between the $n(\epsilon-\epsilon^2)$th and $n-n(\epsilon-\epsilon^2)$th bit-flips with probability at least $\left(\epsilon-\epsilon^2\right)^{2/\epsilon} e^{-1}$ Since the initial solution x_i^* does not assign any small jobs to M_1 and the sum of the processing times of the small jobs is less than $P/2$, by the $n(\epsilon-\epsilon^2)$th bit-flip the expected total processing time of small jobs moved from M_2 to M_1 is at most $(\epsilon-\epsilon^2)P/2$ according to Lemma 3. Due to Markov's inequality, with probability at least $1-((\epsilon-\epsilon^2)P/(\epsilon)P) = \epsilon$ the moved sum is less than $\epsilon P/2$ and the makespan of the solution is at most $L+\epsilon P/2$. In general, $OPT \geq L$ with OPT indicating the optimal makespan, since introducing the small jobs cannot improve the makespan and also $OPT \geq P/2$ since a perfect split is the best possible partition. Thus $(L+\epsilon P/2)/OPT$ is less than $(1+\epsilon)$. This implies that with probability

$$p_{\approx} \geq \frac{(\epsilon-\epsilon^2)^{2/\epsilon}}{e}\epsilon = \frac{\epsilon^{(2/\epsilon)+1}}{e(1-\epsilon)^{-2/\epsilon}} = \frac{\epsilon^{(2/\epsilon)+1}(1-\epsilon)^2}{e(1-\epsilon)^{-((1/\epsilon)-1)2}} \geq \frac{\epsilon^{(2/\epsilon)+1}(1-\epsilon)^2}{e^3}$$

a $(1+\epsilon)$ approximation is found unless an improvement is obtained before. The total expected waiting time to observe either an approximation or an improvement in all local optima y_i for $i<k$ is at most $(1/\epsilon^{(2/\epsilon)+1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon}$, since $k \leq 2^s \leq 2^{2/\epsilon}$. If we add the time spent in between local optima, $n^2 2^{2/\epsilon}$, we obtain the bound on the expected number of iterations until subspace A_k is reached for the first time.

Once subspace A_k is reached, by definition the underlying large job configuration has a discrepancy which is not large enough to fit all small jobs. This means that when the next local optimum is found, the discrepancy is less than half of the processing time of a small job. Thus, the locally optimal solution is a $(1+\epsilon)$ approximation since the processing time of small jobs is at most $\epsilon P/2$. Such a solution is found in n^2 expected time after the first solution in A_k is sampled. Summing up all the expected times bounded above, the expected runtime we obtain is: $(\epsilon^{-(2/\epsilon)-1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon} + n^2 2^{2/\epsilon} + n^2$. □

4.2 Ageing

The following theorem shows that the $(1+1)$ EA^{ageing} can find $(1+\epsilon)$ approximations. The proof follows the same ideas used to prove that RLS and the $(1+1)$ EA achieve a $(1+\epsilon)$ approximation if an appropriate restart strategy

is put in place [16,17]. In particular, the main proof idea is to use the success probability of the simple $(1+1)$ EA and show that ageing automatically causes restarts whenever the $(1+1)$ EA fails to find the approximation. Hence, a restart strategy is not necessary for the $(1+1)$ EA^{ageing} to achieve the desired approximation.

Theorem 6. *Let* $\tau = \Omega(n^{1+c})$ *where* $c = \Omega(1)$. *The* $(1+1)$ EA^{ageing} *finds a* $(1+\epsilon)$ *approximation to any instance of* Partition *in at most* $(en^2+\tau)2^{(e\log e+e)\lceil 2/\epsilon\rceil \ln(4/\epsilon)+\lceil 4/\epsilon\rceil-1}$ *fitness function evaluations in expectation for any* $\epsilon \geq 4/n$.

5 Conclusion

To the best of our knowledge this is the first time that polynomial expected runtime guarantees of solution quality have been provided concerning AIS for a classical combinatorial optimisation problem. We presented a class of instances of Partition to illustrate how hypermutations and ageing can efficiently escape from local optima where the standard bit mutations used by EAs get stuck for exponential time. Then we showed how this capability allows the AIS to achieve arbitrarily good $(1+\epsilon)$ approximations to any instance of Partition in polynomial time for any constant ϵ. In contrast to standard EAs and RLS, that require parallel runs or restart schemes to achieve such approximations, the AIS find them in a single run. The result is achieved in different ways. The ageing operator locates more promising basins of attraction by restarting the optimisation process after implicitly detecting it has found a local optimum. Hypermutations find improved approximate solutions efficiently by performing large jumps in the search space. Naturally, the proof would also apply to the complete Opt-IA [6,9] if the ageing parameter is set large enough, i.e., $\tau = \omega(n\epsilon^{-2/\epsilon})$.

References

1. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of 1994 IEEE Symposium on Security and Privacy, pp. 202–212 (1994)
2. Hedberg, S.: Combating computer viruses: IBM's new computer immune system. IEEE Par. Dist. Tech.: Syst. Appl. **4**(2), 9–11 (1996)
3. Dasgupta, D., Majumdar, N.S.: Anomaly detection in multidimensional data using negative selection algorithm. In: Proceedings of CEC 2002, pp. 1039–1044 (2002)
4. de Castro, L.N., Von Zuben, F.J.: Learning and optimization using the clonal selection principle. IEEE Trans. Evol. Comp. **6**(3), 239–251 (2002)
5. Kelsey, J., Timmis, J.: Immune inspired somatic contiguous hypermutation for function optimisation. In: Cantú-Paz, E. (ed.) GECCO 2003. LNCS, vol. 2723, pp. 207–218. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_26
6. Cutello, V., Nicosia, G., Pavone, M., Timmis, J.: An immune algorithm for protein structure prediction on lattice models. IEEE Trans. Evol. Comp. **11**(1), 101–117 (2007)

7. Jansen, T., Oliveto, P.S., Zarges, C.: On the analysis of the immune-inspired B-cell algorithm for the vertex cover problem. In: Liò, P., Nicosia, G., Stibor, T. (eds.) ICARIS 2011. LNCS, vol. 6825, pp. 117–131. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22371-6_13
8. Jansen, T., Zarges, C.: Computing longest common subsequences with the B-cell algorithm. In: Coello Coello, C.A., Greensmith, J., Krasnogor, N., Liò, P., Nicosia, G., Pavone, M. (eds.) ICARIS 2012. LNCS, vol. 7597, pp. 111–124. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33757-4_9
9. Corus, D., Oliveto, P.S., Yazdani, D.: On the runtime analysis of the Opt-IA artificial immune system. In: Proceedings of GECCO 2017, pp. 83–90 (2017)
10. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proceedings of GECCO 2017, pp. 777–784 (2017)
11. Oliveto, P.S., Lehre, P.K., Neumann, F.: Theoretical analysis of rank-based mutation-combining exploration and exploitation. In: Proceedings of CEC 2009, pp. 1455–1462 (2009)
12. Corus, D., Oliveto, P.S.: Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Trans. Evol. Comp. (2017)
13. Dang, D.-C., et al.: Emergence of diversity and its benefits for crossover in genetic algorithms. IEEE Trans. Evol. Comp. (2017, to appear)
14. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theor. Comp. Sci. **567**, 87–104 (2015)
15. Corus, D., He, J., Jansen, T., Oliveto, P.S., Sudholt, D., Zarges, C.: On easiest functions for mutation operators in bio-inspired optimisation. Algorithmica **78**(2), 714–740 (2016)
16. 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). https://doi.org/10.1007/978-3-540-31856-9_4
17. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16544-3
18. Neumann, F., Witt, C.: On the runtime of randomized local search and simple evolutionary algorithms for dynamic makespan scheduling. In: Proceedings of the 24th International Conference on Artificial Intelligence, pp. 3742–3748. AAAI Press (2015)
19. Corus, D., Oliveto, P.S., Yazdani, D.: Artificial immune systems can find arbitrarily good approximations for the NP-Hard partition problem. arXiv e-prints (2018). http://arxiv.org/abs/1806.00300
20. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1 + 1) evolutionary algorithm. Theor. Comp. Sci. **276**(1–2), 51–81 (2002)
21. Oliveto, P.S., Yao, X.: Runtime analysis of evolutionary algorithms for discrete optimisation. In: Auger, A., Doerr, B. (eds.) Theory of Randomized Search Heuristics: Foundations and Recent Developments, chap. 2, pp. 21–52. World Scientific (2011)
22. Oliveto, P.S., Sudholt, D.: On the runtime analysis of stochastic ageing mechanisms. In: Proceedings of GECCO 2014, pp. 113–120 (2014)
23. Jansen, T., Zarges, C.: On the role of age diversity for effective aging operators. Evol. Intell. **4**(2), 99–125 (2011)
24. Lehre, P.K., Oliveto, P.S.: Theoretical analysis of stochastic search algorithms. In: Marti, R., Pardalos, P., Resende, M. (eds.) Handbook of Heuristics, pp. 1–36. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-07153-4_35-1

25. Graham, R.: Bounds on multiprocessing timing anomalies. SIAM J. App. Maths **17**, 263–269 (1969)
26. Hochbaum, D.: Appromixation Algorithms for NP-Hard Problems. PWS Publishing Company, Boston (1997)
27. Serfling, R.J.: Probability inequalities for the sum in sampling without replacement. Ann. Stat. 39–48 (1974)

A Simple Proof for the Usefulness of Crossover in Black-Box Optimization

Eduardo Carvalho Pinto[1] and Carola Doerr[2(✉)]

[1] DreamQuark, Paris, France
[2] Sorbonne Université, CNRS, LIP6, Paris, France
Carola.Doerr@lip6.fr

Abstract. The idea to recombine two or more search points into a new solution is one of the main design principles of evolutionary computation (EC). Its usefulness in the combinatorial optimization context, however, is subject to a highly controversial discussion between EC practitioners and the broader Computer Science research community. While the former, naturally, report significant speedups procured by crossover, the belief that sexual reproduction cannot advance the search for high-quality solutions seems common, for example, amongst theoretical computer scientists. Examples that help understand the role of crossover in combinatorial optimization are needed to promote an intensified discussion on this subject.

We contribute with this work an example of a crossover-based genetic algorithm (GA) that provably outperforms any mutation-based black-box heuristic on a classic benchmark problem. The appeal of our examples lies in its simplicity: the proof of the result uses standard mathematical techniques and can be taught in a basic algorithms lecture.

Our theoretical result is complemented by an empirical evaluation, which demonstrates that the superiority of the GA holds already for quite small problem instances.

Keywords: Evolutionary computation · Crossover · Recombination
Runtime analysis

1 Introduction

Evolutionary Computation (EC) borrows inspiration from phenomena observed in biological evolution processes. One of the fundamental design principles of EC is *crossover;* i.e., the recombination of two or more candidate solutions into one or several *offspring.* EC practitioners frequently report that crossover (which is also referred to as *sexual reproduction*) brings significant performance gains. This belief, however, is often challenged in the broader Computer Science (CS) community, and in particular in the subarea of Theoretical CS, yielding to very generally formulated statements that crossover cannot be beneficial in combinatorial optimization. As an example we mention a quote by Christos Papadimitriou and

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 29–41, 2018.
https://doi.org/10.1007/978-3-319-99259-4_3

colleagues, who formulated the claim that "Simulated annealing tends to work quite well, but genetic algorithms do not".[1]

In light of this discrepancy, it has been one of the main focus question in the theory of EC community to contribute to a better understanding of when and why crossover-based algorithms can perform better than purely mutation-based ones. It seems quite notable that only very few examples exist where such an effect can be rigorously proven.

1.1 Selected Theoretical Results on the Benefits of Crossover

We summarize a few selected results that prove an advantage of crossover in the discrete black-box optimization context, and refer the interested reader to [16] for an extended discussion. The first work observing a benefit of crossover-based GAs over a standard evolutionary algorithm (EA) dates back to [11], where so-called Jump functions are considered, in which algorithms are required to "jump" a gap between local and global optima. As discussed in [3,16], several follow-up works introduced similarly artificial problems to demonstrate an advantage of crossover. The first classical combinatorial example for which recombination could be shown to be beneficial was presented in [8]. In this work, a problem-tailored crossover operation was shown to be advantageous for the all-pairs-shortest-path problem.

These theoretical results, albeit being very appealing, do not answer the question of how beneficial the use of crossover is in standard EAs, or for standard benchmark problems. Starting with the work [14], the quest to prove advantages of crossover for simple hill-climbing tasks has recently taken considerable momentum. Sudholt proved that a greedy $(\mu + 1)$ GA with a diversity mechanism that avoids duplicates needs $(1 + o(1))\frac{e}{2} n \ln n \approx 1.359 n \ln n + o(n \ln n)$ function evaluations, on average, to optimize OneMax; the combinatorial optimization problem asking to minimize the Hamming distance to an unknown bit string $z \in \{0,1\}^n$. This runtime is better by a factor of two than the expected $(1 + o(1))en \ln n$ optimization time of any evolutionary algorithm using only standard bit mutation [15,17]. Sudholt also proved that the expected runtime of the algorithm can be further reduced to approximately $1.19 n \ln n + o(n \ln n)$ by increasing the mutation rate from $1/n$ to $(1 + \sqrt{5})/(2n)$.

The results of [14] were generalized to less greedy $(\mu+1)$ GAs in [16] and to GAs avoiding the diversity mechanism in [3]. All these works show an advantage of crossover-based $(\mu + 1)$ GAs over evolutionary algorithms using standard bit mutation. They do not, however, beat the average performance of another very common randomized optimization heuristic, Randomized Local Search (RLS). The expected optimization time of RLS on OneMax is $n \ln(n/2) + \gamma\, n \pm o(1)$, with $\gamma \approx 0.5772156649$ being the EulerMascheroni constant [4]. For a more convincing argument in favor of crossover, one would like to have an example for a crossover-based heuristic that outperforms not only mutation-based EAs but also RLS as well as any other so-called *unary unbiased black-box algorithm.*

[1] See, for example, here: https://www.simonsfoundation.org/2010/05/18/why-sex/.

The notion of a unary unbiased black-box algorithm was introduced in [13] as a model for purely mutation-based algorithms. While it was already proven in [13] that any unary unbiased black-box algorithm has an expected optimization time on OneMax of order at least $n \log n$, a precise lower bound, which is $n \ln(n) - cn \pm o(n)$ for a constant c between 0.2539 and 0.2665, could be shown only recently [7]. The expected optimization times proven in [3,14,16] are all by a multiplicative factor of at least 1.19 larger than this bound.

In [9] it was shown that *binary* unbiased black-box algorithms exist that achieve a linear expected optimization time on OneMax. While this can be seen as a proof in favor of recombination, the algorithm is highly problem-tailored. A more appealing example rigorously proving an advantage of crossover over any unary unbiased black-box algorithm has been presented in [6]. The $(1 + (\lambda, \lambda))$ GA uses only well-known and widely applied building blocks from the EC literature (standard bit mutation, (biased) uniform crossover, and elitist selection), but recombines them in a new way: by first mutating a best so-far solution through standard bit mutation, the crossover operator becomes a "repair mechanism". For suitable parameter settings, the $(1 + (\lambda, \lambda))$ GA can achieve linear expected optimization time on OneMax [5,6], and, by the lower bounds of [7,13], therefore scales much more favorably with the problem dimension than any unary unbiased black-box algorithm.

1.2 Our Results

In this work, we revisit the analysis of the greedy $(\mu + 1)$ GA with diversity mechanism presented in [14]. Following the suggestion made in [1] we take a more implementation-aware perspective on this algorithm, in that we do not charge function evaluations for search points that are identical to one of their direct ancestors. Put differently, we try to avoid creating such offspring, as they do not provide any new information about the problem instance at hand. In the absence of noise, this is how one would implement the $(\mu+\lambda)$ GA for all practical purposes, cf. [1] for a discussion. We note that for the two variation operators employed by the $(\mu + \lambda)$ GA, standard bit mutation and uniform crossover, tracking whether or not an offspring equals one of its parents is very simple and comes at almost no cost.

Quite surprisingly, we show that this simple modification yields performance bounds that are strictly better than the above-mentioned $n \ln(n) - cn \pm o(n)$ lower bound valid for all unary unbiased black-box algorithms. More precisely, we show that, for a suitably chosen mutation rate p, the modified greedy $(\mu + 1)$ GA with diversity mechanism achieves an $0.851..n \ln(n) + o(n \log n)$ expected optimization time on OneMax. The proof of this result is surprisingly simple, and can be taught in an undergraduate course.

2 The Greedy $(\mu + 1)$ GA

We present the crossover-based genetic algorithm (GA) for which we will prove in Sect. 3 that it outperforms any mutation-based algorithm on the Hamming

distance problem OneMax. The algorithm is a (mild) modification of an algorithm previously suggested for the study of the effectiveness of crossover: the greedy $(\mu + 1)$ GA presented in [14]. We present the original algorithm in Sect. 2.1, motivate our modifications in Sect. 2.2, and describe the modified greedy $(\mu + 1)$ GA in Sect. 2.3.

2.1 The Original Greedy $(\mu + 1)$ GA

The greedy $(\mu + 1)$ GA proposed by Sudholt in [14] is Algorithm 1 with lines 5 to 8 replaced by "Sample ℓ from $\mathrm{Bin}(n, p)$". It maintains a population $\mathcal{P}$ of μ individuals. $\mathcal{P}$ is initialized by sampling μ search points independently and uniformly at random. Each iteration consists of two steps; a crossover step and a mutation step. In the *crossover step* two parents x and y are selected uniformly at random (with replacement) from those individuals $u \in \mathcal{P}$ for which $f(u) = \max_{v \in \mathcal{P}} f(v)$ holds. From these two search points an offspring z' is created by *uniform crossover* $\mathrm{cross}(x, y)$, which samples a new search point by choosing, independently for every position $i \in [n]$ and uniformly at random, whether to copy the entry of the first or the second argument. In the *mutation phase* this offspring z' is modified by *standard bit mutation*, which flips each bit independently with some probability $p \in (0, 1)$. The so-created offspring z is evaluated. If $z \notin \mathcal{P}$ and its fitness is at least as good as $\min_{v \in \mathcal{P}} f(v)$, it replaces the worst individual in the population, ties broken uniformly at random. The requirement $z \notin \mathcal{P}$ is a so-called *diversity mechanism*.

From this description, we easily observe that from the whole population only those with a best-so-far fitness value are relevant, the others are never selected for reproduction, hence the attribute "greedy" in the name of this algorithm. When there is only one search point of best-so-far function value, the crossover simply creates a copy of this search point, and progress has to be made by mutation, while in the case that at least two different search points with best-so-far fitness exist, there is positive probability that crossover recombines these into a strictly better solution. Sudholt proved that this probability is large enough for the greedy $(\mu+1)$ GA to outperform its mutation-only analog, the $(1+1)$ EA. More precisely, it is shown in [14] that, for $\mu \geq 2$ and $n \geq 2$, the expected optimization time of the greedy $(\mu+1)$ GA on OneMax, i.e., the expected number of function evaluations that the algorithm performs until it evaluates for the first time an optimal solution, is at most

$$\frac{\ln(n^2 p + n) + 1 + p}{p(1-p)^{n-1}(1+np)} + \frac{8n}{(1-p)^n}. \tag{1}$$

As mentioned in the introduction, this bound was later generalized to a less greedy variant of the $(\mu + 1)$ GA in [16] and to one avoiding the diversity mechanism in [3]. These generalizations are not relevant to this present work.

The bound in (1) is by a multiplicative factor of about $1/(1 + np)$ smaller than the expected optimization time of the $(1 + 1)$ EA. For $p = 1/n$ this factor evaluates to $1/2$, showing that for this choice of p the greedy $(\mu+1)$ GA is about

a factor of two faster than the $(1 + 1)$ EA. This advantage can be boosted by choosing larger mutation rates. In fact, the expression in (1) is minimized for $p = (1 + \sqrt{5})/(2n)$. With this mutation rate, the expected optimization time of the greedy $(\mu + 1)$ GA on OneMax is at most $1.19n \ln n + 35n$. This is better than the expected optimization time of the $(1 + 1)$ EA, but worse than the $nH_{n/2} - 1/2 \approx n \ln(n) - 0.1159n + O(1)$ expected optimization time of RLS [4].

2.2 Standard Bit Mutation: Theory vs. Practice

When the offspring created in the crossover phase of the greedy $(\mu + 1)$ GA equals one of its parents, the only source for a successful iteration is the standard bit mutation operator applied in the mutation step. Standard bit mutation is probably *the* most frequently used variation operator in evolutionary approaches for the optimization of pseudo-Boolean problems $f : \{0,1\}^n \to \mathbb{R}$. We discuss in this section that most EA practitioners do not take the literal definition of standard bit mutation provided above too seriously, and implement a slightly different variation operator instead.

We start our discussion by observing that for every mutation rate $p \in (0,1)$ the probability that standard bit mutation merely creates a copy of the parent individual is strictly positive. More precisely, the number ℓ of bits that are flipped by the standard bit mutation operator follows the binomial distribution $\mathrm{Bin}(n,p)$. That is, for all $k \in [0..n] := \{0,1,\dots,n\}$ the probability to flip exactly k bits equals $\binom{n}{k}p^k(1-p)^{n-k}$. For $k = 0$ this expression evaluates to $(1-p)^n$. The evaluation of copies, however, does not provide any new information about the problem instance f, unless f is a dynamic function or its evaluation is noisy.

The question how to deal with these offspring disunites theoretical and empirical research in evolutionary computation. While almost all theoretical runtime results for evolutionary algorithms charge the algorithms for evaluating such copies, the practitioner would typically not call the function evaluation for such offspring. Two strategies are commonly used in practice. The first one, which is the most common one for +-selection strategies, avoids to generate copies in the first place, by sampling from a conditional distribution that assigns probability 0 to sampling the parent individual. An alternative strategy, that is more reasonable for comma-selection, does include sampled copies of the parent individual in the offspring population, but does not evaluate these as their function values are already known. When the performance measure is based on counting function evaluations, both aforementioned strategies coincide for the $(\mu + 1)$-type algorithms considered in this work.

We now describe how the creation of copies can be avoided. To this end, we first observe that a reasonable implementation of standard bit mutation would first sample the number ℓ of bits to flip, and then apply the mut_ℓ variation operator that flips ℓ pairwise different, uniformly selected bits. As discussed above, in the literal interpretation of standard bit mutation the number ℓ is distributed according to $\mathrm{Bin}(n,p)$. If we do not want to create copies, we only need to change the distribution that we sample from. The most common implementation of standard bit mutation uses a resampling strategy in which ℓ is sampled

Algorithm 1. The greedy $(\mu+1)$ GA_{mod} with mutation probability p for the maximization of a given function $f : \{0,1\}^n \to \mathbb{R}$.

```
Choose x^(1), ..., x^(μ) from {0,1}^n independently and u.a.r. and evaluate them;
for t = 1, 2, 3, ... do
    Choose x, y ∈ arg max_{w∈P} f(w) u.a.r. (with replacement);
    if x ≠ y then z' ← cross(x, y); else z' ← x;
    if z' ∉ {x, y} then
        Sample ℓ from Bin(n, p);
    else
        Sample ℓ from Bin_{>0}(n, p);
    Sample z ← mut_ℓ(z') and evaluate f(z);
    if (z ∉ P and f(z) ≥ min_{w∈P} f(w)) then
        Choose v ∈ arg min_{w∈P} f(w) u.a.r. and replace v by z;
```

from $\text{Bin}(n,p)$ until a strictly positive value is sampled for the first time. Thus, effectively, in this resampling approach, the *mutation strength* ℓ is sampled from the conditional binomial distribution $\text{Bin}_{>0}(n,p)$, which assigns to every $k \in [n]$ a probability of $\text{Bin}(n,p)(k)/\sum_{i=1}^{\infty}\text{Bin}(n,p)(i) = \binom{n}{k}p^k(1-p)^{n-k}/(1-(1-p)^n)$.

2.3 The Modified Greedy $(\mu+1)$ GA

We apply the resampling idea to the greedy $(\mu+1)$ GA. To motivate this, we briefly discuss the circumstances under which the solution created in the crossover phase is identical to one of its parents. Note that only in this case we need to enforce that at least one bit is flipped by the standard bit mutation, since in the other case, the crossover may have successfully created a new solution that is at least as good as its parents. When the population contains only one search point of current-best function value, this one is (deterministically) selected twice for the crossover step, so that the crossover operator cannot create diversity. However, even in the presence of $k > 1$ different search points of best-so far fitness, the probability to choose the same one twice equals $1/k$.[2] Furthermore, it can happen that the parents are not identical, but the offspring copies one of them. This event is also not unlikely: if we denote by d the Hamming distance of the two selected parents x and y, the probability that the offspring created by the uniform crossover cross equals either x or y is $1/2^{d-1}$. The situation $d = H(x,y) = 2$ occurs quite frequently, resulting in a $1/2$ probability that the crossover reproduces one of the two parents. In all these cases, the chances to make progress rely exclusively on the mutation phase.

[2] See Sect. 3 for a discussion of the fact that sampling the parent without replacement improves the expected optimization time of this algorithm on OneMax. We do not apply this modified parent selection rule in the greedy $(\mu+1)$ GA_{mod} to highlight that the main improvement stems from the modified mutation step.

Tracking whether or not the offspring created by crossover equals one of its parents is very simple, and can be done efficiently while creating it. As argued above, in such an event we would like to avoid that the mutation operator chooses mutation strength $\ell = 0$. In line with common implementations of standard bit mutation, we use the re-sampling strategy described in Sect. 2.2. With this re-sampling strategy, the greedy $(\mu + 1)$ GA becomes Algorithm 1, which we refer to as the greedy $(\mu + 1)$ GA_{mod}.

3 Theoretical Investigation

Following very closely the proof of Theorem 2 in [14], it is not difficult to obtain the following runtime statement, which is the main result of this paper.

Theorem 1. *For $n \geq 2$ and $\mu \geq 2$, the expected optimization time of the greedy $(\mu+1)$ GA_{mod} with mutation rate p on any* OneMax *function $\text{Om}_u : \{0,1\}^n \to [0..n], x \mapsto |\{i \in [n] \mid x_i = u_i\}|$ is at most*

$$\frac{(1-(1-p)^n)(\ln(n^2p+n)+1+p)}{p(1-p)^{n-1}(1+np)} + \frac{8n}{(1-p)^n}. \tag{2}$$

Before presenting the proof for Theorem 1, we first discuss its consequences, and why it shows that the greedy $(\mu + 1)$ GA_{mod} can hillclimb faster than any unary unbiased black-box algorithm.

For mutation rate $p = c/n$, the upper bound (2) evaluates to

$$\frac{1-(1-c/n)^n}{c(1-c/n)^{n-1}(1+c)} n\ln(n) + \Theta(n).$$

For large n, we can approximate the factor $B(c,n) := \frac{1-(1-c/n)^n}{c(1-c/n)^{n-1}(1+c)}$ in this expression by $A(c) := \frac{1-\exp(-c)}{c\exp(-c)(1+c)}$. Evaluating $B(1,n)$ and minimizing $A(c)$ with respect to c gives the following result.

Corollary 1. *For $\mu \geq 2$ the expected optimization time of the greedy $(\mu + 1)$ GA_{mod} with mutation rate $p = 1/n$ on* OneMax *is at most $(1 + o(1))\frac{e-1}{2} n\ln(n) \approx 0.859140914n\ln(n) + o(n\ln n)$ and for $p = 0.773581/n$ it is at most $(1 + o(1))0.850953n\ln(n)$.*

By the result of [7], these two bounds are about 14 to 15 % smaller than the expected optimization time of *any* unary unbiased black-box algorithm. As far as we know this is the first time that a "classic" GA is shown to outperform RLS on OneMax—the only other evolutionary algorithm that we are aware of is the $(1 + (\lambda, \lambda))$ GA with fitness-based [6] and self-adjusting [5] population size.

To study the convergence towards the mutation rate used in Corollary 1, we summarize in the following table how the value of c that minimizes $B(c,n)$ changes with the problem dimension n. We also provide a numerical evaluation of the factor $B(1,n)$, the multiplicative factor of the $n\ln n$ term for the greedy $(\mu + 1)$ GA_{mod} with mutation rate $p = 1/n$.

n	10	100	500	1 000	5 000
c	0.783953	0.774577	0.773778	0.773679	0.773599
$B(c,n)$	0.831839	0.859091	0.850581	0.850766	0.850915
$B(1,n)$	0.840587	0.857340	0.858782	0.858961	0.859105

Proof (of Theorem 1). Following [14], we say that the algorithm is on fitness level i if the best individual in the population has function value i. Like Sudholt, for each i, we distinguish two cases.

Case i.1: there is exactly one search point $x \in \mathcal{P}$ with $f(x) = i$ and for all $y \in \mathcal{P} \setminus \{x\}$ it holds that $f(y) < i$. In this situation, the offspring z is the outcome of standard bit mutation on x. The algorithm leaves this situation when (a) $f(z) > i$ or (b) $f(z) = f(x)$ and $z \neq x$. The probability for (a) to happen is at least $(n-i)p(1-p)^{n-1}/(1-(1-p)^n)$, since this is the probability that exactly one of the zero bits is flipped in the mutation phase. Likewise, the probability of event (b) is $i(n-i)p^2(1-p)^{n-2}/(1-(1-p)^n) \geq i(n-i)p^2(1-p)^{n-1}/(1-(1-p)^n)$. Once the algorithm has left case i.1 it never returns to it. This is ensured by the diversity mechanism, which allows to include z in the population only if it isn't there yet (line 10 of Algorithm 1). The total expected time spent in the cases $i.1$, $i = 0, \dots, n-1$ is therefore at most

$$\frac{1-(1-p)^n}{p(1-p)^{n-1}} \sum_{i=0}^{n-1} \frac{1}{(n-i)(1+ip)}.$$

The same algebraic computations as in [14] show that this expression can be bounded from above by

$$\frac{(1-(1-p)^n)(\ln(pn^2+n)+1+p)}{p(1-p)^{n-1}(1+np)}.$$

Case i.2: there are at least two different search points x and y with $f(x) = f(y) = i$ and, for all $w \in \mathcal{P}$, $f(w) \leq i$ holds. For this case we can use exactly the same arguments as Sudholt does for the original greedy $(\mu+1)$ GA: the probability to sample two different parents $x \neq y$ in the crossover step is at least $1/2$. Assuming that we are in this situation, it is not difficult to show that the probability that the intermediate offspring z' satisfies $f(z') > i$ is at least $1/4$, cf. [14] for an explicit proof. Conditioning on this event, we certainly have $z' \notin \{x, y\}$ so that the mutation strength ℓ is therefore sampled from the unconditional binomial distribution $\mathrm{Bin}(n,p)$. The probability to sample $\ell = 0$ equals $(1-p)^n$. Putting everything together, we see that, starting in case $i.2$, the total probability to leave fitness level i is at least $(1-p)^n/8$, so that the total expected time spent in the cases $i.2$, $i = 0, \dots, n-1$ is at most $8n/(1-p)^n$. □

The reader familiar with the notion of k-ary unbiased black-box algorithms may wonder if the greedy $(\mu+1)$ $\mathrm{GA}_{\mathrm{mod}}$ is unbiased, and of which arity it is.

We note without proof that it is unbiased, but that care has to be taken when computing the arity of this algorithm. Line 10 of Algorithm 1 seems to suggest that the arity of this algorithm is $\mu + 1$. Note however, that in particular for the case $\mu = 2$, only a mild modification of Algorithm 1 is needed to obtain a binary unbiased algorithm whose expected optimization time on OneMax also satisfies the bound stated in Theorem 1. This not being the main focus of the present work (rather are we interested in a simple example of a "classic" GA that can be proven to outperform any unary unbiased black-box optimization algorithm), we defer the details of this alternative to an extended journal version of this work.

Additional Performance Gains. It is beyond the scope of this work to analyze the tightness of the upper bounds proven in Theorem 1, and additional gains may be possible by choosing different values for p. We also remark that RLS_{opt} (described below), the RLS-variant from [7] achieving the (up to lower order term) optimal runtime among all unary unbiased black-box algorithms on OneMax, uses fitness-dependent mutation rates. It is possible (and likely) that the greedy $(\mu + 1)$ GA_{mod}, as well, could profit further from choosing its mutation rate in such an adaptive way. We have to leave this question for future work.

One may wonder why we have not abbreviated line 4 as "$z' \leftarrow \text{cross}(x, y)$", regardless of whether or not $x = y$. This would of course give the same algorithm. Our variant, however, makes it more explicit that it may happen that $x = y$ is sampled in line 3. As discussed above, when $k := |\arg\max_{w \in \mathcal{P}} f(w)| = 1$, this is always the case. But also for $k > 1$ this situation can occur, because the sampling in line 3 uses replacement. If we focus, for a moment on the situation $\mu = 2$, then one might argue that it is more "natural" to do a crossover of both parents in line 4, provided that they have the same function value. More generally, one would want to enforce $x \neq y$ whenever $k > 1$. This modification does not affect the cases $i.1$ in the proof of Theorem 1, but it does increase the success probability of the cases $i.2$ by a multiplicative factor of 2. With this observation, we easily see that the additive $\frac{8n}{(1-p)^n}$ term in the runtime bounds for the $(\mu + 1)$ GA and the greedy $(\mu + 1)$ GA_{mod} with mutation rate p can be replaced by $\frac{4n}{(1-p)^n}$.

4 Empirical Evaluation

Complementing the theoretical results above, we now investigate the performance of the greedy $(2 + 1)$ GA_{mod} on OneMax by empirical means, to shed light on its behavior for small dimensions. As we shall see, our experiments confirm a considerable advantage of the greedy $(2 + 1)$ GA_{mod} over RLS already for small problem dimensions. We use this section also to compare the greedy $(2 + 1)$ GA_{mod} with another crossover-based genetic algorithm, the self-adjusting $(1 + (\lambda, \lambda))$ GA suggested in [6]. For a fair comparison, we modify the $(1 + (\lambda, \lambda))$ GA in the same spirit in which we have modified the greedy $(\mu + 1)$ GA. Finally, we also provide a comparison with RLS_{opt}, the RLS variant that in each iteration chooses the

drift-maximizing mutation strength. We briefly describe these two algorithms before we present our empirical findings.

Modifying the $(1 + (\lambda, \lambda))$ GA. It was shown in [5] that the $(1 + (\lambda, \lambda))$ GA achieves a linear optimization time on OneMax when equipped with a self-adjusting choice of the offspring population size. No static parameter choice can achieve this performance [5] and experimental results presented in [6] suggest that already for $n \geq 1\,500$ the self-adjusting choice of the population size outperforms any static one.

For reasons of space, we cannot discuss the algorithm in great detail and refer the reader to [5] for a discussion of the self-adjusting $(1 + (\lambda, \lambda))$ GA. In line with our modifications of the greedy $(\mu+1)$ GA, we change the original $(1+(\lambda, \lambda))$ GA by choosing the mutation strength ℓ from the conditional $\mathrm{Bin}_{>0}(n, p)$ distribution (instead of sampling from $\mathrm{Bin}(n, p)$) and by not evaluating those offspring created in the crossover phase that are identical to one of their two direct parents.

As in the original self-adjusting $(1 + (\lambda, \lambda))$ GA we use a mutation rate of $p = \lambda/n$, a crossover bias $c = 1/\lambda$, and update strength $F = 3/2$. With this parametrization, the probability of the original $(1 + (\lambda, \lambda))$ GA to sample a mutation strength $\ell = 0$ equals $(1 - \lambda/n)^n \approx \exp(-\lambda)$. A choice of $\ell = 0$ results in an entirely useless iteration that costs 2λ function evaluations. Note further that particularly in the beginning (λ is close to one) but also in the last steps of the optimization process (λ approaches $\sqrt{n}$), the probability that an offspring created from $\mathrm{cross}_{1/\lambda}(x, y)$ equals x or y is fairly large. It is therefore not surprising that our modified $(1+(\lambda, \lambda))$ $\mathrm{GA}_{\mathrm{mod}}$ indeed corresponds to how practitioners have implemented the $(1 + (\lambda, \lambda))$ GA for an empirical evaluation [10].

None of our modifications can influence the *asymptotic order* of the optimization time, since the linear performance of the original $(1 + (\lambda, \lambda))$ GA is already asymptotically optimal [5]. What we do observe, however, is that our modifications have a non-trivial impact on the leading constant.

RLS with Fitness-Dependent Mutation Strengths. $\mathrm{RLS}_{\mathrm{opt}}$ is the $(1+1)$-type heuristic which in every iteration creates one offspring y from the parent x by flipping a number of bits that is chosen to maximize the expected progress towards the optimum. y replaces x if it is at least good; i.e., if $f(y) \geq f(x)$ holds.

It was proven in [7] that this *drift maximizer* is (almost) optimal among all unary unbiased black-box algorithms. More precisely, it is shown that the performance of any unary unbiased algorithm can be better by at most an additive $o(n)$ term.

To run $\mathrm{RLS}_{\mathrm{opt}}$ in our experiments, we have computed, for every tested dimension n and every fitness value $v \in [0..n-1]$ the value $\ell^*_{n,v}$ that maximizes the expected drift

$$\begin{aligned} B(n, v, \ell) &:= \mathbb{E}[\max\{\textsc{Om}(y) - \textsc{Om}(x), 0\} \mid \textsc{Om}(x) = v, y = \mathrm{mut}_\ell(x)] \\ &= \sum_{i=\lceil \ell/2 \rceil}^{\ell} \frac{\binom{n-v}{i}\binom{v}{\ell-i}(2i-\ell)}{\binom{n}{\ell}}, \end{aligned} \tag{3}$$

i.e., we do not work with the approximation proposed in [7] but the original drift maximizer.

Experimental Results. Figure 1 shows experimental data for the performance of the aforementioned algorithms on OneMax, for n ranging from 500 to 5 000. The $(1 + (\lambda, \lambda))$ GA and the $(1 + (\lambda, \lambda))$ GA_{mod} use self-adjusting λ values, and for the greedy $(2 + 1)$ GA_{mod} we use mutation rate $0.773581/n$ and the variant that recombines both parents if their function values are identical. In the reported ranges, the expected performance of the original greedy $(2+1)$ GA from [14] with mutation rate $p = (1 + \sqrt{5})/(2n)$ is very similar to that of the self-adjusting $(1+(\lambda, \lambda))$ GA (cf. Fig. 8 in [6]); we do not plot these data points to avoid an overloaded plot. Detailed statistical information for Fig. 1 can be found in [2]. We observe that both the $(1 + (\lambda, \lambda))$ GA_{mod} as well as the greedy $(2 + 1)$ GA_{mod} are better than RLS_{opt} already for quite small problem sizes. We also observe that, in line with the theoretical bounds, the advantage of the $(1 + (\lambda, \lambda))$ GA_{mod} over the greedy $(2 + 1)$ GA_{mod} and over RLS_{opt} increases with the problem size.

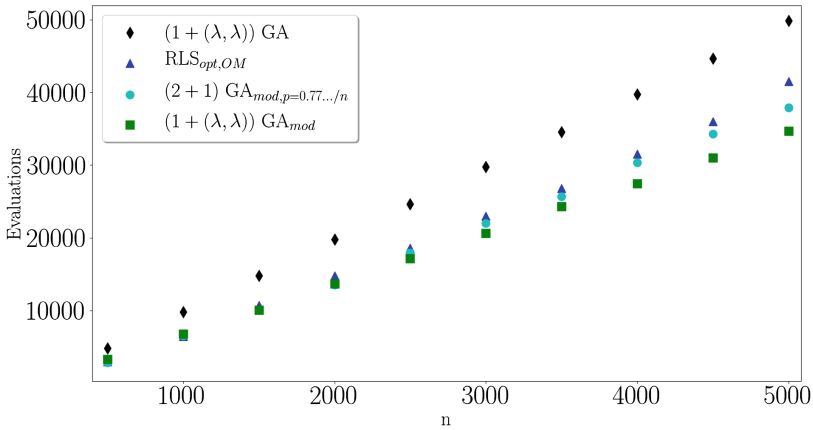


Fig. 1. Average runtimes for 100 independent runs of the respective algorithms on OneMax for different problem sizes n.

5 Conclusions

We have presented a simple example of a crossover-based heuristic that performs better than any unary unbiased black-box algorithm on the OneMax benchmark function. The mathematical proof is surprisingly easy, and raises the question why the result has been previously overlooked, despite the considerable attention that the *usefulness of crossover* question has received in the runtime analysis community.

The main idea behind our proof is a more careful performance evaluation. We therefore believe that the discussion how to measure the efficiency of an evolutionary algorithm, which had previously been suggested in [12], should be taken more seriously, in particular in light of the significant increase in the precision of state-of-the-art runtime results. We believe this question to be particularly relevant for the comparison of evolutionary algorithms with other standard optimization approaches like local search.

The proof of Theorem 1 does not invoke any involved mathematical machinery, and can be taught to undergraduate students. We hope that this makes it an appealing example for the discussion on the role of sexual reproduction in combinatorial optimization.

Acknowlegement. We thank the anonymous reviewers of this paper for their constructive feedback, which has helped us to improve the presentation of our main result. This research benefited from the support of the FMJH Program Gaspard Monge in optimization and operation research, and from the support to this program from EDF.

References

1. Pinto, E.C., Doerr, C.: Discussion of a more practice-aware runtime analysis for evolutionary algorithms. In: EA 2017, pp. 298–305 (2017)
2. http://www-desir.lip6.fr/~doerr/CarvalhoDoerr-PPSN18-Crossover.pdf
3. Corus, D., Oliveto, P.S.: Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Trans. Evol. Comput. (2018, to appear)
4. Doerr, B., Doerr, C.: The impact of random initialization on the runtime of randomized search heuristics. Algorithmica **75**, 529–553 (2016)
5. Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. Algorithmica **80**, 1658–1709 (2018)
6. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theor. Comput. Sci. **567**, 87–104 (2015)
7. Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. In: GECCO 2016, pp. 1123–1130. ACM (2016)
8. Doerr, B., Happ, E., Klein, C.: Crossover can provably be useful in evolutionary computation. Theor. Comput. Sci. **425**, 17–33 (2012)
9. Doerr, B., Johannsen, D., Kötzing, T., Lehre, P.K., Wagner, M., Winzen, C.: Faster black-box algorithms through higher arity operators. In: FOGA 2011, pp. 163–172. ACM (2011)
10. Goldman, B.W., Punch, W.F.: Fast and efficient black box optimization using the parameter-less population pyramid. Evol. Comput. **23**, 451–479 (2015). https://github.com/brianwgoldman?tab=repositories
11. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms - a proof that crossover really can help. Algorithmica **34**, 47–66 (2002)
12. Jansen, T., Zarges, C.: Analysis of evolutionary algorithms: from computational complexity analysis to algorithm engineering. In: FOGA 2011, pp. 1–14. ACM (2011)
13. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**, 623–642 (2012)

14. Sudholt, D.: Crossover speeds up building-block assembly. In: GECCO 2012, pp. 689–702. ACM (2012)
15. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Trans. Evol. Comput. **17**, 418–435 (2013)
16. Sudholt, D.: How crossover speeds up building block assembly in genetic algorithms. Evol. Comput. **25**, 237–274 (2017)
17. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Comb. Probab. Comput. **22**, 294–318 (2013)

Destructiveness of Lexicographic Parsimony Pressure and Alleviation by a Concatenation Crossover in Genetic Programming

Timo Kötzing[1], J. A. Gregor Lagodzinski[1], Johannes Lengler[2], and Anna Melnichenko[1(✉)]

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
[2] ETH Zürich, Zürich, Switzerland
anna.melnichenko@hpi.de

Abstract. For theoretical analyses there are two specifics distinguishing GP from many other areas of evolutionary computation. First, the variable size representations, in particular yielding a possible bloat (i.e. the growth of individuals with redundant parts). Second, the role and realization of crossover, which is particularly central in GP due to the tree-based representation. Whereas some theoretical work on GP has studied the effects of bloat, crossover had a surprisingly little share in this work.

We analyze a simple crossover operator in combination with local search, where a preference for small solutions minimizes bloat (*lexicographic parsimony pressure*); the resulting algorithm is denoted *Concatenation Crossover GP*. For this purpose three variants of the well-studied Majority test function with large plateaus are considered. We show that the Concatenation Crossover GP can efficiently optimize these test functions, while local search cannot be efficient for all three variants independent of employing bloat control.

1 Introduction

Genetic Programming (GP) is a field of Evolutionary Computing (EC) where the evolved objects encode programs. Usually a tree-based representation of a program is iteratively improved by applying variation operators (mutation and crossover) and selection of suitable offspring according to their quality (fitness). Most other areas of EC deal with fixed-length representations, whereas the tree-based representation distinguishes GP. This representation of variable size leads to one of the main problems when applying GP: *bloat*, which describes an unnecessary growth of representations. Solutions may have many redundant parts, which could be removed without afflicting the quality, and search is slowed down, wasted on uninteresting areas of the search space.

In this paper we study GP from the point of view of run time analysis. While many previous theoretical works analyzed mutational GP with the

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 42–54, 2018.
https://doi.org/10.1007/978-3-319-99259-4_4

offspring produced by varying a single parent, we analyze a GP algorithm employing a simple crossover with the offspring produced from two parents. Although our crossover is far from practical applications of GP (it merely concatenates the two parent trees), this simple setting aims at understanding the interplay between (our variant of) crossover, the problem of bloat and *lexicographic parsimony pressure*, a method for bloat control introduced in [14]. Other theoretical work in GP has analyzed different problems and phenomena, in particular for the Probably Approximately Correct (PAC) learning framework [10], the Max-Problem [5,11,13] as well as Boolean functions [15,16,18].

For the effects of bloat in the sense of redundant parts in the tree, we draw on previous theoretical works that analyzed this phenomenon, especially [2,19]. In these, the fitness function MAJORITY as introduced in [6] was analyzed. Individuals for MAJORITY are binary trees, where each inner node is labeled J (short for *join*, but without any associated semantics) and leaves are labeled with variable symbols; we call such trees *GP-trees*. The set of variable symbols is $\{x_1, \ldots, x_n\} \cup \{\overline{x}_1, \ldots, \overline{x}_n\}$, for some n. In particular, variable symbols are paired: x_i is paired with $\overline{x}_i$. For MAJORITY, we call a variable symbol x_i *expressed* if there is a leaf labeled x_i and there are at least as many leaves labeled x_i as there are leaves labeled $\overline{x}_i$; the positive instances are in the majority. The fitness of a GP-tree is the number of its expressed variable symbols x_i. This setting captures two important aspects of GP: variable length representations and that any given functionality can be achieved by many different representations. However, the tree-structure, typically crucial in GP problems, is completely unimportant for the MAJORITY function.

Table 1. Overview of the results of the paper. A check mark denotes optimization in polynomial time with high probability, a cross denotes superpolynomial optimization time. A check mark with a subscript e denotes the results obtained experimentally.

Problem class	Local search		Crossover
	w/bloat control	w/o bloat control	w/bloat control
$+c$-MAJORITY	× Theorem 1	$\checkmark_e$ Figure 2	✓ Theorem 10
2/3-MAJORITY	✓ Theorem 2	$\checkmark_e$ Figure 2	✓ Theorem 10
2/3-SUPERMAJORITY	✓ Theorem 5	× Theorem 6	✓ Theorem 13

We know that MAJORITY can be efficiently optimized by a mutational GP called $(1+1)$ GP (see Algorithm 1 for details, basically performing a randomized local search). This holds in the case preferring shorter representations by lexicographic parsimony pressure, as shown in [19], as well as in the case without such preference [2]. Similar to recent literature on theory of GP, we will consider lexicographic parsimony pressure as our method of bloat control and henceforth only speak of *bloat control* to denote this method. We note, however, that the GP literature knows many more methods for controlling bloat which is beyond the scope of our theoretical analysis.

In addition to weighted versions of Majority, another, similar fitness function Order (see also [3,20]) has been considered, but neither of these provide us with a strong differences in the optimization behavior of different GP algorithms. Thus, we propose three variants of Majority, called $+c$-Majority, 2/3-Majority and 2/3-SuperMajority, which negatively affect the optimization of certain GP algorithms.

For $+c$-Majority a variable is expressed if its positive literals are not only in the majority, but also there has to be at least c more positive than negative literals. On the one hand, we show that a random GP-tree with a linear number of leaves expresses any given variable with constant probability . On the other hand, with constant probability such a tree has a majority of negative literals of any given variable (indeed, there is a constant probability that the variable has neither positive nor negative literals in the GP-tree). This yields a plateau of equal fitness which can only be overcome by adding c positive literals, i.e., we need a rich set of neutral mutations that allow genetic drift to happen. Bloat control suppresses this genetic drift by biasing the search towards smaller solutions. Specifically, it may not allow to add positive literals one by one, which results in an infinite run time (see Lemma 1). Note that allowing the local search to add c leaves at the same time still results only in a small chance of $O(n^{-c})$ of jumping the plateau. Hence, the $+c$-Majority fitness function serves as an example where bloat control explicitly harms the search.

For 2/3-Majority, a variable is expressed if its positive literals hold a 2/3 majority, i.e., if 2/3 of all its literals are positive. The fitness associated with 2/3-Majority is the number of expressed variables while for 2/3-SuperMajority each expressed variable contributes a score between 1 and 2, where larger majorities give larger scores (see Sect. 2 for details). The variant 2/3-SuperMajority is utilized to aggravate the effect of bloat since it rewards large numbers of (positive) literals. We show that local search with bloat control is efficient for these two problems (Theorems 2 and 5). However, without bloat control local search fails on 2/3-SuperMajority due to bloat (see Theorem 6).

Regarding optimization without bloat control, we obtain experimental results as depicted in Fig. 2. They provide a strong indicator that, when no bloat control is applied, optimization of $+c$-Majority is efficient, in contrast to the case of bloat control. The trend for 2/3-Majority indicates that optimization proceeds significantly more slowly without bloat control than with bloat control. Nevertheless, optimization seems to be feasible in contrast to the case of 2/3-SuperMajority.

Subsequently, we study a simple crossover which works as follows. The algorithm maintains a population of λ individuals, which are initialized randomly before a local search with bloat control is performed for a number of iterations. As a local search we employ the $(1+1)$ GP, a simple mutation-only GP which iteratively either adds, deletes, or substitutes a vertex of the tree. We employ this algorithm for a number of rounds large enough to ensure that each vertex of the tree has been considered for deletion at least once with high probability, which aims at controlling bloat. Afterwards, the optimization proceeds in rounds; in

each round, each individual t_0 is mated with a random other individual t_1 by joining t_0 and t_1 to obtain a tree t' which contains both t_0 and t_1; then local search is performed on t' as before yielding a tree t''. If t'' is at least as fit as t_0, we replace t_0 in the population by t''. The algorithm is called *Concatenation* since it joins two individuals, which is basically a concatenation. It is different from other approaches for memetic crossover GP as found, for example, in [4]. Note that this crossover is very different from GP crossovers found in the literature because of its almost complete disregard for the tree structure of the individuals. However, this crossover already highlights some benefits which can be obtained with crossover, and it has the great advantage of being analyzable.

We show that the Concatenation Crossover GP with bloat control efficiently optimizes all three test functions $+c$-Majority, 2/3-Majority as well as 2/3-SuperMajority, due to its ability to combine good solutions (see Theorem 10). We summarize our findings in Table 1.

In Sect. 2 we state the formal definitions of algorithms and problems, as well as the mathematical tools we use. Section 3 gives the results for local search *with* bloat control, Sect. 4 for local search *without* bloat control and Sect. 5 for the Concatenation Crossover GP. In Sect. 6 we show and discuss our experimental results, before Sect. 7 concludes the paper.

Due to space restrictions, we only provide sketches for the proofs. A full version of the paper can be found at https://arxiv.org/abs/1805.10169.

2 Preliminaries

For a given n we let $[n] = \{1, \dots, n\}$ be the set of variables. The only non-terminal (function symbol) is J of arity 2; the *terminal set* X consists of $2n$ literals, where $\overline{x}_i$ is the complement of x_i:

$$F \coloneqq \{J\}, \text{ J has arity 2}, \quad X \coloneqq \{x_1, \overline{x}_1, \dots, x_n, \overline{x}_n\}.$$

For a GP-tree t, we denote by $S(t)$ the set of leaves in t. By $S_i^+(t)$ and $S_i^-(t)$ we denote the set of leaves that are x_i-literals and $\overline{x}_i$-literals, respectively, and by $S_i(t) \coloneqq S_i^+(t) \cup S_i^-(t)$ we denote the set of all i-literals. By $S^+(t) \coloneqq \bigcup_{i=1}^n S_i^+(t)$ and $S^-(t) \coloneqq \bigcup_{i=1}^n S_i^-(t)$ we denote the set of all positive and negative leaves, respectively. We denote the sizes of all these sets by the corresponding lower case letters, i.e., $s(t) \coloneqq |S(t)|$, $s_i(t) \coloneqq |S_i(t)|$, etc. In particular, we refer to $s(t)$ as the *size* of t.

On the syntax trees, we analyze the problems $+c$-Majority, 2/3-Majority, and 2/3-SuperMajority, which are defined as

$$+c\text{-Majority} \coloneqq |\{i \in [n] \mid s_i^+ \geq s_i^- + c\}|\,;$$
$$2/3\text{-Majority} \coloneqq |\{i \in [n] \mid s_i \geq 1 \text{ and } s_i^+ \geq \tfrac{2}{3} s_i\}|\,;$$
$$2/3\text{-SuperMajority} \coloneqq \sum_{i=1}^{n} f_i, \text{ where } f_i \coloneqq \begin{cases} 0 & \text{, if } s_i = 0 \text{ or } s_i^+ < \frac{2}{3} s_i, \\ 2 - 2^{s_i^- - s_i^+} & \text{, otherwise.} \end{cases}$$

We call a variable contributing to the fitness *expressed*. Since both $+c$-Majority and 2/3-Majority count the number of expressed variables, they take values between 0 and n. The function 2/3-SuperMajority is similar to 2/3-Majority, but if a 2/3 majority is reached 2/3-SuperMajority awards a bonus for larger majorities: the term f_i grows with the difference $s_i^+ - s_i^-$. Since $f_i \leq 2$, the function 2/3-SuperMajority takes values in $[0, 2n]$. Note that the value $2n$ can never actually be reached, but can be arbitrarily well approximated.

In this paper we consider simple mutation-based genetic programming algorithms which use a modified version of the Hierarchical Variable Length (HVL) operator [21,22] called HVL-Prime as discussed in [3]. HVL-Prime allows trees of variable length to be produced by applying three different operations: insert, delete and substitute (see Fig. 1). Each application of HVL-Prime chooses one of these three operations uniformly at random. We note that the literature also contains variants of the mutation operator that apply several such operations simultaneously (see [3,20]).

Given a GP-tree t, mutate t by applying HVL-Prime. For each application, choose uniformly at random one of the following three options.	
substitute	Choose a leaf uniformly at random and substitute it with a leaf in X selected uniformly at random.
insert	Choose a node $v \in X$ and a leaf $u \in t$ uniformly at random. Substitute u with a join node J, whose children are u and v, with the order of the children chosen uniformly at random.
delete	Choose a leaf $u \in t$ uniformly at random. Let v be the sibling of u. Delete u and v and substitute their parent J by v.

Fig. 1. Mutation operator HVL-Prime.

The first algorithm we study is the $(1+1)$ GP. The algorithm is initialized with a tree generated by s_{init} random insertions. Afterwards, it maintains the best-so-far individual t. In each round, it creates an offspring of t by mutation. This offspring is discarded if its fitness is worse than t, otherwise it replaces t. We recall that the fitness in the case with bloat control contains the size as a second order term. Algorithm 1 states the $(1+1)$ GP more formally.

Algorithm 1. $(1+1)$ GP with mutations according to Figure 1

```
1 Let t be a random initial tree of size s_init;
2 while optimum not reached do
3 |   t' ← mutate(t);
4 |   if f(t') ≥ f(t) then t ← t';
```

2.1 Crossover

The second algorithm we consider is population-based. When introduced by Koza [12], Genetic Programming used fitness-proportionate selection and a genetic crossover, however mutation was hardly considered. In subsequent works many different setups for the crossover operator were introduced and studied. For instance, in [21] combinations of GP with local search in the form of mutation operators were studied and yielded better performance than GP.

Usually, two parents (a *current solution* and a *mate*) are used to generate a number of offspring. These offspring are a recombination of the alleles from both parents derived in a probabilistic manner. By modeling each individual as a GP-tree, a crossover-point in both parents is decided upon due to a heuristic and the subtrees attached to these points are exchanged creating new GP-trees.

In the Crossover hill climbing algorithm first described by Jones [7,8] only one GP-tree is created from the current solution and a random mate. This offspring is evaluated and replaces the current solution if the fitness is not worse.

We consider the following simple crossover: the *Concatenation Crossover GP* working as follows (see also Algorithm 2). For a fixed population of GP-trees, each GP-tree is chosen to be the parent once. For each parent we choose a mate uniformly at random from the population and create one offspring by *joining* the two trees using a new join-node. Before evaluating the offspring, we employ a local search in the form of the $(1+1)$ GP with bloat control. This local search is performed for a fixed amount of iterations before we discard the GP-tree with worse fitness. The fixed amount depends on the size of the tree and ensures the absence of redundant leaves with high probability (see Lemma 11). We note that the amount of redundant leaves depends on the function to be optimized. The functions we studied are variants of Majority, for other functions the amount of iterations ensuring the absence of redundant leaves might be different.

The initial population is generated by creating λ random trees of size s_{init} and employing the local search on each of them. We then proceed in rounds of crossover as described above. We note that we assume all crossover operations to be performed in parallel. Hence, the new population is based entirely on the old population and not partially on previously generated individuals of the new generation.

2.2 Terminology

For the analysis, it will be helpful to partition the set of leaves into three classes as follows. The set $C^+(t) \subseteq S^+(t)$ of *positive critical leaves* is the set of leaves u, whose deletion from the tree results in a decreased fitness. Similarly, the set $C^-(t) \subseteq S^-(t)$ of *negative critical leaves* is the set of leaves u, whose deletion from t results in an increased fitness. Finally, the set $R(t) \coloneqq [n] \setminus (C^+(t) \cup C^-(t))$ of *redundant leaves* is the set of all leaves u, whose deletion from t does not affect the fitness. Similar as before, we denote $c^-(t) = |C^-(t)|$, $c^+(t) = |C^-(t)|$, and $r(t) = |R(t)|$.

Algorithm 2. Concatenation Crossover-GP

Let LS(t) denote local search by the $(1+1)$ GP with bloat control on tree t for $90s \log s$ steps, where s is the number of leaves in t;
for $i = 1$ ***to*** λ **do**
  Let t_i be a random initial tree of size s_{init};
  $t_i \leftarrow \text{LS}(t_i)$;
while *optimum not reached* **do**
  for $i = 1$ ***to*** λ **do**
    Choose $m \in \{1, \dots, \lambda\} \setminus \{i\}$;
    $t_i' \leftarrow \text{join}(t_i, t_m)$;
    $t_i'' \leftarrow \text{LS}(t_i')$;
    if $f(t_i'') \geq f(t_i)$ **then** $t_i \leftarrow t_i''$;

Given a time $\tau \geq 0$, we denote by t_τ the GP-tree after τ iterations of the algorithm. Additionally, we use $S(\tau), s(\tau), S_i(\tau), \dots$ in order to denote $S(t_\tau), s(t_\tau), S_i(t_\tau), \dots$. Moreover, we apply the standard Landau notation $\text{O}(\cdot)$, $\text{o}(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$, $\Theta(\cdot)$ as detailed in [1].

3 $(1+1)$ GP with Bloat Control

In this section we study how local search with bloat control performs on the given fitness functions. Theorem 1 shows that for small initial trees $+c$-Majority cannot be efficiently optimized, while Theorem 2 shows that this is possible for 2/3-Majority. Finally, Theorem 5 considers 2/3-SuperMajority.

Theorem 1. *Consider the $(1+1)$ GP on $+c$-Majority with bloat control on the initial tree with size $s_{\text{init}} < n$. If $c > 1$, with probability equal to 1, the algorithm will never reach the optimum.*

The proof is based on an optimal GP-tree for $+c$-Majority needing cn leaves, but bloat control does not allow to add leaves without fitness gain.

Next we state the upper bound for the performance on 2/3-Majority. The proof of Theorem 2 is almost identical to the one of Theorem 4.1 in [2], the bounds stated in Lemma 4.2 and Lemma 4.1 in [2] need to be suitably adjusted, since these do not hold for 2/3-Majority.

Theorem 2. *Consider the $(1+1)$ GP on 2/3-Majority with bloat control on the initial tree with size s_{init}. The expected time until the algorithm computes the optimum is in $\text{O}(n \log n + s_{\text{init}})$.*

Corollary 3. *Consider the $(1+1)$ GP on 2/3-Majority with bloat control on the initial tree with size $s_{\text{init}} < n$. The expected time until the algorithm computes the optimum is in $\text{O}(n \log n)$.*

We turn to 2/3-SuperMajority with Theorem 5. The proof is based on the following lemma showing that redundant leaves will be removed with sufficient probability. Hence, insertions of positive literals can increase fitness.

Lemma 4. *Consider the (1 + 1) GP on* 2/3-SuperMajority *with bloat control with* $n \geq 55$ *on the initial tree with size* $s_{\text{init}} < n$*. With probability at least* $1 - (\tau/(n \log^2 n))^{-1/(1+4/\sqrt{\log n})}$ *the algorithm will delete any given negative leaf of the initial tree within* $\tau \geq n \log^2 n$ *rounds. For a positive redundant leaf, with the same probability it will either be deleted or turned into a positive critical leaf.*

Theorem 5. *Consider the (1 + 1) GP on* 2/3-SuperMajority *with bloat control on an initial tree with size* $s_{\text{init}} < n$*, and let* $\varepsilon > 0$*. Then, the algorithm will express every literal after* $n^{2+\varepsilon}$ *iterations with probability* $1 - \mathrm{o}(1)$*.*

4 (1 + 1) GP Without Bloat Control

In this section we study the fitness function 2/3-SuperMajority, which facilitates bloat of the string.

Theorem 6. *For any constant* $\nu > 0$*, consider the (1 + 1) GP without bloat control on* 2/3-SuperMajority *on the initial tree with size* $s_{\text{init}} = \nu n$*. There is* $\varepsilon = \varepsilon(\nu) > 0$ *such that, with probability* $1 - \mathrm{o}(1)$*, an* ε*−fraction of the indices will never be expressed. In particular, the algorithm will never reach a fitness larger than* $(2 - 2\varepsilon)n$*.*

We commence with some preparatory lemmas before proving the theorem. First, we analyze how the size of the GP-tree evolves over time. We recall that $s(\tau)$ is the number of leaves of the GP-tree at time τ.

Lemma 7. *There is a constant* $0 < \eta \leq 1$ *such that, with probability* $1 - \mathrm{o}(1)$*, for all* $\tau \geq 0$ *we have* $s(\tau) \geq \eta\tau$*.*

In order to continue we need some more terminology. For an index $i \in [n]$, we recall that $s_i^+(\tau)$ and $s_i^-(\tau)$ denote the number of x_i- and $\overline{x}_i$-literals at time τ, respectively, and $s_i(\tau) \coloneqq s_i^+(\tau) + s_i^-(\tau)$. We call index i *touched* in round τ, if a literal x_i or $\overline{x}_i$ is deleted, inserted or substituted, or if a literal is substituted by x_i or $\overline{x}_i$. We call the touch *increasing* if it is either an insertion or if a literal is substituted by x_i or $\overline{x}_i$. We call the touch *decreasing* if it is a deletion or substitution of a x_i or $\overline{x}_i$ literal. We note that in exceptional cases a substitution may be both increasing and decreasing. Let $\rho_i(\tau)$ be the number of increasing touches of i up to time τ. We call a decreasing step *critical* if it happens at time τ with $s_i(\tau) \leq \eta\tau/(4n)$, and we call $\gamma_i(\tau)$ the number of critical steps up to time τ. Finally, we call a round *accepting* if the offspring is accepted in this round.

The approach for the remainder of the proof is as follows. First, we will show that in the regime, where critical steps may happen (i.e, $s_i(\tau) \leq \eta\tau/(4n)$), it is more likely to observe increasing than decreasing steps. The reason is that a step is only critical if there are relatively few i-literals, in which case it is unlikely to

delete or substitute one of them, whereas the probability to insert an i-literal is not affected. It will follow that $s_i(\tau)$ grows with τ, since otherwise we would need many critical steps. Finally, if $s_i(\tau)$ keeps growing it becomes increasingly unlikely to obtain a 2/3 majority. In order to state the first points more precisely we fix a $j_0 \in \mathbb{N}$ and call an index i *bad* (or more precisely, j_0-bad) if the following conditions hold: for all $\tau \geq j_0 n$ and $\tau_0 \coloneqq j_0 n$

$$\begin{array}{ll} \text{(A) } s_i^+(\tau_0) \leq s_i^-(\tau_0) \leq j_0 & \text{(B) } \tau/(2n) \leq \rho_i(\tau) \leq 2\tau/n \\ \text{(C) } \gamma_i(\tau) \leq 2\tau/n & \text{(D) } s_i(\tau) \geq \eta\tau/(8n). \end{array}$$

In particular, in (A) x_i is not expressed at time τ_0.

Lemma 8. *For every fixed $i_0 > 0$, with probability $1 - o(1)$ there are $\Omega(n)$ bad indices.*

Lemma 9. *Every bad index has probability $\Omega(1)$ that it is never expressed, independent of the other bad indices.*

We note that Lemmas 8 and 9 imply Theorem 6 by a straightforward application of the Chernoff bound.

5 Concatenation Crossover GP

In the following we will study the performance of the Concatenation Crossover GP (Algorithm 2) on $+c$-Majority and 2/3-Majority with bloat control. As observed in Theorem 1 the $(1+1)$ GP with bloat control may never reach the optimum when optimizing an initial tree of size $s_{\text{init}} < n$. We will deduce that crossover solves this issue and the algorithm reaches the optimum fast. We commence this section by stating the exact formulation of the mentioned result in Theorem 10 followed by an outline of its proof. Finally, we show the corresponding result for 2/3-SuperMajority in Theorem 13.

Theorem 10. *Consider the Concatenation Crossover GP on $+c$-Majority or 2/3-Majority with bloat control on the initial tree with size $2 \leq n/2 \leq s_{\text{init}} \leq bn$ (for constant $b > 0$). Then there is a constant $c_\lambda > 0$ such that for all $c_\lambda \log n \leq \lambda \leq n^2$, with probability in $(1 - O(n^{-1}))$, the algorithm reaches the optimum after at most $O(n \log^3(n))$ steps.*

The following two auxiliary lemmas are used to proof the theorem. Here, they serve towards an outline of the proof. First, Lemma 11 states the absence of redundant leaves in a GP-tree t after the local search with a probability of $1 - n^{-5}$. This will be applied after every local search. We observe for two GP-trees t_1 and t_2 *without redundant leaves*: if t' is the tree resulting from joining t_1 and t_2, then a variable $i \in [n]$ is expressed in t' if and only if it is expressed in t_1 or t_2.

Second, Lemma 12 states that, with a probability of $1-n^{-5}$, each variable $i \in [n]$ is expressed in at least one of $\lambda/2$ trees before the first crossover. Combining both lemmas, for a fixed GP-tree t it will suffice to observe the time until t has been joined with at least $\lambda/2$ different trees.

Lemma 11. *Consider the (1 + 1) GP with bloat control on either* $+c$-MAJORITY *or* 2/3-MAJORITY. *For an initial tree with size* $2 \leq n/2 \leq s_{\text{init}} \leq bn$ *(for constant* $b > 0$*) after* $90 s_{\text{init}} \log(s_{\text{init}})$ *iterations, with probability at least* $1 - n^{-5}$, *the current solution will have no redundant leaves.*

Lemma 12. *Consider the Concatenation Crossover GP on* $+c$-MAJORITY *or* 2/3-MAJORITY *with bloat control on initial trees with size* $2 \leq n/2 \leq s_{\text{init}} \leq b\,n$ *(for constant* $b > 0$*). Then there is a constant* $c_\lambda > 0$ *such that for all* $\lambda \geq c_\lambda \log n$, *with probability at least* $1 - n^{-5}$, *each variable will be expressed in at least one of* $\lambda/2$ *trees before the first crossover.*

Finally, we turn to 2/3-SUPERMAJORITY. For the proof we use a result from the area of rumor spreading relating to the *pull protocol* [9,17] in order to study the time until every individual of the population has every variable expressed. The idea here is similar to previous proofs with crossover: expressed variables can be collected with crossover. For this purpose we show that the number of x_i in individuals, which have a variable i expressed, is asymptotically larger than the number of $\overline{x}_i$ in individuals, which do not have i expressed.

Theorem 13. *Consider the Concatenation Crossover GP without substitutions with bloat control with initial tree size* $s_{\text{init}} = n/2$ *on* 2/3-SUPERMAJORITY. *Then there is a constant* $c_\lambda > 0$ *such that, for* $\lambda = c_\lambda \log n$, *each GP-tree in the population has all variables expressed after at most* $\mathrm{O}(n^{1+o(1)})$ *steps with probability at least* $1 - \mathrm{O}(n^{-4})$.

6 Experiments

This section is dedicated to complementing our theoretical results with experimental justification for the otherwise open cells of Table 1, i.e. for the $(1+1)$ GP without bloat control on $+c$-MAJORITY and 2/3-MAJORITY.

All experimental results shown in Fig. 2 are box-and-whiskers plots, where lower and upper whiskers are the minimal and maximal number of *fitness evaluations* the algorithm required over 100 runs until all variables are expressed or the time limit of 1000000 evaluations is reached. The middle lines in each box are the median values (the second quartile), the bottom and top of the boxes are the first and third quartiles. Note that all experiments are platform independent since we count number of fitness evaluations independently of real time. The solid lines in the plots allow to estimate the asymptotic run time of the $(1+1)$ GP.

The left hand side of Fig. 2 concerns $+c$-MAJORITY and shows that the $(1+1)$ GP with bloat control always fails (corresponding to Theorem 1). We used the $(1+1)$ GP with $s_{\text{init}} = 10n$, $c = 2$ and n as indicated along the x-axis. It is easy to see that bloat control leads the algorithm to local optima and does not allow to leave it, whereas the $(1+1)$ GP *without* bloat control finds an optimum in a reasonable number of evaluations. Due to time and computational restrictions the constant c was chosen equal to 2. For larger c the run time of the algorithm goes up significantly, but a similar pattern is visible.

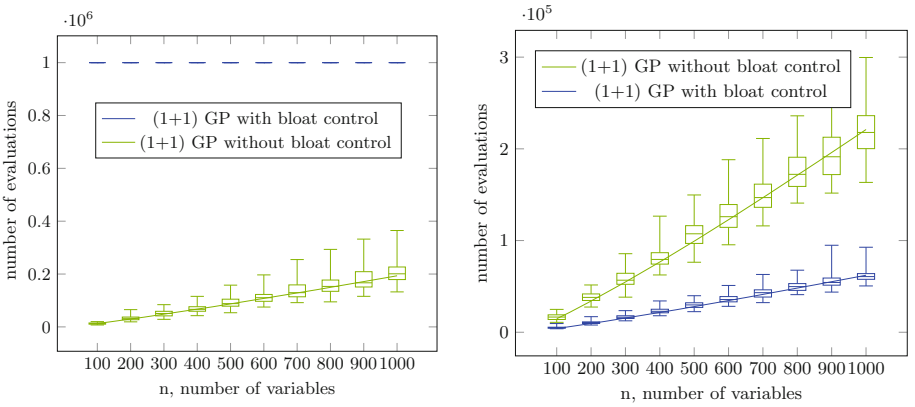


Fig. 2. Number of evaluations required by the $(1+1)$ GP over 100 runs for each n with the initial tree size $s_{\text{init}} = 10n$ until all variables are expressed or the time limit, equal to 1000000 evaluations, is reached. The left figure shows the experimental results for $+c-$ MAJORITY with $c = 2$; the solid line is $28n \log n$. On the right figure is shown $2/3-$ MAJORITY; the blue solid line is $9n \log n$, the green solid line is $32n \log n$. (Color figure online)

The right hand side of Fig. 2 shows the results of $(1+1)$ GP on 2/3-MAJORITY, using $s_{\text{init}} = 10n$. One can see that bloat control is more efficient in comparison with the $(1+1)$ GP without bloat control. The set of median values is well-approximated by $w \cdot n \log n$ for a constant w, which leads us to the conjecture that the algorithm's run time is $O(n \log n)$. We did not analyze the influence of s_{init}, but it might be significant especially for 2/3-MAJORITY without bloat control.

7 Conclusion

We defined three variants of the MAJORITY problem in order to introduce some fitness plateaus that are difficult to cross. The $+c$-MAJORITY allows for progress at the end of the plateau with large representation; in this sense, bloat is necessary for progress. On the other hand, for 2/3-MAJORITY, progress can be made at the end of the plateau with small representation, so that bloat control guides the search to the fruitful part of the search space. We also considered 2/3-SUPERMAJORITY which exemplifies fitness functions where bloat is inherent due to the possibility of small improvements by adding an increasing amount of nodes to the GP-tree. In this case we showed that not employing bloat control leads to inefficient optimization.

In order to obtain results somewhat closer to practically relevant GP we turned to crossover and showed how a Concatenation Crossover GP can efficiently optimize all three considered test functions.

For future work it might be interesting to analyze the effect of other crossover operators. In order to obtain a better understanding of such other operators,

other test functions might be necessary making essential use of the tree structure (all our test functions might as well use lists or even multisets of the leaves as representations). Such test functions should not be too complex, which would hinder a theoretical analysis, but still embody a structure frequently found in GP, so as to inform about relevant application areas. The search for such test functions remains a central open problem of the theory of GP.

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
2. Doerr, B., Kötzing, T., Lagodzinski, J.A.G., Lengler, J.: Bounding bloat in genetic programming. In: Proceedings of GECCO 2017, pp. 921–928. ACM (2017)
3. Durrett, G., Neumann, F., O'Reilly, U.M.: Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics. In: Proceedings of FOGA 2011, pp. 69–80 (2011)
4. Eskridge, B.E., Hougen, D.F.: Memetic crossover for genetic programming: evolution through imitation. In: Deb, K. (ed.) GECCO 2004. LNCS, vol. 3103, pp. 459–470. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24855-2_57
5. Gathercole, C., Ross, P.: An adverse interaction between the crossover operator and a restriction on tree depth. In: Proceedings of GP 1996, pp. 291–296 (1996)
6. 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). https://doi.org/10.1007/BFb0055925
7. Jones, T.: Crossover, macromutation, and population-based search. In: Proceedings of ICGA 1995, pp. 73–80. Morgan Kaufmann Publishers Inc. (1995)
8. Jones, T.: Evolutionary algorithms, fitness landscape and search. Ph.D. thesis, University of New Mexico (1995)
9. Karp, R.M., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized rumor spreading. In: Proceedings of FOCS 2000, pp. 565–574 (2000)
10. Kötzing, T., Neumann, F., Spöhel, R.: PAC learning and genetic programming. In: Proceedings of GECCO 2011, pp. 2091–2096 (2011)
11. Kötzing, T., Sutton, A.M., Neumann, F., O'Reilly, U.M.: The max problem revisited: the importance of mutation in genetic programming. In: Proceedings of GECCO 2012, pp. 1333–1340 (2012)
12. Koza, J.R.: Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford, CA, USA (1990)
13. Langdon, W.B., Poli, R.: An analysis of the MAX problem in genetic programming. In: Proceedings of GP 1997, pp. 222–230 (1997)
14. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: Proceedings of GECCO 2002, pp. 829–836 (2002)
15. Mambrini, A., Manzoni, L.: A comparison between geometric semantic GP and cartesian GP for Boolean functions learning. In: Proceedings of GECCO 2014, pp. 143–144 (2014)

16. Mambrini, A., Oliveto, P.S.: On the analysis of simple genetic programming for evolving Boolean functions. In: Proceedings of EuroGP 2016, pp. 99–114 (2016)
17. Mercier, H., Hayez, L., Matos, M.: Optimal epidemic dissemination. CoRR abs/1709.00198 (2017). http://arxiv.org/abs/1709.00198
18. Moraglio, A., Mambrini, A., Manzoni, L.: Runtime analysis of mutation-based geometric semantic genetic programming on Boolean functions. In: Proceedings of FOGA 2013, pp. 119–132 (2013)
19. Neumann, F.: Computational complexity analysis of multi-objective genetic programming. In: Proceedings of GECCO 2012, pp. 799–806 (2012)
20. Nguyen, A., Urli, T., Wagner, M.: Single- and multi-objective genetic programming: new bounds for weighted ORDER and MAJORITY. In: Proceedings of FOGA 2013, pp. 161–172 (2013)
21. O'Reilly, U.M.: An analysis of genetic programming. Ph.D. thesis, Carleton University, Ottawa, Canada (1995)
22. O'Reilly, U.-M., Oppacher, F.: Program search with a hierarchical variable length representation: genetic programming, simulated annealing and hill climbing. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) PPSN 1994. LNCS, vol. 866, pp. 397–406. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58484-6_283

Exploration and Exploitation Without Mutation: Solving the *Jump* Function in $\Theta(n)$ Time

Darrell Whitley[1(✉)], Swetha Varadarajan[1], Rachel Hirsch[1], and Anirban Mukhopadhyay[2]

[1] Colorado State University, Fort Collins, CO 80523, USA
{whitley,swetha.varadarajan,rachel.hirsch}@colostate.edu
[2] University of Kalyani, Kalyani, Nadia 741235, West Bengal, India
anirban@klyuniv.ac.in

Abstract. A number of modern hybrid genetic algorithms do not use mutation. Instead, these algorithms use local search to improve intermediate solutions. This same strategy of combining local search and crossover is also used by stochastic local algorithms, such the LKH heuristic for the Traveling Salesman Problem. We prove that a simple *hybrid* genetic algorithm that uses only local search and a form of deterministic "voting crossover" can solve the well known *Jump* Function in $\Theta(n)$ time where the jump distance is $log(n)$.

1 Introduction

The *Jump* function is a function of unitation that uses the $OneMax(x)$ function as an intermediate form in the construction of the evaluation function. Functions of unitation [15,16] are pseudo-Boolean functions where all bit string inputs that have the same number of 1 bits have exactly the same evaluation. The *Jump* evaluation function first computes $OneMax(x)$, the number of 1 bits in string x. A "gap" or "moat" is then created that the search must jump across to reach the global optimum, where the global optimum is the string of all 1 bits [1].

This paper proposes a novel approach to solving the *Jump* function. A hybrid genetic algorithm is used that improves the population using local search. The hybrid genetic algorithm also uses a form of multi-parent recombination. This hybrid genetic algorithm easily solves instances of the *Jump* function in $\Theta(n)$ time when the jump distance is $log(n)$. The use of a hybrid genetic algorithm is, in fact, common practice in that part of the evolutionary computation community concerned with real world applications. Furthermore, if local search can provide diversity, there is no need for mutation. Many highly effective evolutionary algorithms do not use mutation. Thus, a meta-level goal of this work is to make theory more relevant to the community as a whole by focusing on hybrid genetic algorithms, the aggressive use of recombination, and the role of diversity in genetic algorithms.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 55–66, 2018.
https://doi.org/10.1007/978-3-319-99259-4_5

2 Background and Basics

Let x denote a bit string, let n denote string length, and let m be width of the gap that must be jumped to reach the global optimum. The $Jump$ function is then defined as follows.

$$Jump_{m,n}(x) = \begin{cases} m + OneMax(x) & \text{if } OneMax(x) \leq (n-m) \\ & \text{or } OneMax(x) = n \\ n - OneMax(x) & \text{otherwise} \end{cases}$$

where $OneMax(x)$ denotes the number of 1 bits in string x. The $Jump_{m,n}$ function is illustrated in the left side of Fig. 1. Another way to think about the $Jump$ function is also shown in the right side of Fig. 1. The $Jump$ function represents a worst case situation in as much as the global optimum is located on a single point surrounded by a moat (the "gap" that must be jumped). Otherwise, the entire landscape is a symmetric hill. We can think of the edge of the "moat" as being a ridge encircling the global optimum, since all points at the edge of the moat have the same evaluation. There are $Choose(n, m)$ points in the search space at the edge of the moat. The local optima at the edge of the moat are not a connected plateau under 1 bit flip, since flipping 1 bit increases or decreases the number of 1 bits in the string. Thus all of the $Choose(n, m)$ points at the edge of the "moat" are distinct local optima. But the same level set of points are a single connected plateau under a 2 bit flip neighborhood operator.

Under "Black Box Complexity" the number of calls to the evaluation function is typically used in place of the true runtime cost. In this paper we consider the true runtime cost. Under normal black box optimization, each execution of the $Jump$ evaluation function takes $\Theta(n)$ time, since one must count all of the 1 bits in string x in order to compute $OneMax(x)$. However, we can also create an incremental, partial evaluation function that will execute in $\Theta(1)$ time.

We create an auxillary function $Jump_{m,n}(x) = Jump1_{m,n}(x'_i, eval(x'))$ where string x is created by flipping the bit x'_i in string x' to generate string x, and $eval(x') = Jump_{m,n}(x')$ stores the evaluation of string x'.

This will allow us to create alternative implementations of $OneMax$ where we invert the $eval(x') = Jump_{m,n}(x')$ function to calculate OneMax(x').

$$OneMax(x') = \begin{cases} Jump_{m,n}(x') - m & \text{if } Jump_{m,n}(x') > m \\ n - Jump_{m,n}(x') & \text{otherwise} \end{cases}$$

We can then create an incremental update to calculate OneMax(x).

$$OneMax(x) = \begin{cases} OneMax(x') + 1 & \text{if } x'_i = 0 \\ OneMax(x') - 0 & \text{otherwise} \end{cases}$$

Lemma 1. *An incremental implementation of the Jump evaluation function can be executed in $\Theta(1)$ time when evaluating $Jump_{m,n}(x) = Jump1_{m,n}(x'_i, eval(x'))$ assuming $eval(x_i)$ is given, and x and x'_i are Hamming distance 1 apart.*

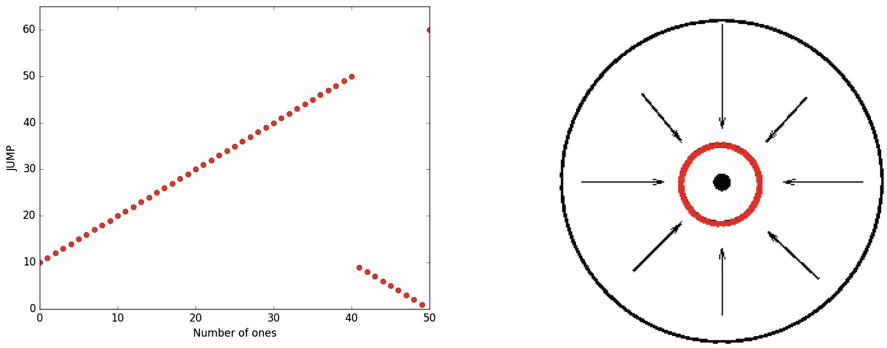


Fig. 1. An instance of the $Jump_{m,n}(x)$ function, with $m = 10$ and $n = 50$ is shown on the left. Another way of visualizing the $Jump$ function is shown on the right. The global optimum is at the center of a hill, surrounded by a "moat". At the edge of the moat (shown in red) are $Choose(n, m)$ local optima under a single bit-flip neighborhood with exactly the same evaluation. For crossover to be effective, there must be a diverse set of solutions distributed along the edge of the moat. (Color figure online)

Proof. Assuming $eval(x'_i)$ is given, then the $Jump_{m,n}(x')$ function can be inverted in order to compute $OneMax(x')$ in $\Theta(1)$ time. $OneMax(x)$ can be computed in $\Theta(1)$ time given x'_i and the $OneMax(x')$ evaluation. The original $Jump$ function now executes in $\Theta(1)$ time given the $OneMax(x)$ evaluation. □

2.1 Jansen's and Wegener's Classic Result

The $Jump$ function suggests two very simple questions: given a multi-modal, nonlinear function, can an evolutionary algorithm jump across short barriers in the search space? Second, can crossover be useful in solving this class of functions, or any other class of functions?

Jansen and Wegener [1] prove that a $(1 + 1)ES$ using a mutation rate of $1/n$ and a gap of m bits has an expected running time of $\Theta(n^m + n\ log\ n)$. They then consider a relatively standard steady state genetic algorithm, with two restrictions. First, they disallow duplicate strings, meaning that the same string cannot occur in the population more than once. Second, the crossover probabilities were unusually small.

When a genetic algorithm moves along a trajectory from a randomly generated population to a location on the edge of the moat, it tends to converge to a small localized region on the edge of the moat because the population has lost diversity. But for crossover to jump across the moat, the population must be diverse. In effect, the population needs to surround the moat.

After the population reaches the edge of the moat, mutation must be high enough to scatter and spread the population around the edge of the moat. At the same time, crossover and selection must be low enough to allow this increase in diversity. Thus, as stated by Jansen and Wegener, only in the final phase of

search is crossover critical. They provide the following expected running time for their genetic algorithm:

$$\Theta(\mu n(m(n)^2 + log(\mu n)) + 2^{2m(n)}/pc)$$

where μ is the population size, and pc is the probability of crossover.

3 Hybrid Genetic Algorithms

The work of Jansen and Wegener was groundbreaking. However, there is still a tendency in the theory community to overly focus on the (1+1)ES, or Holland's Simple Genetic Algorithm.

We consider instead a "hybrid genetic algorithm" or"memetic algorithm" [11] where the population is improved by applying local search to all of the individuals in the population every generation. We will use the "next ascent bit climber" introduced by Dave Davis [4] to the genetic algorithm community. The next ascent bit climber generates a random permutation, then flips the bits in the order indicated by the permutation. The "next ascent bit climber" accepts each improvement as it is found. After every bit has been flipped once, the process is repeated with a new permutation until a local optimum is reached.

In the case of the *Jump* function, the initial population that is improved by local search will either (1) include the global optimum by chance, or (2) reaches the global optimum by improving a randomly sampled string adjacent to the global optimum, or (3) all of the strings are at the edge of the "moat". Sampling the global optimum by chance occurs with probability $1/2^n$ per sample. Sampling a point adjacent to the global optimum occurs with probability $n/2^n$; however, the probability that "next ascent" will select exactly the right bit to improve first is only $1/n$. Thus, we will conservatively assume the global optimum is not found without recombination.

Lemma 2. *Assume a random initial population has been generated that is then improved by the "next ascent bit climber" local search. Assuming the global optimum is not generated randomly, or discovered by local search, then any constant size population improved by using the "next ascent bit climber" will be uniformly distributed around the edge of the "moat" in* $\Theta(n)$ *time.*

Proof. It requires $\Theta(n)$ time to evaluate each member of the initial population using a standard (not incremental) form of the $Jump_{m,n}(x)$ function. Assuming the population size is bounded by a constant, the initial population is evaluated in $\Theta(n)$ time. Next, a $\Theta(1)$ implementation of the *Jump* function can be used to implement local search. Because "next ascent bit climber" tries every bit in the string once before flipping any bit a second time, the bit climber must reach the edge of the moat in at most n evaluations for every string in the population. We can confirm the point is a local optimum by attempting another n bit flips. Since each initial string was randomly selected, and the order of improving moves is randomized as well, local search is equally likely to yield any point on the edge of the moat. This work requires $\Theta(n)$ time. □

3.1 Deterministic Crossover: 3-Parent Voting Crossover

A number of simple test problems can be solved by exploiting low level hyperplane information. This is also true for the *Jump* function; it is trivial to compute the averages of the first order hyperplane subspaces for functions of unitation, and it is trivial to prove that any order-1 hyperplane with a single 1 bit in any position is better on average than any order-1 hyperplane with a single 0 bit in any position for the *Jump* function. Any crossover operator that can effectively exploit first order hyperplane averages has a clear advantage.

Recently, a number of deterministic crossover operators have been proven to be highly effective on classic NP-Hard problems. The *partition crossover* operator has produced excellent results on large, one million variable NK-Landscapes [2,17] without using mutation. The LKH search algorithm for the TSP also uses Iterative Partial Transcription (IPT) [9,10,12]. In the area of scheduling, Deb and Myburgh [5] used a deterministic form of *block crossover* and deterministic repair operators (which they call "mutation" operators) to generate near optimal solutions to one billion variable cast scheduling problems. All of these algorithms use deterministic crossover operators to solve classic NP-Hard problems but none of these very modern evolutionary algorithms uses random mutation operators.

Perhaps the best operator for exploiting low level hyperplane information is "Voting Crossover." Voting crossover uses an odd number of parents (e.g., 3 parents), and then each parent "votes" for a 1 bit or a 0 bit in every bit position, where the majority wins. Thus, the result is deterministic. This operator was first introduced at the PPSN conference in a highly cited paper by Eiben et al. in 1994 [6]. It was given the name "Occurrence Based Scanning" crossover, and in its most general form it could use any number of parents. (We argue that the name "Voting Crossover" is more intuitive, more descriptive and easier to remember.) A randomized version of this crossover was also introduced called "Uniform Scanning" crossover [6]. Under "Uniform Scanning" crossover, the "vote" is interpreted probabilistically. For example, given 3 parents, if 2 parents have a 1 bit and 1 parent has a 0 bit, then the 1 bit is inherited with 2/3 probability.

Eiben et al. [6] reported that multi-parent crossover operators yields superior results on the classical DeJong test suite. On other benchmarks they considered, the results were more mixed, but overall, multi-parent crossover operators were competitive with classical operators such as 2 parent uniform crossover. A follow up study also suggested that multi-parent crossover operators are more effective on NK-Landscapes with low-epistasis [7]. This should not be surprising. One of the problems with classical uniform crossover is that all bits that are shared are inherited from the parents, but when the 2 parents differ, the bit assignment is completely random. This means that uniform crossover just randomly picks a string drawn from the largest hyperplane subspace that contains both parents.

We use 3 Parent Voting Crossover for two reasons. First, is very easy to mathematically characterize the outcome of using just 3 parents because crossover is deterministic. Second, using just 3 parents allows for some diversity to remain in the search process and the population; using 5 parents or 7 parents would create

more selection toward the bit pattern that (already) most commonly occurs in the population in that particular position. This does not matter for the *Jump* function, but could be important for other objective functions.

3.2 The Probability of Success (POS) for 3-Parent Voting Crossover

Lemma 3. *Given 3 random parent strings with m 0 bits and $(n-m)$ 1 bits (strings on the edge of the gap), Voting Crossover yields the global optimum with probability $POS(n,m)$ for the Jump function, where:*

$$POS(n,m) = \frac{(n-m)!/(n-3m)!}{(n!/(n-m)!)^2}$$

Proof. Assume you have n buckets, and have m red marbles, m green marbles and m blue marbles. We can use the red marbles to construct a bit string with m bits of 0, and $n-m$ 1 bits. Place each red marble in a random bucket that does not already contain a red marble. This yields a bit string: a bucket without a red marble is assigned a 1 bit, and a bucket with a red marble is assigned a 0 bit. A second string is constructed using the green marbles, and a third string is constructed using the blue marbles.

After 3 strings are generated, if every bucket contains at most 1 marble then 3-parent voting crossover will jump to the global optimum: in every bit position there is at most 1 vote (1 marble) for 0, and therefore 2 or more votes for 1.

Place the 1st red marble and 1st green marble and 1st blue marble randomly into a bucket. The probability of zero conflicts for these 3 events is $n/n \cdot (n-1)/n \cdot (n-2)/n$ because the marbles can go anywhere.

Place the 2nd red marble and 2nd green marble and 2nd blue marble randomly into a bucket. The probability of zero conflicts for these 3 events is:

$$((n-3)/(n-1)) \cdot ((n-4)/(n-1)) \cdot ((n-5)/(n-1))$$

Generalizing, as each marble is placed, the enumerator is decreases by 1. But the denominator is decreasing by 1 only after 1 marble of each color has been placed, every 3 steps. This yields the following general result:

$$\frac{n!/(n-3m)!}{n!/(n-m)! \cdot n!/(n-m)! \cdot n!/(n-m)!} = \frac{(n-m)!/(n-3m)!}{n!/(n-m)! \cdot n!/(n-m)!}$$

□

3.3 A Lower Bound on the Probabilities

An alternative way to calculate POS (equivalent by simple algebra) is as follows:

$$POS(n,m) = \frac{(n-m)!/(n-2m)!}{n!/(n-m)!} \cdot \frac{(n-2m)!/(n-3m)!}{n!/(n-m)!}$$

We will use this form to compute a simple but sufficient bound on the probability $POS(n, m)$. Since there are m integers in the following sequence, we automatically obtain the following result by taking the smallest number in the enumerator and the largest number in the denominator.

$$\frac{(n-(2m-1))^m}{n^m} < \frac{(n-m)!/(n-2m)!}{n!/(n-m)!}$$

By identical logic:

$$\frac{(n-(3m-1))^m}{n^m} < \frac{(n-2m)!/(n-3m)!}{n!/(n-m)!}$$

This yield a bound

$$\frac{(n-(2m-1))^m}{n^m} \cdot \frac{(n-(3m-1))^m}{n^m} < \frac{(n-m)!/(n-2m)!}{n!/(n-m)!} \cdot \frac{(n-2m)!/(n-3m)!}{n!/(n-m)!}$$

See Fig. 2. From this bound, and we can obtain the following theorem:

Lemma 4. *Assume $m = Floor(log_2(n))$. Then for all positive integers:*

$$Bound(POS(n, log_2(n)) = \frac{(n-(2m-1))^m}{n^m} \cdot \frac{(n-(3m-1))^m}{n^m}$$

is a non-decreasing function that converges in the limit to probability 1 for large n. This function is a lower bound on $POS(n, log_2(n))$.

Proof. When n is a power of 2, the following inequalities hold by simple algebra:

$$\frac{(n-(2m-1))^m}{n^m} < \frac{(2n-(2(m+1)-1))^{(m+1)}}{(2n)^{(m+1)}}$$

and

$$\frac{(n-(3m-1))^m}{n^m} < \frac{(2n-(2(m+1)-1))^{(m+1)}}{(2n)^{(m+1)}}$$

This is sufficient to prove the *Bound* function is non-decreasing.

Again assume that n is a power of 2. Now consider any integer $n + x$ such $n < n + x < 2n$ where m is given by $m = Floor(log_2(n))$. Then, by simple algebra:

$$\frac{(n-(2m-1))^m}{n^m} < \frac{((n+x)-(2m-1))^m}{(n+1)^m}$$

and

$$\frac{(n-(3m-1))^m}{n^m} < \frac{((n+x)-(3m-1))^m}{(n+1)^m}$$

□

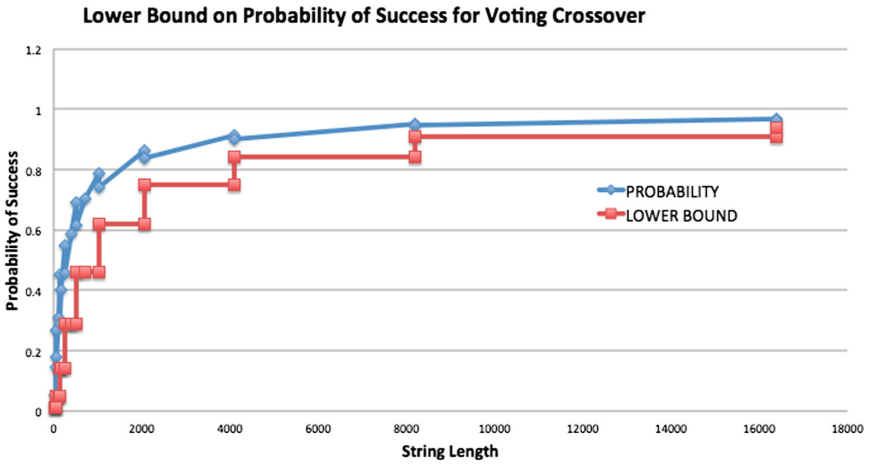


Fig. 2. The Probability of Success (POS) for Voting Crossover and the Lower Bound on the Probability of Success. The true probability is not monotonic, but the Lower Bound on the POS is a non-decreasing function that asymptotically converges to 1.0.

4 Probabilities and Populations

For Voting Crossover, even a single crossover yields a high probability of reaching the global optimum (e.g., >50%) for $n > 512$. We can use the population to boost that probability. But we will apply crossover more frequently than is normally the case in a simple genetic algorithm. Because we are using 3-parent crossover, we will allow each parent to be involved in up to 3 crossover events. This is related to the concept of "Brood Selection", where 2 parents can generate multiple offspring. "Brood Selection" is critical to the highly successful EAX algorithm for the Traveling Salesman Problem, where the number of offspring generated each generation is typically 30 times the population size [13,14]. This allows crossover to be utilized as an exploration operator.

We will assume each crossover event must be independent. For example, if we recombine the three parents $P1, P2, P3$, we will then not allow the crossover of parents $P1, P2, P4$, since $P1$ and $P2$ have already been paired in the previous crossover. For example, with a population size of 6, an independent set of recombination events might include the following subsets of parents:

$$\{\{P1, P2, P3\}, \{P1, P4, P5\}, \{P2, P4, P6\}, \{P3, P5, P6\}\}$$

where each parent is involved in 2 recombination events, but no pair of parents occurs in more than 1 recombination event. Let μ denote the population size and let λ denote the number of offspring generated in one generation. Table 1 calculates the probabilities of discovering the global optimum in the first generation of our hybrid genetic algorithm.

We next show both the theoretical results and empirical results based on 1000 runs of our hybrid genetic algorithm with Voting Crossover. Because our

Table 1. Theoretical probability analysis that the algorithm will converge to the global maximum using 3 Parent Voting Crossover. A probability of "1" in this case means that the probability is greater than 0.999999999. "μ" denoted population size, and λ denoted the number of independent offspring generated.

N	Bound	Probability ($\lambda = 1$)	$\mu = 7$ ($\lambda = 7$)	$\mu = 11$ ($\lambda = 13$)	$\mu = 13$ ($\lambda = 22$)	$\mu = 15$ ($\lambda = 35$)	$\mu = 25$ ($\lambda = 75$)
32	0.011	0.05242	0.31402	0.57747	0.69412	0.84810	0.98237
64	0.051	0.14659	0.67031	0.87263	0.96941	0.99610	0.99999
128	0.143	0.29076	0.90972	0.98851	0.99947	0.99999	1
256	0.290	0.45779	0.98622	0.99964	0.99999	1	1
512	0.461	0.61534	0.99875	0.99999	1	1	1
1024	0.622	0.74326	0.99992	0.99999	1	1	1
2^{11}	0.750	0.83653	0.99999	1	1	1	1
2^{12}	0.843	0.89954	0.99999	1	1	1	1
2^{13}	0.910	0.94830	1	1	1	1	1
2^{14}	0.943	0.96470	1	1	1	1	1
2^{15}	0.967	0.97959	1	1	1	1	1
2^{16}	0.981	0.98834	1	1	1	1	1
2^{17}	0.989	0.99340	1	1	1	1	1
2^{18}	0.993	0.99629	1	1	1	1	1
2^{19}	0.996	0.99793	1	1	1	1	1
2^{20}	0.998	0.99984	1	1	1	1	1

calculations are precise and because Voting Crossover operator is deterministic, the theoretical results and empirical results match more or less perfectly. This can be seen in Fig. 3. We can now state the overall result.

Theorem 1. *Let the gap used by the Jump function be $m = Floor(log_2(n))$. For sufficiently large n, a hybrid genetic algorithm which uses (1) a $\Theta(1)$ time incremental evaluation function, (2) a population size bounded by a constant, (3) "next ascent bit climber" local search and (4) Voting Crossover using 3 parents converges to the global optimum of the $Jump_{m,n}$ function in $\Theta(n)$ time with probability approaching 1 in one generation using no mutation, assuming each parent is allowed to be involved in up to 3 recombinations.*

Proof. Lemma 1 establishes that local search can evaluate potential solutions in $\Theta(1)$ time after the initial population has been evaluated. Lemma 2 demonstrates that the population will be uniformly distributed along the edge of the "moat" in $\Theta(n)$ time after the first generation is improved by next ascent bit climbing. By Lemma 3, the probability of generating the global optimum by a single 3 parent Voting Crossover is at least $POS(n, log_2(n)) = \frac{(n-m)!/(n-3m)!}{(n!/(n-m)!)^2}$ and by Lemma 4 this probability is bounded by a nondecreasing function that

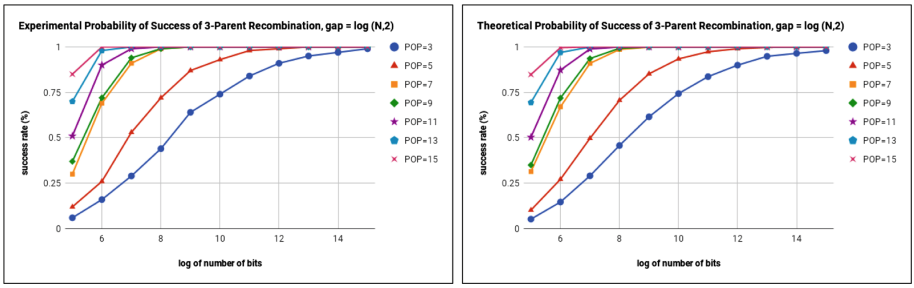


Fig. 3. The probability of finding the global optimum (jumping across the gap) in one generation for a hybrid genetic algorithm using Voting Crossover as a function of population size. The experimental data and the theoretical data match almost perfectly.

in the limit converges to 1 for large n. For smaller values of n we can use the population and multiple recombinations to boost the probability of finding the global optimum. Let $p = POS(n, log_2(n))$ and let $\lambda \leq 3\mu$ denote the number of offspring generated by *independent* recombinations, where the parents are located on the edge of the "moat" on the *Jump* function. Then the probability of finding the global optimum in the first generation is given by: $1 - (1 - p)^{\lambda}$. For all $n \geq 64$ and all $\mu = 25$, the probability of finding the global optimum is greater than 0.99999. This probability also converges to 1 in the limit for large n. The time to find a set of independent crossovers is bounded by a constant when the population is of constant size. Each crossover takes $\Theta(n)$ time. But the total number of applications of crossover is a constant when the population size is bounded by a constant. □

5 Other Crossover Operators

While multi-parent crossover and voting crossover are in literature and have been shown to be effective on some benchmarks and on low epistasis NK Landscapes, one might ask what happens when a more conventional crossover operator is used. The first answer is that the probability of successful crossover depends on m but is independent of n when m is also independent of n. Thus, if we fix m at some value such as $m = 8$, the ability of crossover to find the global optimum in $O(n)$ time can be preserved. But the "constant" involved (or more precisely, the cost depending on m) can still vary dramatically.

"Uniform Crossover" is perhaps the worst possible choice of crossover operators. Assuming maximally different parents (that are still local optima), there are 2^{2m} possible offspring, only one of which is the global optimum.

A better choice would be the HUX (Half Uniform Crossover) operator of CHC [8]. HUX determines which bits are different in two parents, then selects exactly one half of the non-shared bits from one parent and the remaining non-shared bits from the other parent. If the two parents are maximally distant from each other on the edge of the moat, then they have $2m$ bits that differ, and HUX

will select m bits from each parent. Thus, there are $Choose(2m, m)$ possible offspring (compared to 2^{2m} possible offspring under Uniform crossover). The CHC algorithm also boosts the probability that parents are maximally different by "incest prevention." Thus, when recombining parents, it prefers to pair parents that are maximally different. When the populations used by CHC are improved using next ascent bit climbing, CHC also finds the global optimum for JUMP functions as long as the rate of crossover and the population size is sufficient to allow on the order of $Choose(2m, m)$ recombinations of maximally different parents. For $m = 8$, $Choose(16, 8) = 12,870$ recombinations with an associated probability of success of 0.000077, while $2^{16} = 65,536$ recombinations is needed for Uniform crossover, with an associated probability of success of 0.000015.

6 Conclusions

We have shown that a hybrid genetic algorithm which improves the initial population using next ascent bit climbing, and which uses 3 parent Voting Crossover can solve the $Jump_{m,n}$ function when $m = log_2(n)$ with probability asymptotic to 1 for sufficiently large n in $\Theta(n)$ time. We would argue that this is not at all surprising. Theoretically, it is easy to prove that the $Jump_{m,n}$ is also solved in $\Theta(n)$ time by calculating the averages of all of the order-1 hyperplanes. Any combination of crossover and local search that actively exploits this property of the $Jump$ function should also arrive at the global optimum.

It is also the case that the $Jump$ function becomes easier to solve for recombination operators as n increases, because $Choose(n, m)$ becomes larger and the probability that two parents are different enough to have a successful recombination increases. On the other hand, for mutation to make a jump of length m becomes harder as n increases. Assuming that m mutations happen at once, there are still $Choose(n, m)$ ways to select m bits. To make the jump from a local optima, exactly the right m bits must be selected.

The work presented here highlights the advantages of using crossover to enhance exploration. There is no reason that two parents should have only 1 or 2 offspring (as is normally the case for genetic algorithms); there are many biological species where two parents might have dozens of offspring at a time. This idea, while common in the mutation driven $(\mu + \lambda)$ES, has probably not received the attention it deserves in a crossover driven genetic algorithm.

This work also emphasizes the critical role that diversity places in genetic algorithms. We also want to acknowledge the work by Dang et al. [3] looking at how diversity mechanisms, such as fitness sharing, and the use of an Island model, can also make crossover more effective when solving the $Jump$ function when using a $(\mu + 1)$ genetic algorithm. These are all very simple ideas, and common strategies in genetic algorithm applications.

Acknowledgements. This work was supported by a grant from the US National Science Foundation CISE/ACE, SSI-SI2. Dr. Mukhopadhyay was supported by a Fulbright-Nehru Academic and Professional Excellence Fellowship.

References

1. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms-a proof that crossover really can help. Algorithmica **34**, 47–66 (2002)
2. Chicano, F., Whitley, D., Ochoa, G., Tinos, R.: Optimizing one million variable NK landscapes by hybridizing deterministic recombination and local search. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 753–760. ACM (2017)
3. Dang, D., et al.: Escaping local optima with diversity mechanisms and crossover. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 645–652. ACM (2016)
4. Davis, L.: Bit-climbing, representational bias, and test suit design. In: Booker, L., Belew, R. (eds.) International Conference on Genetic Algorithms, pp. 18–23. Springer, Heidelberg (1991)
5. Deb, K., Myburgh, C.: Breaking the billion variable barrier in real world optimization. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 653–660. ACM (2016)
6. Eiben, A.E., Raué, P.-E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) PPSN III 1994. LNCS, vol. 866, pp. 78–87. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58484-6_252
7. Eiben, A.E., Schippers, C.A.: Multi-parent's niche: N-ary crossovers on NK-landscapes. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (eds.) PPSN IV 1996. LNCS, vol. 1141, pp. 319–328. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61723-X_996
8. Eshelman, L.: The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In: Foundations of Genetic Algorithms (FOGA), vol. 1, pp. 265–283. Morgan Kauffman (1991)
9. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math. Program. Comput. **1**(2–3), 119–163 (2009)
10. Helsgaun, K.: DIMACS TSP challenge results: current best tours found by LKH (2013). http://www.akira.ruc.dk/keld/research/LKH/DIMACSresults.html
11. Moscato, P.: Memetic algorithms: a short introduction. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 219–234 (1999)
12. Möbius, A., Freisleben, B., Merz, P., Schreiber, M.: Combinatorial optimization by iterative partial transcription. Phys. Rev. E **59**(4), 4667–4674 (1999)
13. Nagata, Y., Kobayashi, S.: Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem. In: International Conference on Genetic Algorithms (ICGA), pp. 450–457. Morgan Kaufmann (1997)
14. Nagata, Y., Kobayashi, S.: A powerful genetic algorithms using edge assemble crossover the traveling salesman problem. INFORMS J. Comput. **25**(2), 346–363 (2013)
15. Rowe, J.: Population fixed-points for functions of unitation. In: Foundations of Genetic Algorithms (FOGA), vol. 5, pp. 69–84. Morgan Kauffman (1998)
16. Srinivas, M., Patnaik, L.M.: On modeling genetic algorithms for functions of unitation. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **26**(6), 809–821 (1996)
17. Tinós, R., Whitley, D., Chicano, F.: Partition crossover for pseudo-boolean optimization. In: Foundations of Genetic Algorithms, pp. 137–149 (2015)

Fast Artificial Immune Systems

Dogan Corus(✉), Pietro S. Oliveto, and Donya Yazdani

Rigorous Research, University of Sheffield, Sheffield, UK
{d.corus,p.oliveto,dyazdani1}@sheffield.ac.uk

Abstract. Various studies have shown that characteristic Artificial Immune System (AIS) operators such as hypermutations and ageing can be very efficient at escaping local optima of multimodal optimisation problems. However, this efficiency comes at the expense of considerably slower runtimes during the exploitation phase compared to standard evolutionary algorithms. We propose modifications to the traditional 'hypermutations with mutation potential' (HMP) that allow them to be efficient at exploitation as well as maintaining their effective explorative characteristics. Rather than deterministically evaluating fitness after each bit-flip of a hypermutation, we sample the fitness function stochastically with a 'parabolic' distribution which allows the 'stop at first constructive mutation' (FCM) variant of HMP to reduce the linear amount of wasted function evaluations when no improvement is found to a constant. By returning the best sampled solution during the hypermutation, rather than the first constructive mutation, we then turn the extremely inefficient HMP operator without FCM, into a very effective operator for the standard Opt-IA AIS using hypermutation, cloning and ageing. We rigorously prove the effectiveness of the two proposed operators by analysing them on all problems where the performance of HPM is rigorously understood in the literature.

Keywords: Artificial immune systems · Runtime analysis

1 Introduction

Several Artificial Immune Systems (AIS) inspired by Burnet's clonal selection principle [1] have been developed to solve optimisation problems. Amongst these, Clonalg [2], the B-Cell algorithm [3] and Opt-IA [4,5] are the most popular. A common feature of these algorithms is their particularly high mutation rates compared to more traditional evolutionary algorithms (EAs). For instance, the *contiguous somatic hypermutations* (CHM) used by the B-Cell algorithm, choose two random positions in the genotype of a candidate solution and flip all the bits in between[1]. This operation results in a linear number of bits being flipped in an

[1] A parameter may be used to define the probability that each bit in the region actually flips. However, advantages of CHM over EAs have only been shown when all bits in the region flip.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 67–78, 2018.
https://doi.org/10.1007/978-3-319-99259-4_6

average mutation. The *hypermutations with mutation potential* (HMP) used by Opt-IA tend to flip a linear number of bits unless an improving solution is found first (i.e., if no *stop at first constructive mutation* mechanism (FCM) is used, then the operator fails to optimise efficiently any function with a polynomial number of optima [6]).

Various studies have shown how these high mutation rates allow AIS to escape from local optima for which more traditional randomised search heuristics struggle. Jansen and Zarges proved for a benchmark function called Concatenated Leading Ones Blocks (CLOB) an expected runtime of $O(n^2 \log n)$ using CHM versus the exponential time required by EAs relying on standard bit mutations (SBM) since many bits need to be flipped simultaneously to make progress [7]. Similar effects have also been shown on the NP-Hard longest common subsequence [8] and vertex cover [9] standard combinatorial optimisation problems with practical applications where CHM efficiently escapes local optima where EAs (with and without crossover) are trapped for exponential time.

This efficiency on multimodal problems comes at the expense of being considerably slower in the final exploitation phase of the optimisation process when few bits have to be flipped. For instance CHM requires $\Theta(n^2 \log n)$ expected function evaluations to optimise the easy OneMax and LeadingOnes benchmark functions. Indeed it has recently been shown to require at least $\Omega(n^2)$ function evaluations to optimise any function since its expected runtime for its easiest function is $\Theta(n^2)$ [10]. A disadvantage of CHM is that it is *biased*, in the sense that it behaves differently according to the order in which the information is encoded in the bitstring. In this sense the unbiased HMP used by Opt-IA are easier to apply. Also these hypermutations have been proven to be considerably efficient at escaping local optima such as those of the multimodal Jump, Cliff, and Trap benchmark functions that standard EAs find very difficult [6]. This performance also comes at the expense of being slower in the exploitation phase requiring, for instance, $\Theta(n^2 \log n)$ expected fitness evaluations for OneMax and $\Theta(n^3)$ for LeadingOnes.

In this paper we propose a modification to the HMP operator to allow it to be very efficient in the exploitation phases while maintaining its essential characteristics for escaping from local optima. Rather than evaluating the fitness after each bit flip of a hypermutation as the traditional FCM requires, we propose to evaluate it based on the probability that the mutation will be successful. The probability of hitting a specific point at Hamming distance i from the current point, $\binom{n}{i}^{-1}$, decreases exponentially with the Hamming distance for $i < n/2$ and then it increases again in the same fashion. Based on this observation we evaluate each bit following a 'parabolic' distribution such that the probability of evaluating the ith bit flip decreases as i approaches $n/2$ and then increases again. We rigorously prove that the resulting hypermutation operator, which we call P-hype$_{FCM}$, locates local optima asymptotically as fast as Random Local Search (RLS) for any function where the expected runtime of RLS can be proven with the standard artificial fitness levels method. At the same time the operator is still exponentially faster than EAs for the standard multimodal Jump, Cliff, and Trap benchmark functions.

Hypermutations with mutation potential are usually applied in conjunction with ageing operators in the standard Opt-IA AIS. The power of ageing at escaping local optima has recently been enhanced by showing how it makes the difference between polynomial and exponential runtimes for the BALANCE function from dynamic optimisation [11]. For very difficult instances of CLIFF, ageing even makes RLS asymptotically as fast as any unbiased mutation based algorithm can be on any function [12] by running in $O(n \ln n)$ expected time [6]. However, the power of ageing at escaping local optima is lost when it is used in combination with hypermutations with mutation potential. In particular, the FCM mechanism does not allow the operator to accept solutions of lower quality, thus cancelling the advantages of ageing. Furthermore, the high mutation rates combined with FCM make the algorithm return to the previous local optimum with very high probability. While the latter problem is naturally solved by our newly proposed P-hype$_{FCM}$ that does not evaluate all bit flips in a hypermutation, the former problem requires a further modification to the HMP. The simple modification that we propose is for the operator, which we call P-hype$_{BM}$, to return the best solution it has found if no constructive mutation is encountered. We rigorously prove that Opt-IA then benefits from both operators for all problems where it was previously analysed in the literature, as desired. Due to space limitations some proofs are omitted for this extended abstract[2].

2 Preliminaries

Static hypermutations with mutation potential using FCM (i.e., stop at the first constructive mutation) mutate $M = cn$ distinct bits for a constant $0 < c \leq 1$ and evaluate the fitness after each bit-flip [6]. If an improvement over the original solution is found before the Mth bit-flip, then the operator stops and returns the improved solution. This behaviour prevents the hypermutation operator to waste further fitness function evaluations if an improvement has already been found. However, for any realistic objective function the number of iterations where there is an improvement constitutes an asymptotically small fraction of the total runtime. Hence, the fitness function evaluations saved due to the FCM stopping the hypermutation have a very small impact on the global performance of the algorithm. Our proposed modified hypermutation operator, called P-hype, instead only evaluates the fitness after each bit-flip with a probability that depends on how many bits have already been flipped in the current hypermutation operation. Since previous theoretical analyses have considered $c = 1$ (i.e., $M = n$) [6], we also use this value throughout this paper. Let p_i be the probability that the solution is evaluated after the ith bit has been flipped. The 'parabolic' probability distribution is defined as follows, where the parameter γ should be between $0 < \gamma \leq 2$ (Fig. 1):

[2] A complete version of the paper including all the proofs is available on arXiv [13].

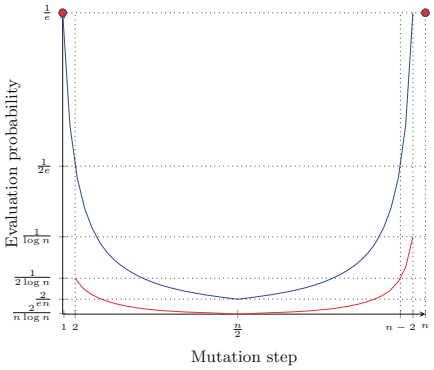


Fig. 1. The parabolic evaluation probabilities (1) for $\gamma = 1/\log n$ and $\gamma = 1/e$.

$$p_i = \begin{cases} 1/e & \text{for } i = 1 \text{ and } i = n \\ \gamma/i & \text{for } 1 < i \leq n/2 \\ \gamma/(n-i) & \text{for } n/2 < i < n \end{cases} \tag{1}$$

The lower the value of γ, the fewer the expected fitness function evaluations that occur in each hypermutation. On the other hand, with a small enough parameter γ value, the number of wasted evaluations can be dropped to the order of $O(1)$ per iteration instead of the linear amount wasted by the traditional operator when improvements are not found. The resulting hypermutation operator is formally defined as follows.

Definition 1 (P-hype$_{FCM}$). *P-hype$_{FCM}$ flips at most n distinct bits selected uniformly at random. It evaluates the fitness after the ith bit-flip with probability p_i (as defined in (1)) and remembers the last evaluation. P-hype$_{FCM}$ stops flipping bits when it finds an improvement; if no improvement is found, it will return the last evaluated solution. If no evaluations are made, the parent will be returned.*

In the next section we will prove its benefits over the standard static HMP with FCM, when incorporated into a $(1+1)$ framework (Algorithm 1). However, in order for the operator to work effectively in conjunction with ageing, a further modification is required. Instead of stopping the hypermutation at the first constructive mutation, we will execute all n mutation steps, evaluate each bitstring with the probabilities in (1) and as the offspring, return the best solution evaluated during the hypermutation or the parent itself if no other bitstrings are evaluated. We will prove that such a modification, which we call P-hype$_{BM}$, may allow the complete Opt-IA to escape local optima more efficiently by P-hype$_{BM}$ producing solutions of lower quality than the local optimum on which the algorithm was stuck while individuals on the local optimum die due to ageing. P-hype$_{BM}$ is formally defined as follows.

Algorithm 1. $(1+1)$ Fast-IA

1: Initialise x uniformly at random.
2: **while** a global optimum is not found **do**
3: Create $y = x$, then $y =$ P-hype(y);
4: If $f(y) \geq f(x)$, then $x = y$.
5: **end while**

Definition 2 (P-hype$_{BM}$). *P-hype$_{BM}$ flips n distinct bits selected uniformly at random. It evaluates the fitness after the ith bit-flip with probability p_i (as defined in (1)) and remembers the best evaluation found so far. P-hype$_{BM}$ returns the mutated solution with the best evaluation found. If no evaluations are made, the parent will be returned.*

For sufficiently small values of the parameter γ only one function evaluation per hypermutation is performed in expectation (although all bits will be flipped). Since it returns the best found one, this solution will be returned by P-hype$_{BM}$ as it is the only one it has encountered. Interestingly, this behaviour is similar to that of the HMP without FCM that also evaluates one point per hypermutation and returns it. However, while HMP without FCM has exponential expected runtime for any function with a polynomial number of optima [6], we will show in the following sections that P-hype$_{BM}$ can be very efficient. From this point of view, P-hype$_{BM}$ is as a very effective way to perform hypermutations with mutation potential without FCM.

In Sect. 4, we consider P-hype$_{BM}$ in the complete Opt-IA framework [4–6] hence analyse its performance combined with cloning and ageing. The algorithm which we call Fast Opt-IA, is depicted in Algorithm 2. We will use the *hybrid ageing* operator as in [6,11], which allows us to escape local optima. Hybrid ageing removes candidate solutions (i.e. b-cells) with probability $p_{die} = 1 - (1/(\mu+1))$ once they have passed an age threshold τ. After initialising a population of μ b-cells with $age = 0$, at each iteration the algorithm creates dup copies of each b-cell. These copies are mutated by the P-hype operator, creating a population of mutants called P^{hyp} which inherit the age of their parents if they do not improve the fitness; otherwise their age will be set to zero. At the next step, all b-cells with $age \geq \tau$ will be removed from both populations with probability p_{die}. If less than μ individuals have survived ageing, then the population is filled up with new randomly generated individuals. At the selection phase, the best μ b-cells are chosen to form the population for the next generation.

3 Fast Hypermutations

We start our analysis by relating the expected number of fitness function evaluations to the expected number of P-hype operations until the optimum is found. The following result holds for both P-hype operators. The lemma quantifies the number of expected fitness function evaluations which are wasted by a hypermutation operation.

Algorithm 2. Fast Opt-IA

Initialise a population of μ b-cells, P, created uniformly at random;
for each $x \in P$ set $x^{age} = 0$.
while a global optimum is not found **do**
  for each $x \in P$ set $x^{age} = x^{age} + 1$;
  for *dup* times for each $x \in P$ **do**
    $y =$ P-hype(x);
    if $f(y) > f(x)$ **then** $y^{age} = 0$ **else** $y^{age} = x^{age}$;
    Add y to P^{hyp}.
  end for
  Add P^{hyp} to P, set $P^{hyp} = \emptyset$;
  for each $x \in P$ **if** $x^{age} \geq \tau$ **then** remove x with probability p_{die};
  if $|P| < \mu$ **then** add $\mu - |P|$ solutions to P with age zero generated uniformly at random;
  if $|P| > \mu$ **then** remove $|P| - \mu$ solutions with the lowest fitness from P breaking ties uniformly at random.
end while

Lemma 1. *Let T be the random variable denoting the number of P-hype operations applied until the optimum is found. Then, the expected number of total function evaluations is at most: $E[T] \cdot O(1 + \gamma \log n)$.*

Proof. Let the random variable X_i for $i \in [T]$ denote the number of fitness function evaluations during the ith execution of P-hype. Additionally, let the random variable X'_i denote the number of fitness function evaluations at the ith operation assuming that no improvements are found. For all i it holds that $X_i \preceq X'_i$ since finding an improvement can only decrease the number of evaluations. Thus, the total number of function evaluations $E[\sum_{i=1}^{T} X_i]$ can be bounded above by $E[\sum_{i=1}^{T} X'_i]$ which is equal to $E[T] \cdot E[X']$ due to Wald's equation [14] since X'_i are identically distributed and independent from T.

We now write the expected number of fitness function evaluations in each operation as the sum of n indicator variables Y_i for $i \in [n]$ denoting whether an evaluation occurs after the ith bit mutation. Referring to the probabilities in (1), we get, $E[X] = E\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} Pr\{Y_i = 1\} = \frac{1}{e} + \frac{1}{e} + 2\sum_{i=2}^{n/2} \gamma \frac{1}{i} \leq \frac{2}{e} + 2\gamma(\ln n/2 - 1)$. □

In Lemma 1, γ appears as a multiplicative factor in the expected runtime measured in fitness function evaluations. An intuitive lower bound of $\Omega(1/\log n)$ for γ can be inferred since smaller mutation rates will not decrease the runtime. While a smaller γ does not decrease the asymptotic order of expected evaluations per operation, in Sect. 4 we will provide an example where a smaller choice of γ reduces $E[T]$ directly. For the rest of our results though, we will rely on $E[T]$ being the same as for the traditional static hypermutations with FCM while the number of wasted fitness function evaluations decreases from n to $O(1+\gamma \log n)$.

Table 1. Expected runtimes of the standard $(1+1)$ EA and $(1+1)$ IA$^{\text{hyp}}$ versus the expected runtime of the $(1+1)$ Fast-IA. For $\gamma = O(1/\log n)$, the $(1+1)$ Fast-IA is asymptotically at least as fast as the $(1+1)$ EA and faster by a linear factor compared to the $(1+1)$ IA$^{\text{hyp}}$ for the unimodal and trap functions. For not too large jump and cliff sizes (i.e., $o(n/\log n)$), the $(1+1)$ Fast-IA has an asymptotic speed up compared to the $(1+1)$ IA$^{\text{hyp}}$ for the same parameter setting. For not too small jump and cliff sizes both AISs are much faster than the $(1+1)$ EA.

Function	$(1+1)$ EA	$(1+1)$ IA$^{\text{hyp}}$	$(1+1)$ Fast-IA
OneMax	$\Theta(n \log n)$ [15]	$\Theta(n^2 \log n)$ [6]	$\Theta\left(n \log n \left(1+\gamma \log n\right)\right)$
LeadingOnes	$\Theta(n^2)$ [15]	$\Theta(n^3)$ [6]	$\Theta\left(n^2 \left(1+\gamma \log n\right)\right)$
Trap	$\Theta(n^n)$ [15]	$\Theta(n^2 \log n)$ [6]	$\Theta\left(n \log n \left(1+\gamma \log n\right)\right)$
$\text{Jump}_{d>1}$	$\Theta(n^d)$ [15]	$O(n\binom{n}{d})$ [6]	$O\left((d/\gamma) \cdot (1+\gamma \log n) \cdot \binom{n}{d}\right)$
$\text{Cliff}_{d>1}$	$\Theta(n^d)$ [16]	$O(n\binom{n}{d})$ [6]	$O\left((d/\gamma) \cdot (1+\gamma \log n) \cdot \binom{n}{d}\right)$

We will now analyse the simplest setting where we can implement P-hype. The $(1+1)$ Fast-IA keeps a single individual in the population and uses P-hype to perturb it at every iteration. The performance of the $(1+1)$ IA$^{\text{hyp}}$, a similar barebones algorithm using the classical static hypermutation operator has recently been related to the performance of the well-studied Randomised Local Search algorithm (RLS) [6]. RLS_k flips exactly k bits of the current solution to sample a new search point, compares it with the current solution and continues with the new one unless it is worse. According to Theorems 3.3 and 3.4 of [6], any runtime upper bound for RLS obtained via Artificial Fitness Levels (AFL) method also holds for the $(1+1)$ IA$^{\text{hyp}}$ with an additional factor of n (e.g., an upper bound of $O(n)$ for RLS derived via AFL translates into an upper bound of $O(n^2)$ for the $(1+1)$ IA$^{\text{hyp}}$). The following theorem establishes a similar relationship between RLS and the $(1+1)$ Fast-IA with a factor of $O(1+\gamma \log n)$ instead of n. In the context of the following theorem, $(1+1)$ Fast-IA$_{\geq}$ denotes the variant of $(1+1)$ Fast-IA which considers an equally good solution as constructive while $(1+1)$ Fast-IA$_{>}$ stops the hypermutation only if a solution strictly better than the parent is sampled.

Theorem 1. *Let $E\left(T_A^{AFL}\right)$ be any upper bound on the expected runtime of algorithm A established by the artificial fitness levels method. Then $E\left(T^{AFL}_{(1+1)\ Fast\text{-}IA_{>}}\right) \leq E\left(T^{AFL}_{(1+1)\ RLS_k}\right) \cdot k/\gamma \cdot O(1+\gamma \log n)$. Moreover, for the special case of $k=1$, $E\left(T^{AFL}_{(1+1)\ Fast\text{-}IA_{\geq}}\right) \leq E\left(T^{AFL}_{(1+1)\ RLS_1}\right) \cdot O(1+\gamma \log n)$ also holds.*

Apart from showing the efficiency of the $(1+1)$ Fast-IA, the theorem also allows easy achievements of upper bounds on the runtime of the algorithm, by just analysing the simple RLS. For $\gamma = O(1/\log n)$, Theorem 1 implies the upper bounds of $O(n \log n)$ and $O(n^2)$ for classical benchmark functions OneMax and LeadingOnes respectively (see Table 1). Both of these bounds are

asymptotically tight since each function's unary unbiased black-box complexity is in the same order as the presented upper bound [12].

Corollary 1. *The expected runtimes of the (1 + 1) Fast-IA to optimise* OneMax$(x) := \sum_{i=1}^{n} x_i$ *and* LeadingOnes $:= \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$ *are respectively* $O\left(n \log n \left(1 + \gamma \log n\right)\right)$ *and* $O(n^2 \left(1 + \gamma \log n\right))$. *For* $\gamma = O(1/\log n)$ *these bounds reduce to* $\Theta(n \log n)$ *and* $\Theta(n^2)$.

P-hype samples the complementary bit-string with probability one if it cannot find any improvements. This behaviour allows an efficient optimisation of the deceptive Trap function which is identical to OneMax except that the optimum is in 0^n. Since n bits have to be flipped to reach the global optimum from the local optimum, EAs based on SBM require exponential runtime with overwhelming probability [17]. By evaluating the sampled bitstrings stochastically, the $(1+1)$ Fast-IA provides up to a linear speed-up for small enough γ compared to the $(1+1)$ IA$^{\text{hyp}}$ on Trap as well.

Theorem 2. *The expected runtime of the (1 + 1) Fast-IA to optimise* Trap *is* $\Theta(n \log n \left(1 + \gamma \log n\right))$.

The results for the $(1+1)$ IA$^{\text{hyp}}$ on Jump and Cliff functions [6] can also be adapted to the $(1+1)$ Fast-IA in a straightforward manner, even though they fall out of the scope of Theorem 1. Both Jump$_d$ and Cliff$_d$ have the same output as OneMax for bitstrings with up to $n - d$ 1-bits and the same optimum 1^n. For solutions with the number of 1-bits between $n - d$ and n, Jump has a reversed OneMax slope creating a gradient towards $n - d$ while Cliff has a slope heading toward 1^n even though the fitness values are penalised by an additive factor d. Being designed to accomplish larger mutations, the performance of hypermutations on Jump and Cliff functions is superior to standard bit mutation [6]. This advantage is preserved for the $(1+1)$ Fast-IA as seen in the following theorem.

Theorem 3. *The expected runtime of the (1 + 1) Fast-IA to optimise* Jump$_d$ *and* Cliff$_d$ *is* $O\left((d/\gamma) \cdot (1 + \gamma \log n) \cdot \binom{n}{d}\right)$.

For Jump and Cliff, the superiority of the $(1+1)$ Fast-IA in comparison to the deterministic evaluations scheme depends on the function parameter d. If $\gamma = \Omega(1/\log n)$, the $(1+1)$ Fast-IA performs better for $d = o(n/\log n)$ while the deterministic scheme (i.e., $(1+1)$ IA$^{\text{hyp}}$) is preferable for larger d. However, for small d the difference between the runtimes can be as large as a factor of n in favor of the $(1+1)$ Fast-IA while, even for the largest d, the difference is less than a factor of $\log n$ in favor of the deterministic scheme. Here we should also note that for $d = \Omega(n/\log n)$ the expected time is exponentially large for both algorithms (albeit considerably smaller than that of standard EAs) and the $\log n$ factor has no realistic effect on the applicability of the algorithm.

4 Fast Opt-IA

In this section we will consider the effect of our proposed evaluation scheme on the complete Opt-IA algorithm. The distinguishing characteristic of the Opt-IA algorithm is its use of the ageing and hypermutation operators. In [6] a fitness function called HiddenPath (Fig. 2) was presented where the use of both operators is necessary to find the optimum in polynomial time. The function HiddenPath provides a gradient to a local optimum, which allows the hypermutation operator to find another gradient which leads to the global optimum but situated on the opposite side of the search space (i.e., nearby the complementary bitstrings of the local optima). However, the ageing operator is necessary for the algorithm to accept a worsening; otherwise the second gradient is not accessible. To prove our upper bound, we can follow the same proof strategy in [18], which established an upper bound of $O(\tau\mu n + \mu n^{7/2})$ for the expected runtime of the traditional Opt-IA on HiddenPath. We will see that Opt-IA benefits from an $n/\log n$ speed-up due to P-hype.

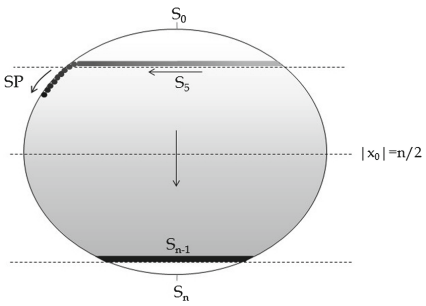


Fig. 2. HiddenPath [6]

Theorem 4. *The Fast Opt-IA needs* $O(\tau\mu + \mu n^{5/2} \log n)$ *fitness function evaluations in expectation to optimise* HiddenPath *with* $\mu = O(\log n)$, *dup* $= 1$, $1/(4 \ln n) \geq \gamma = \Omega(1/\log n)$ *and* $\tau = \Omega(n \log^2 n)$.

HiddenPath was artificially constructed to fit the behaviour of the Opt-IA to illustrate its strengths. One of those strengths was the ageing mechanism's ability to escape local optima in two different ways. First, it allows the algorithm to restart with a new random population after it gets stuck at a local optimum. Second, ageing allows individuals with worse fitness than the current best to stay in the population when all the current best individuals are removed by the ageing operator in the same iteration. If an improvement is found soon after the worsening is accepted, then this temporary non-elitist behaviour allows the algorithm to follow other gradients which are accessible by variation from the local optima but leads away from them. On the other hand, even though it is coupled with ageing in the Opt-IA, the FCM mechanism does not allow worsenings.

More precisely, for the hypermutation with FCM, the complementary bit-string of the local optimum is sampled with probability 1 if no other improvements are found. Indeed, HiddenPath was designed to exploit this high probability. However, by only stopping on improving mutations, the traditional hypermutations with FCM do not allow, in general, to take advantage of the power of ageing at escaping local optima. For instance, for the classical benchmark function $\textsc{Cliff}_d$ with parameter $d = \Theta(n)$, hypermutation with FCM turned out to be a worse choice of variation operator to couple with ageing than both local search and standard-bit-mutation [18]. Ageing coupled with RLS and SBM can reach the optimum by local moves, which respectively yields upper bounds of $O(n \log n)$ and $O(n^{1+\epsilon} \log n)$ for arbitrarily small positive constant ϵ on their runtimes. However, hypermutations with FCM require to increase the number of 1-bits in the current solution by d at least once before the hypermutation stops. This requirement implies the following exponential lower bound on the runtime regardless of the evaluation scheme (as long as the hypermutation only stops on a constructive mutation).

Theorem 5. *Fast Opt-IA using P-hype$_{FCM}$ requires at least $2^{\Omega(n)}$ fitness function evaluations in expectation to find the optimum of* $\textsc{Cliff}_d$ *for $d = (1-c)n/4$, where c is a constant $1 > c > 0$.*

The following theorem will demonstrate how P-hype$_{FCM}$, that, instead of stopping the hypermutation at the first constructive mutation, will execute all n mutation steps, evaluate each bitstring with the probabilities in (1) and return the best found solution, allows ageing and hypermutation to work in harmony in Opt-IA.

Theorem 6. *Fast Opt-IA using P-hype$_{BM}$ with $\mu = 1$, dup $= 1$, $\gamma = 1/(n \log^2 n)$ and $\tau = \Theta(n \log n)$ needs $O(n \log n)$ fitness function evaluations in expectation to optimise* Cliff *with any linear $d \leq n/4 - \epsilon$ for an small constant ϵ.*

Note that the above result requires a γ in the order of $\Theta(1/(n \log^2 n))$, while Lemma 1 implies that any $\gamma = \omega(1/\log n)$ would not decrease the expected number of fitness function evaluations below the asymptotic order of $\Theta(1)$. However, having $\gamma = 1/(n \log^2 n)$ allows Opt-IA, with constant probability, to complete its local search before any solution with larger Hamming distance is ever evaluated. In Theorem 6, we observe that this opportunity allows the Opt-IA to hillclimb the second slope before jumping back to the local optima. The following theorem rigorously proves that a very small choice for γ in this case is necessary (i.e., $\gamma = \Omega(1/\log n)$ leads to exponential expected runtime).

Theorem 7. *At least $2^{\Omega(n)}$ fitness function evaluations in expectation are executed before the Fast Opt-IA using P-hype$_{BM}$ with $\gamma = \Omega(1/\log n)$ finds the optimum of* $\textsc{Cliff}_d$ *for $d = (1-c)n/4$, where c is a constant $1 > c > 0$.*

5 Conclusion

Due to recent analyses of increasingly realistic evolutionary algorithms, higher mutation rates, naturally present in artificial immune systems, than previously

recommended or used as a rule of thumb, are gaining significant interest in the evolutionary computation community [19–22].

We have presented two alternative 'hypermutations with mutation potential' operators, P-hype$_{FCM}$ and P-hype$_{BM}$ and have rigorously proved, for several significant benchmark problems from the literature, that they maintain the exploration characteristics of the traditional operators while outperforming them up to linear factor speed-ups in the exploitation phase.

The main modification that allows to achieve the presented improvements is to sample the solution after the ith bit-flip stochastically with probability roughly $p_i = \gamma/i$, rather than deterministically with probability one. The analysis shows that the parameter γ can be set easily. Concerning P-hype$_{FCM}$, that returns the first sampled constructive mutation and is suggested to be used in isolation, any $\gamma = O(1/\log(n))$ allows optimal asymptotical exploitation time (based on the unary unbiased black box complexity of OneMax and LeadingOnes) while maintaining the traditional exploration capabilities. Concerning P-hype$_{BM}$, which does not use FCM and is designed to work harmonically with ageing as in the standard Opt-IA, considerably lower values of the parameter (i.e., $\gamma = 1/(n \log^2 n)$) are required to escape from difficult local optima efficiently (e.g., Cliff) such that the hypermutations do not return to the local optima with high probability. While these low values for γ still allow optimal asymptotic exploitation in the unbiased unary black box sense, they considerably reduce the capability of the operator to perform the large jumps required to escape the local optima of functions with characteristics similar to Jump, i.e., where ageing is ineffective due to the second slope of decreasing fitness. Future work may consider an adaptation of the parameter γ to allow it to automatically increase and decrease throughout the run [23,24]. Furthermore, the performance of the proposed operators should be evaluated for classical combinatorial optimisation problems and real-world applications.

References

1. Burnet, F.M.: The Clonal Selection Theory of Acquired Immunity. Cambridge University Press, Cambridge (1959)
2. de Castro, L.N., Von Zuben, F.J.: Learning and optimization using the clonal selection principle. IEEE Trans. Evol. Comput. **6**(3), 239–251 (2002)
3. Kelsey, J., Timmis, J.: Immune inspired somatic contiguous hypermutation for function optimisation. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 207–218. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_26
4. Cutello, V., Nicosia, G., Pavone, M., Timmis, J.: An immune algorithm for protein structure prediction on lattice models. IEEE Trans. Evol. Comput. **11**(1), 101–117 (2007)
5. Cutello, V., Nicosia, G., Pavone, M.: A hybrid immune algorithm with information gain for the graph coloring problem. In: Cantú-Paz, E. (ed.) GECCO 2003. LNCS, vol. 2723, pp. 171–182. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_23

6. Corus, D., Oliveto, P.S., Yazdani, D.: On the runtime analysis of the Opt-IA artificial immune system. In: Proceedings of the GECCO 2017, pp. 83–90 (2017)
7. Jansen, T., Zarges, C.: Analyzing different variants of immune inspired somatic contiguous hypermutations. Theor. Comput. Sci. **412**(6), 517–533 (2011)
8. Jansen, T., Zarges, C.: Computing longest common subsequences with the B-Cell algorithm. In: Coello Coello, C.A., Greensmith, J., Krasnogor, N., Liò, P., Nicosia, G., Pavone, M. (eds.) ICARIS 2012. LNCS, vol. 7597, pp. 111–124. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33757-4_9
9. Jansen, T., Oliveto, P.S., Zarges, C.: On the analysis of the immune-inspired B-Cell algorithm for the vertex cover problem. In: Liò, P., Nicosia, G., Stibor, T. (eds.) ICARIS 2011. LNCS, vol. 6825, pp. 117–131. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22371-6_13
10. Corus, D., He, J., Jansen, T., Oliveto, P.S., Sudholt, D., Zarges, C.: On easiest functions for mutation operators in bio-inspired optimisation. Algorithmica **78**(2), 714–740 (2016)
11. Oliveto, P.S., Sudholt, D.: On the runtime analysis of stochastic ageing mechanisms. In: Proceedings of the GECCO 2014, pp. 113–120 (2014)
12. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**(4), 623–642 (2012)
13. Corus, D., Oliveto, P.S., Yazdani, D.: Fast artificial immune systems. ArXiv e-prints (2018). http://arxiv.org/abs/1806.00299
14. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
15. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1 + 1) evolutionary algorithm. Theor. Comput. Sci. **276**(1–2), 51–81 (2002)
16. Paixão, T., Heredia, J.P., Sudholt, D., Trubenová, B.: Towards a runtime comparison of natural and artificial evolution. Algorithmica **78**(2), 681–713 (2017)
17. Oliveto, P.S., Yao, X.: Runtime analysis of evolutionary algorithms for discrete optimization. In: Auger, A., Doerr, B. (eds.) Theory of Randomized Search Heuristics, pp. 21–52. World Scientific (2011)
18. Corus, D., Oliveto, P.S., Yazdani, D.: When hypermutations and ageing enable artificial immune systems to outperform evolutionary algorithms. ArXiv e-prints (2018). http://arxiv.org/abs/1804.01314
19. Oliveto, P.S., Lehre, P.K., Neumann, F.: Theoretical analysis of rank-based mutation-combining exploration and exploitation. In: Proceedings of the CEC 2009, pp. 1455–1462 (2009)
20. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proceedings of the GECCO 2017, pp. 777–784 (2017)
21. Corus, D., Oliveto, P.S.: Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Trans. Evol. Comput. (2017)
22. Dang, D.-C., et al.: Emergence of diversity and its benefits for crossover in genetic algorithms. IEEE Trans. Evol. Comput. (2017, to appear)
23. Doerr, B., Lissovoi, A., Oliveto, P.S., Warwicker, J.A.: On the runtime analysis of selection hyper-heuristics with adaptive learning periods. In: Proceedings of the GECCO 2018. ACM (2018, to appear)
24. Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. Algorithmica **80**, 1658–1709 (2018)

First-Hitting Times for Finite State Spaces

Timo Kötzing and Martin S. Krejca(✉)

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
martin@krejca.de

Abstract. One of the most important aspects of a randomized algorithm is bounding its expected run time on various problems. Formally speaking, this means bounding the expected first-hitting time of a random process. The two arguably most popular tools to do so are the *fitness level method* and *drift theory*. The fitness level method considers arbitrary transition probabilities but only allows the process to move toward the goal. On the other hand, drift theory allows the process to move into any direction as long as it move closer to the goal in expectation; however, this tendency has to be monotone and, thus, the transition probabilities cannot be arbitrary.

We provide a result that combines the benefit of these two approaches: our result gives a lower and an upper bound for the expected first-hitting time of a random process over $\{0, \dots, n\}$ that is allowed to move forward and backward by 1 and can use arbitrary transition probabilities. In case that the transition probabilities are known, our bounds coincide and yield the exact value of the expected first-hitting time. Further, we also state the stationary distribution as well as the mixing time of a special case of our scenario.

1 Introduction

A very important part of recent research on the theoretical analysis of evolutionary algorithms (EAs) is concerned with run time analysis, and over the years, different tools have been proposed in order to derive run time results more easily. The approaches used for run time analysis all follow the same very broad outline: the algorithm is viewed as a random process whose progress over time is measured. The aim is to bound the expected first-hitting time of the process reaching a certain state, usually finding an optimum. Depending on how much progress can be achieved in different phases of the algorithm, a bound on the expected first-hitting time – the run time – can then be derived. The approaches differ in what phases they consider and how restricted the random process needs to be. We discuss the two arguably most well-known approaches: the *fitness level method* and *drift theory*.

The **fitness level method** is historically older than drift and was first formally defined by Wegener [13] in the context of EAs; a nice overview of this tool including tail bounds was provided by Witt [14]. The method considers a

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 79–91, 2018.
https://doi.org/10.1007/978-3-319-99259-4_7

partition of the optimization domain into levels. The progress of an EA is then measured by the expected time it takes to get from one level to another. This means that the expected first-hitting time can be bounded by the sum of the waiting times per level. The major drawback of this tool is that it assumes that there are no cycles among the levels, that is, once the EA advances to a next level, it cannot return to any older level. Hence, this approach basically bounds the first-hitting time of a random walk on a directed path. However, the limitation of having no cycles among the levels results in a very concise theorem that is able to yield *exact* bounds when the actual transition probabilities are known.

Dang and Lehre [1] provide theorems similar to the fitness level method but allow for cycles among the levels. While this approach can yield good upper bounds easily, especially for non-elitist EAs, it assumes that the algorithm of interest makes use of a population. Thus, the theorems cannot be applied to all random processes. Further, without a lower bound, it is not clear how tight a result actually is.

Drift theory is an entirely different approach to deriving expected first-hitting times; see the informative article of Lengler [9] for a general introduction to this topic. Different from the fitness level method, drift theory does not estimate the progress of a random process via waiting times in different levels but instead looks at the expected change of the process after a single step – the *drift*. In this setting, arbitrary steps closer to the goal or away from it may be permitted. The expected first-hitting time then follows from a bound on the drift of the process. Similar to the fitness level method, drift theory can provide upper and lower bounds that are exact if the actual transition probabilities are known.

In its most restrictive setting – the *additive drift theorem* (see Theorem 1), the bound on the drift has to be the same for all states of the random process considered. This means that the bounds on the transition probabilities have to be the same, which limits applicability. In a case where this is overly confining, more advanced theorems like the *variable drift theorem* (see Theorem 2) can be used, which allows to bound the drift dependent on the current state of the process. However, all of these theorems have in common that the the drift needs to be bounded in a monotone way. This means that the drift has to decrease as the goal is approached – a restriction that the fitness level method does not have.

In this paper, we combine the benefits of the fitness level method and of drift theory. Our main result, Theorem 3, considers a random process that is allowed to move toward the goal or away from it in any (not necessarily monotone) fashion. Our setting assumes, in its simplest form, a random walk on an undirected path with the nodes 0 through n. For this setting, we get the exact expected first-hitting time. Our result also provides upper bounds when the process makes larger steps toward the goal and lower bounds when it makes larger steps away from it. We show that our result is a generalization of the fitness level method (Corollary 6) and that it yields bounds that cannot be derived with the variable drift theorem (see Example 4) – the most general drift theorem available. Hence, our result sheds new light on the behavior of random processes over finite state spaces when the progress of the process is not monotone or if the drift is 0.

Further, we also analyze our setting in the context of Markov chains. We give the stationary distribution when the process is a random walk on a path (Theorem 7), again, allowing for arbitrary transition probabilities, and we state an upper bound on the mixing time of the process (Corollary 9). This allows to estimate the probability of the process being in a certain state at a specific point in time – a concept strongly connected to the *any-time analysis* introduced by Jansen and Zarges [6] and similar to *occupation probabilities* as discussed by Lissovoi and Witt [11] and Kötzing et al. [8].

Our paper is structured as follows: Sect. 2 introduces the setting we consider as well as the tools we need in order to derive our results. Section 3 contains our main result, Theorem 3, as well as examples of how the bounds following from it cannot be achieved via any known drift theorem. Last, in Sect. 4, we consider the stationary distribution and the mixing time of the processes we consider.

Note that a special case of Theorem 3 has already been proven by Droste et al. [2] when the Markov chain is a path. Our result extends theirs by providing an upper and a lower bound for scenarios where more transition probabilities are allowed. Further, our result is proven using drift theory, a modern tool that was not available to Droste et al. back then.

2 Setting

We consider random processes $(X_t)_{t\in\mathbb{N}}$ over the finite set $\{0,\dots,n\}$, for an $n \in \mathbb{N}$. In its simplest form, the process is only allowed to move from state s to $s-1$, s, and $s+1$ (if they exist). However, our main result (Theorem 3) generalizes to settings where the process can additionally either make arbitrary long jumps to the front (that is, from state s to any state $s' < s$) or to the back. Our process can be thought of as a random walk as seen in Fig. 1. In the most-restricted scenario, where the process can only move to neighboring states, it performs a random walk on a path.

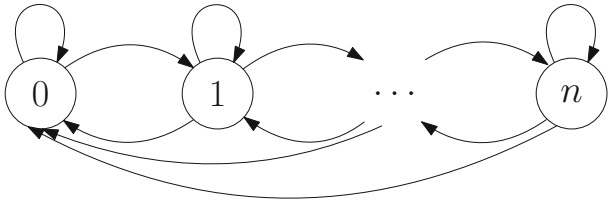


Fig. 1. An exemplary setting we consider. Each node represents a state, and each edge represents a possible transition. Here, the process can move from a state s to any state $s' < s$ when moving to the left but can only move to state $s+1$ when moving to the right.

We are interested in the expected first-hitting time of such a process reaching the state 0. More formally, let $T = \min\{t \mid X_t = 0\}$; we want to bound $\mathrm{E}[T]$.

In our results, we give bounds for $\mathrm{E}[T \mid X_0]$ instead, which is a random variable, as X_0 is a random variable. Bounds on $\mathrm{E}[T]$ can then be derived by the law of total expectation, that is, $\mathrm{E}[T] = \mathrm{E}\big[\mathrm{E}[T \mid X_0]\big]$.

In order to be able to actually reach our goal state 0, we assume that the probability of the random process X_t to move left is positive for any state $s > 0$, that is, $\Pr[X_t - X_{t+1} \geq 1 | X_t = s] > 0$.

2.1 Stochastic Tools

Our results make use of drift theory – a tool that allows to estimate the first-hitting time of a random process when given only estimates of local changes of that process.

The main theorem we use is the following additive drift theorem by He and Yao [3,4]. It yields bounds on the first-hitting time of a random process reaching 0 when the expected local change – the *drift* – can be bounded by a value independent of the current state. We use this theorem in order to prove our main result.

Theorem 1 (Additive Drift [3,5]). *Let $(X_t)_{t\in\mathbb{N}}$ be nonnegative random variables over a finite space $S \subset \mathbb{R}_{\geq 0}$ containing 0, and let $T = \min\{t \mid X_t = 0\}$.*

If there is a constant $\delta > 0$ such that, for all $s \in S$ and all $t < T$,

$$\mathrm{E}[X_t - X_{t+1} \mid X_t = s] \geq \delta, \text{ then } \mathrm{E}[T \mid X_0] \leq \frac{X_0}{\delta}.$$

And if there is a $\delta > 0$ such that, for all $s \in S$ and all $t < T$,

$$\mathrm{E}[X_t - X_{t+1} \mid X_t = s] \leq \delta, \text{ then } \mathrm{E}[T \mid X_0] \geq \frac{X_0}{\delta}.$$

A more flexible drift theorem is the following variable drift theorem. It allows to upper-bound the expected first-hitting time of a random process reaching 0 when the drift can depend in any monotone fashion on the current state. We use this theorem to compare our main result against.

Theorem 2 (Variable Drift [7,12]). *Let $(X_t)_{t\in\mathbb{N}}$ be nonnegative random variables over $\{0\} \cup S$, where $S \subset \mathbb{R}_{\geq 1}$ is a finite state space containing 1, and let $T = \min\{t \mid X_t < 1\}$.*

If there exists a monotonically increasing function $h\colon \mathbb{R}^+ \to \mathbb{R}_{\geq 0}$ such that $1/h$ is integrable and, for all $s \in S$ and all $t < T$,

$$\mathrm{E}[X_t - X_{t+1} \mid X_t = s] \geq h(s), \text{ then } \mathrm{E}[T \mid X_0] \leq \frac{1}{h(1)} + \int_1^{X_0} \frac{1}{h(x)}\,\mathrm{d}x.$$

3 General First-Hitting Times

We start by stating and discussing our main result, Theorem 3, which provides an upper and a lower bound of the first-hitting time of a random process in the setting described in Sect. 2. Those bounds make use of bounds on the transition probabilities of the process.

Theorem 3. *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\{0,\dots,n\}$ and let T denote the first point in time t such that $X_t = 0$.*

1. *Suppose there are two functions $p^{\leftarrow}\colon \{1,\dots,n\} \to [0,1]$ and $p^{\rightarrow}\colon \{0,\dots,n-1\} \to [0,1]$ such that, for all $t < T$ and all $s \in \{1,\dots,n\}$,*
   - $p^{\leftarrow}(s) > 0$,
   - $\Pr[X_t - X_{t+1} \geq 1 | X_t = s] \geq p^{\leftarrow}(s)$,
   - $\Pr[X_t - X_{t+1} = -1 | X_t = s] \leq p^{\rightarrow}(s)$ *(for $s \neq n$), and*
   - $\Pr[X_t - X_{t+1} < -1 | X_t = s] = 0$ *(for $s \neq n$).*

   Then
$$\mathrm{E}[T \mid X_0] \leq \sum_{s=1}^{X_0} \sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)}.$$

2. *Suppose there are two functions $p^{\leftarrow}\colon \{1,\dots,n\} \to [0,1]$ and $p^{\rightarrow}\colon \{0,\dots,n-1\} \to [0,1]$ such that, for all $t < T$ and all $s \in \{1,\dots,n\}$,*
   - $p^{\leftarrow}(s) > 0$,
   - $\Pr[X_t - X_{t+1} = 1 | X_t = s] \leq p^{\leftarrow}(s)$,
   - $\Pr[X_t - X_{t+1} > 1 | X_t = s] = 0$, *and*
   - $\Pr[X_t - X_{t+1} \leq -1 | X_t = s] \geq p^{\rightarrow}(s)$ *(for $s \neq n$).*

   Then
$$\mathrm{E}[T \mid X_0] \geq \sum_{s=1}^{X_0} \sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)}.$$

The bounds on the expected first-hitting time given in Theorem 3 can be thought of as the sum of waiting times. Each waiting time is weighted with the ratio of how likely it is to go away from the goal 0 ($p^{\rightarrow}$) versus going toward it ($p^{\leftarrow}$). Note that the inner sum adds all waiting times up to n. This is where we need that the state space is bounded.

For case 1, note that it does not matter how far left the process moves. In fact, in the proof, we assume the worst case of the process only moving one step closer to the goal. However, we need to guarantee that we can move at most one step away from the goal. The converse is true for case 2: here, we need to guarantee that the process can only move a single step closer to the goal but is allowed to go arbitrarily far away (given its finite state space). Consequently, if the exact transition probabilities are known and, when in state s, the process can only move to the states $s-1$, s, and $s+1$ (if possible), both cases coincide and Theorem 3 yields the exact first-hitting time of the process.

The proof of Theorem 3 is an application of Theorem 1 with a scaled process (a potential) such that the drift can be bounded by 1. The expected first-hitting time is then bounded by the potential of the starting state.

Proof (of Theorem 3). For both cases, we define a potential function $\phi\colon \{0,\dots,n\} \to \mathbb{R}_{\geq 0}$, for $s \in \{0,\dots,n\}$, as follows, using the respective definitions of $p^{\leftarrow}$ and $p^{\rightarrow}$:
$$\phi(s) = \sum_{i=1}^{s} g(i), \text{ where } g(s) = \sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} \text{ for } s \neq 0.$$

Note that ϕ is monotonically increasing and that, for all $t < T$, $\phi(X_t) = 0$ if and only if $X_t = 0$. Thus, the first point in time t such that $\phi(X_t) = 0$ is T.

We prove that the following recursion holds via downward induction over $s \in \{1, \dots, n\}$:

$$g(n) = \frac{1}{p^{\leftarrow}(n)} \text{ and } g(s) = \frac{1}{p^{\leftarrow}(s)} + \frac{p^{\rightarrow}(s)}{p^{\leftarrow}(s)} g(s+1) \text{ for } s \neq n.$$

For the base case $s = n$, we get $g(n) = \sum_{i=n}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=n}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} = \frac{1}{p^{\leftarrow}(n)}$, which is true. As for the inductive step, for $s \neq n$, we get

$$\begin{aligned}
&\frac{1}{p^{\leftarrow}(s)} + \frac{p^{\rightarrow}(s)}{p^{\leftarrow}(s)} g(s+1) = \frac{1}{p^{\leftarrow}(s)} + \frac{p^{\rightarrow}(s)}{p^{\leftarrow}(s)} \sum_{i=s+1}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s+1}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} \\
&= \frac{1}{p^{\leftarrow}(s)} + \sum_{i=s+1}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} = \sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} = g(s).
\end{aligned}$$

Consider case 1. We first compute the drift for $t < T$ and for $X_t = n$:

$$\begin{aligned}
\mathrm{E}[\phi(X_t) - \phi(X_{t+1}) \,|\, X_t = n] &\geq \Pr X_t - X_{t+1} \geq 1[X_t = n] \cdot \big(\phi(n) - \phi(n-1)\big) \\
&\geq p^{\leftarrow}(n) \cdot \big(\phi(n) - \phi(n-1)\big) = p^{\leftarrow}(n) \cdot g(n) = 1,
\end{aligned}$$

where the first inequality follows from the monotonicity of ϕ.

For $s \in \{1, \dots, n-1\}$ and $t < T$, we get

$$\begin{aligned}
&\mathrm{E}[\phi(X_t) - \phi(X_{t+1}) \,|\, X_t = s] \geq \Pr[X_t - X_{t+1} \geq 1 | X_t = s] \cdot \big(\phi(s) - \phi(s-1)\big) \\
&\qquad + \Pr[X_t - X_{t+1} = -1 | X_t = s] \cdot \big(\phi(s) - \phi(s+1)\big) \\
&= \Pr[X_t - X_{t+1} \geq 1 | X_t = s] \cdot g(s) - \Pr[X_t - X_{t+1} = -1 | X_t = s] \cdot g(s+1) \\
&\geq p^{\leftarrow}(s) \cdot g(s) - p^{\rightarrow}(s) \cdot g(s+1) \\
&= p^{\leftarrow}(s) \cdot \left(\frac{1}{p^{\leftarrow}(s)} + \frac{p^{\rightarrow}(s)}{p^{\leftarrow}(s)} g(s+1) \right) - p^{\rightarrow}(s) \cdot g(s+1) = 1,
\end{aligned}$$

using our recursion scheme for g. Again, the first inequality follows from the monotonicity of ϕ.

Since we have a drift of at least 1 in all cases and a bounded step size, we can apply Theorem 1 and get the desired result.

For case 2, we can perform analogous estimations for the drift but into the other direction, making use that $-\phi(s+1)$ is an upper bound for $-\phi(s')$ for all $s' \geq s+1$. This way, we can upper-bound the drift by 1, yielding the respective lower bound when using Theorem 1. □

Note that the recursion of function g given in the proof is defined as an upward recursion. This actually follows from reconstructing how g has to look in order for the drift to be 1. This approach *cannot* be done in this fashion with a downward recursion, using state 1 as base case, as it is not clear what the

potential for that state has to be, since it has two neighboring states. Thus, it is very important for the search space to be bounded, leading to a well defined base case of $g(n) = 1/p^{\leftarrow}(n)$.

We highlight the importance of the upper bound on the state space (that is, its finiteness) in the following example, where we show that Theorem 2 cannot be easily extended such that it works in a scenario where its drift function h is not monotone.

Example 4. Consider two functions $p^{\leftarrow}\colon \{1,\dots,n\} \to [0,1]$ and $p^{\rightarrow}\colon \{0,\dots,n-1\} \to [0,1]$ such that, for all $s \in \{1,\dots,n\}$,

- $p^{\leftarrow}(s) = \frac{1}{2s}$ (for $s \neq 0$) and
- $p^{\rightarrow}(s) = \frac{1}{2(s+1)}$ (for $s \neq n$).

Note that, for all $s \in \{1,\dots,n-1\}$, $p^{\leftarrow}(s) + p^{\rightarrow}(s) \leq 1$.

Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\{0,\dots,n\}$ and let T denote the first point in time t such that $X_t = 0$. Suppose that, for all $s \in \{1,\dots,n\}$,

- $\Pr[X_t - X_{t+1} = 1 | X_t = s] = p^{\leftarrow}(s)$ (for $s \neq 0$),
- $\Pr[X_t - X_{t+1} = -1 | X_t = s] = p^{\rightarrow}(s)$ (for $s \neq n$), and
- $\Pr[X_t - X_{t+1} = 0 | X_t = s] = 1 - p^{\leftarrow}(s) - p^{\rightarrow}(s)$.

First, we consider the drift of this process. For all $s \in \{1,\dots,n-1\}$, we get

$$\mathrm{E}[X_t - X_{t+1} \,|\, X_t = s] = \frac{1}{2s} - \frac{1}{2(s+1)} = \frac{s+1}{2s(s+1)} - \frac{s}{2s(s+1)} = \frac{1}{2s(s+1)},$$

and for $s = n$, we get $\mathrm{E}[X_t - X_{t+1} \,|\, X_t = n] = \frac{1}{2n} \geq \frac{1}{2n(n+1)}$. Thus, the drift is dependent on the current state of the process. Note that this dependency is *not* monotonically increasing. However, we ignore this and apply Theorem 2 anyway. Hence, defining $h(s) = 1/\big(2s(s+1)\big)$, we get

$$\begin{aligned}\mathrm{E}[T \,|\, X_0] &\leq \frac{1}{h(1)} + \int_1^{X_0} \frac{1}{h(x)}\mathrm{d}x = 2 \cdot 2 + \int_1^{X_0} 2x(x+1)\mathrm{d}x \\ &= 4 + \frac{2}{3}x^3\Big|_1^{X_0} + x^2|_1^{X_0} = 4 + \frac{2}{3}X_0^3 - \frac{2}{3} + X_0^2 - 1 = \mathrm{O}\big(X_0^3\big).\end{aligned}$$

We now contrast this result with the result following from Theorem 3. Note that our functions $p^{\leftarrow}$ and $p^{\rightarrow}$ are *equal* to the transition probabilities of our random process. Thus, Theorem 3 yields an exact result, as the upper and lower bound coincide:

$$\mathrm{E}[T \,|\, X_0] = \sum_{s=1}^{X_0} \sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)}.$$

First, we calculate the product in the expected first-hitting time:

$$\prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} = \prod_{j=s}^{i-1} \frac{\frac{1}{2(j+1)}}{\frac{1}{2j}} = \prod_{j=s}^{i-1} \frac{j}{j+1} = \frac{s}{i},$$

as this is a telescope product and the numerator and denominator of neighboring factors cancel out.

As for the inner sum, we now get $\sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} = \sum_{i=s}^{n} 2i\frac{s}{i} = \sum_{i=s}^{n} 2s = 2s(n-s+1)$, since the sum is independent of its summation index i.

Last, for the outer sum, we get

$$\sum_{s=1}^{X_0} \sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} = \sum_{s=1}^{X_0} 2s(n-s+1) = 2\left((n+1)\sum_{s=1}^{X_0} s - \sum_{s=1}^{X_0} s^2\right)$$
$$= 2\left((n+1)\frac{X_0(X_0+1)}{2} - \frac{X_0(X_0+1)(2X_0+1)}{6}\right) = \Theta\big(nX_0^2\big),$$

because $0 \leq X_0 \leq n$, which means that the minuend dominates the difference in the second-to-last line. Overall, Theorem 3 yields $\mathrm{E}[T \mid X_0] = \Theta\big(nX_0^2\big)$.

If we compare this result against the expected first-hitting time due to Theorem 2 of $\mathrm{E}[T \mid X_0] = \mathrm{O}\big(X_0^3\big)$, we see that these results contradict one another if $X_0 = \mathrm{o}(n)$. In fact, if we choose $X_0 = 1$, that is, we are almost at our goal of 0 and have a constant probability of reaching it, the (erroneous) result of Theorem 2 yields a constant first-hitting time, whereas the truth is a first-hitting time linear in n.

Intuitively, this drastic difference comes from the high probability of the process going away from the goal instead of toward it. Thus, if our process does not go toward 0, it may take some time until it returns to 1. Even more important: this waiting time until returning to 1 is dependent on the size of the search space, namely n, as evident by the factor of n in the first-hitting time. Thus, if our search space were unbounded, the expected first-hitting time would be unbounded too, as the probability of returning to 1 would be too small.

This has an even bigger impact on Theorem 2: its result does *not* include the size of the search space.[1] This means that the theorem is inherently not capable of yielding the correct expected first-hitting time in the form given.

When choosing $X_0 = \Theta(n)$, the results of both theorems coincide. In this case, the process starts so far away from the goal 0 that the return time to X_0 is negligible. However, note that this is again due to the search space being bounded. As we start close to the upper bound n, it either takes a short time to return to X_0 (if going away from 0) or we approach the goal. ■

Since Example 4 does not give different results when $X_0 = \Theta(n)$, we provide another example, where the difference in the bounds from Theorems 2 and 3 is tremendous.

Example 5. Consider two functions $p^{\leftarrow}\colon \{1,\dots,n\} \to [0,1]$ and $p^{\rightarrow}\colon \{0,\dots,n-1\} \to [0,1]$ such that, for all $s \in \{1,\dots,n\}$,

- $p^{\leftarrow}(s) = \frac{1}{2}\big(1 + \frac{1}{\mathrm{e}^s}\big)$ (for $s \neq 0$) and
- $p^{\rightarrow}(s) = \frac{1}{2}\big(1 - \frac{1}{\mathrm{e}^s}\big)$ (for $s \neq n$).

[1] The theorem itself assumes the search space to be bounded. However, the actual size of the search space does not matter for the expected first-hitting time.

Note that, for all $s \in \{1, \dots, n-1\}$, $p^{\leftarrow}(s) + p^{\rightarrow}(s) = 1$.

Let $(X_t)_{t \in \mathbb{N}}$ be a random process over $\{0, \dots, n\}$ and let T denote the first point in time t such that $X_t = 0$. Suppose that, for all $s \in \{1, \dots, n\}$,

- $\Pr[X_t - X_{t+1} = 1 | X_t = s] = p^{\leftarrow}(s)$ (for $s \neq 0$),
- $\Pr[X_t - X_{t+1} = -1 | X_t = s] = p^{\rightarrow}(s)$ (for $s \neq n$), and
- $\Pr[X_t - X_{t+1} = 0 | X_t = n] = 1 - p^{\leftarrow}(n)$.

Thus, X_t is almost an unbiased random walk on a path. Further, we assume that $X_0 = n$.

First, we consider the drift of this process. For all $s \in \{1, \dots, n-1\}$, we get $\mathrm{E}[X_t - X_{t+1} \mid X_t = s] = \frac{1}{2}\big(1 + \frac{1}{e^s}\big) - \frac{1}{2}\big(1 - \frac{1}{e^s}\big) = \frac{1}{e^s}$, and for $s = n$, we get $\mathrm{E}[X_t - X_{t+1} \mid X_t = n] = \frac{1}{2}\big(1 + \frac{1}{\mathrm{e}^n}\big) \geq \frac{1}{\mathrm{e}^n}$.

By wrongly applying Theorem 2 with $h(s) = 1/\mathrm{e}^s$, we get $\mathrm{E}[T \mid X_0] \leq \frac{1}{h(1)} + \int_1^{X_0} \frac{1}{h(x)} \mathrm{d}x = \mathrm{e} + \mathrm{e}^n - \mathrm{e} = \mathrm{e}^n$.

We now consider the application of Theorem 3. First, we estimate the product $\prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} \leq 1$. Hence, for the inner sum, we get $\sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} \leq \sum_{i=s}^{n} \frac{2\mathrm{e}^i}{\mathrm{e}^i + 1} \leq \sum_{i=s}^{n} 2 = 2(n - s + 1)$.

Last, for the outer sum, we get

$$\sum_{s=1}^{X_0} \sum_{i=s}^{n} \frac{1}{p^{\leftarrow}(i)} \prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)} \leq \sum_{s=1}^{X_0} 2(n - s + 1) = 2\left((n+1)\sum_{s=1}^{n} 1 - \sum_{s=1}^{n} s\right) = \mathrm{O}\big(n^2\big).$$

Overall, Theorem 3 yields $\mathrm{E}[T \mid X_0] = \mathrm{O}\big(n^2\big)$.

Comparing this with the bound of $\mathrm{O}(\mathrm{e}^n)$ when (wrongly) applying Theorem 2, we see that there is an exponential gap between both results. The result from Theorem 2 is not wrong but nonetheless very much off from the truth. Due to the exponentially declining drift, the bound is exponential. However, the actual first-hitting time of X_t is dominated by the first-hitting time of an unbiased random walk, which hits 0, starting from n, within $\Theta\big(n^2\big)$ steps in expectation. The result from Theorem 3 conforms to this argument. ■

Theorem 3 allows for arbitrary transition probabilities, as long as the state 0 can be reached. If we now restrict the transition probabilities such that the process cannot move to the right, we end up in a scenario where the process can either move closer to the target or stay at its current position. Thus, the expected first-hitting time is the sum of geometrically distributed random variables denoting the number of steps until each state is left. This way, we reconstruct the *fitness level method*.

Corollary 6 (Fitness Level Method [13]). *Let $(X_t)_{t \in \mathbb{N}}$ be a random process over $\{0, \dots, n\}$ and let T denote the first point in time t such that $X_t = 0$. Suppose, for all $t < T$, $X_t - X_{t+1} \geq 0$.*

1. *Suppose there exists a function $p^{\leftarrow} \colon \{1, \dots, n\} \to [0, 1]$ such that, for all $t < T$ and all $s \in \{1, \dots, n\}$,*

- $p^{\leftarrow}(s) > 0$ *and*
- $\Pr[X_t - X_{t+1} \geq 1 | X_t = s] \geq p^{\leftarrow}(s)$.

Then $\mathrm{E}[T \mid X_0] \leq \sum_{s=1}^{X_0} \frac{1}{p^{\leftarrow}(s)}$.

2. *Suppose there exists a function* $p^{\leftarrow} \colon \{1, \dots, n\} \to [0,1]$ *such that, for all* $t < T$ *and all* $s \in \{1, \dots, n\}$,
   - $p^{\leftarrow}(s) > 0$,
   - $\Pr[X_t - X_{t+1} = 1 | X_t = s] \leq p^{\leftarrow}(s)$, *and*
   - $\Pr[X_t - X_{t+1} > 1 | X_t = s] = 0$.

   Then $\mathrm{E}[T \mid X_0] \geq \sum_{s=1}^{X_0} \frac{1}{p^{\leftarrow}(s)}$.

Proof. Both inequalities directly follow from Theorem 3 by noting that the product $\prod_{j=s}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j)}$ is 0 for each $i \geq s+1$, and 1 for $i = s$. □

Note how case 2 assumes that the process can only move one step closer to the goal. If this were not the case, the process could reach the goal 0 earlier (for example, directly from X_0) and we had not to sum over all states between 0 and X_0.

4 Limit Distributions and Mixing Times

Our setting described in Sect. 2 can be interpreted as a Markov chain as depicted in Fig. 1. In this section, we are going to analyze our random process with respect to tools from the theory of Markov chains. We assume that the reader is familiar with the standard terminology in this topic and point to *Markov Chains and Mixing Times* [10] for a nice reference.

In Sect. 3, we determined the expected first-hitting time of a random process on a finite state space. Now we focus on the probability of being in a certain state after a certain time. More specifically, we are interested in a stationary distribution of our process as well as its mixing time. We start with determining a stationary distribution.

In this section, we assume that the process can only move to neighboring states. That is, when in state s, the process can only move to $s-1$, s, and $s+1$ (if possible).

According to Corollary 1.17 from [10], a stationary distribution of a Markov chain is unique if the chain is irreducible, that is, every state can be reached from any other state with positive probability. As we are interested in unique stationary distributions, our following theorem assumes that all transition probabilities to neighboring states are positive.

Theorem 7. *Let* $(X_t)_{t \in \mathbb{N}}$ *be a random process over* $\{0, \dots, n\}$. *Suppose, for all* $t \in \mathbb{N}$, $(X_t - X_{t+1}) \in \{-1, 0, 1\}$. *Suppose there are two functions* $p^{\leftarrow} \colon \{1, \dots, n\} \to [0,1]$ *and* $p^{\rightarrow} \colon \{0, \dots, n-1\} \to [0,1]$ *such that, for all* $s \in \{0, \dots, n\}$,

- $\Pr[X_t - X_{t+1} = 1 | X_t = s] = p^{\leftarrow}(s) > 0$ *(for* $s \neq 0$*) and*
- $\Pr[X_t - X_{t+1} = -1 | X_t = s] = p^{\rightarrow}(s) > 0$ *(for* $s \neq n$*).*

Further, let π denote the stationary distribution of X_t. Then, for all $s \in \{0, \dots, n\}$,

$$\pi(s) = \frac{\prod_{i=0}^{s-1} \frac{p^{\rightarrow}(i)}{p^{\leftarrow}(i+1)}}{\sum_{i=0}^{n} \prod_{j=0}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j+1)}}.$$

Similar to Theorem 3, the ratio of the transition probabilities are very important for our result. However, different from Theorem 3, we now need the values $p^{\rightarrow}(s)/p^{\leftarrow}(s+1)$ instead of $p^{\rightarrow}(s)/p^{\leftarrow}(s)$. This difference makes sense: in Sect. 3, we were interested in reaching the state 0. Thus, it was important how likely the process moves toward or from the goal. Now, there is no special state that we want to reach. We are interested in the probability of being in a certain state. Hence, it is important with which probability to get to a state and with which probability to leave it again.

If the ratio $p^{\rightarrow}(s)/p^{\leftarrow}(s+1)$ is the same for all states s, we can simplify the stationary distribution as follows.

Corollary 8. *Let $(X_t)_{t \in \mathbb{N}}$ be a random process over $\{0, \dots, n\}$. Suppose, for all $t \in \mathbb{N}$, $(X_t - X_{t+1}) \in \{-1, 0, 1\}$. Further, suppose there are two functions $p^{\leftarrow} \colon \{1, \dots, n\} \to [0, 1]$ and $p^{\rightarrow} \colon \{0, \dots, n-1\} \to [0, 1]$ and a value $c > 0$ such that, for all $s \in \{0, \dots, n\}$,*

- $\Pr[X_t - X_{t+1} = 1 | X_t = s] = p^{\leftarrow}(s) > 0$ *(for $s \neq 0$),*
- $\Pr[X_t - X_{t+1} = -1 | X_t = s] = p^{\rightarrow}(s) > 0$ *(for $s \neq n$), and*
- $p^{\rightarrow}(s)/p^{\leftarrow}(s+1) = c$.

Further, let π denote the stationary distribution of X_t. If $c \neq 1$, then, for all $s \in \{0, \dots, n\}$, it holds that $\pi(s) = (c-1)\frac{c^s}{c^{n+1}-c}$.

And if $c = 1$, for all $s \in \{0, \dots, n\}$, it holds that $\pi(s) = \frac{1}{n+1}$.

Proof. We use the definition of π from Theorem 7 and use that $\prod_{j=0}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j+1)} = \prod_{j=0}^{i-1} c = c^i$. Hence, we get $\pi(s) = \frac{\prod_{i=0}^{s-1} \frac{p^{\rightarrow}(i)}{p^{\leftarrow}(i+1)}}{\sum_{i=0}^{n} \prod_{j=0}^{i-1} \frac{p^{\rightarrow}(j)}{p^{\leftarrow}(j+1)}} = \frac{c^s}{\sum_{i=0}^{n} c^i}$, where the result follows by noting that the denominator is a geometric sum (when $c \neq 1$). For $c = 1$, the result is trivial. □

When we have $c < 1$, that is, the probability to move to the right is less than the probability to move to the left, Corollary 8 yields that the probability to be in state s declines exponentially in s. Conversely, if $c > 1$, the probability grows exponentially in s. Last, if $c = 1$, we end up with the uniform distribution.

Given the stationary distribution, it is a natural question to ask whether this distribution will be reached in the limit of the number of steps going to infinity. This is the case if the Markov chain is also aperiodic (besides irreducible; Theorem 4.9 from [10]), which is the case if there is at least one state that has a self loop (that is, it has a positive probability of reaching itself in one step).

Assuming that our Markov chain is also aperiodic, we now determine its mixing time, that is, the time until the probability to be in state s is only at most ε different from the probability stated by the stationary distribution.

The following corollary is a direct consequence of Corollary 14.7 from [10]. The idea behind the corollary is to consider a coupling of two independent copies of the process that start maximally far apart. The goal is that both processes meet. If they arrive in neighboring states and at least one of those states has a self loop, this is possible by one process staying where it is and the other process moving to said state. In order for this argument to translate into a mixing time, it is necessary that the expected distance of two such neighboring processes is less than 1 after one step. That is, in expectation, they move close to one another once they are next to each other. This is formalized in the following corollary.

Corollary 9. *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\{0,\dots,n\}$. Suppose, for all $t \in \mathbb{N}$, $(X_t - X_{t+1}) \in \{-1,0,1\}$. Further, suppose there are two functions $p^{\leftarrow}\colon \{1,\dots,n\} \to [0,1]$ and $p^{\rightarrow}\colon \{0,\dots,n-1\} \to [0,1]$ such that, for all $s \in \{0,\dots,n\}$,*

- $\Pr[X_t - X_{t+1} = 1 | X_t = s] = p^{\leftarrow}(s) > 0$ *(for $s \neq 0$) and*
- $\Pr[X_t - X_{t+1} = -1 | X_t = s] = p^{\rightarrow}(s) > 0$ *(for $s \neq n$).*

Let $(Y_t)_{t\in\mathbb{N}}$ be an independent copy of X_t. Assume that there is an $\alpha > 0$ such that, for all $s \in \{0,\dots,n-1\}$,

$$\mathrm{E}\big[|X_{t+1} - Y_{t+1}| \,\big|\, X_t = s, Y_t = s+1\big] \leq \mathrm{e}^{-\alpha}. \tag{1}$$

Let t_{mix} denote the mixing time of X_t. Then, for each $\varepsilon \in (0,1/2)$,

$$t_{\mathrm{mix}}(\varepsilon) \leq \left\lceil \frac{\ln \frac{n}{\varepsilon}}{\alpha} \right\rceil.$$

Proof. The statement is an application of Corollary 14.7 from [10]. Inequality (1) is a special case of the situation considered in Theorem 14.6 in [10]. □

Note that if we consider a state s such that both s and $s+1$ have no self loop, $\mathrm{E}\big[|X_{t+1} - Y_{t+1}| \,\big|\, X = s, Y_t = s+1\big]$ is at least 1, as X_{t+1} and Y_{t+1} cannot be in the same state. Thus, Corollary 9 is not applicable in this scenario.

References

1. Dang, D.C., Lehre, P.K.: Runtime analysis of non-elitist populations: from classical optimisation to partial information. Algorithmica **75**(3), 428–461 (2016)
2. Droste, S., Jansen, T., Wegener, I.: Dynamic parameter control in simple evolutionary algorithms. In: Proceedings of the FOGA 2000, pp. 275–294 (2000)
3. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **127**(1), 57–85 (2001)
4. He, J., Yao, X.: Erratum to: drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **140**(1), 245–248 (2002)

5. He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. Nat. Comput. **3**(1), 21–35 (2004)
6. Jansen, T., Zarges, C.: Evolutionary algorithms and artificial immune systems on a bi-stable dynamic optimisation problem. In: Proceedings of the GECCO 2014, pp. 975–982 (2014)
7. Johannsen, D.: Random combinatorial structures and randomized search heuristics. Ph.D. thesis, Universität des Saarlandes (2010). http://scidok.sulb.uni-saarland.de/volltexte/2011/3529/pdf/Dissertation3166JohaDani2010.pdf
8. Kötzing, T., Lissovoi, A., Witt, C.: (1 + 1) EA on generalized dynamic OneMax. In: Proceedings of the FOGA XIII, pp. 40–51 (2015)
9. Lengler, J.: Drift analysis. CoRR abs/1712.00964 (2017). http://arxiv.org/abs/1712.00964
10. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov Chains and Mixing Times. American Mathematical Society, Providence (2006)
11. Lissovoi, A., Witt, C.: MMAS versus population-based EA on a family of dynamic fitness functions. Algorithmica **75**(3), 554–576 (2016)
12. Mitavskiy, B., Rowe, J.E., Cannings, C.: Theoretical analysis of local search strategies to optimize network communication subject to preserving the total number of links. Int. J. Intell. Comput. Cybern. **2**(2), 243–284 (2009)
13. Wegener, I.: Theoretical aspects of evolutionary algorithms. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 64–78. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_6
14. Witt, C.: Fitness levels with tail bounds for the analysis of randomized search heuristics. Inf. Process. Lett. **114**(1–2), 38–41 (2014)

First-Hitting Times Under Additive Drift

Timo Kötzing and Martin S. Krejca(✉)

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
martin@krejca.de

Abstract. For the last ten years, almost every theoretical result concerning the expected run time of a randomized search heuristic used *drift theory*, making it the arguably most important tool in this domain. Its success is due to its ease of use and its powerful result: drift theory allows the user to derive bounds on the expected first-hitting time of a random process by bounding expected local changes of the process – the *drift*. This is usually far easier than bounding the expected first-hitting time directly.

Due to the widespread use of drift theory, it is of utmost importance to have the best drift theorems possible. We improve the fundamental additive, multiplicative, and variable drift theorems by stating them in a form as general as possible and providing examples of why the restrictions we keep are still necessary. Our additive drift theorem for upper bounds only requires the process to be nonnegative, that is, we remove unnecessary restrictions like a finite, discrete, or bounded search space. As corollaries, the same is true for our upper bounds in the case of variable and multiplicative drift.

1 Drift Theory

In the theory of randomized algorithms, the first and most important part of algorithm analysis is to compute the expected run time. A finite run time guarantees that the algorithm terminates almost surely, and, due to Markov's inequality, the probability of the run time being far larger than the expected value can be bounded, too. Thus, it is important to have strong and easy to handle tools in order to derive expected run times. The de facto standard for this purpose in the theory of randomized search heuristics is *drift theory*.

Drift theory is a general term for a collection of theorems that consider random processes and bound the expected time it takes the process to reach a certain value – the *first-hitting time*. The beauty and appeal of these theorems lie in them usually having few restrictions but yielding strong results. Intuitively speaking, in order to use a drift theorem, one only needs to estimate the expected change of a random process – the *drift* – at any given point in time. Hence, a drift theorem turns expected local changes of a process into expected first-hitting times. In other words, local information of the process is transformed into global information.

Drift theory gained traction in the theory of randomized search heuristics when it was introduced to the community by He and Yao [7,8] via the *additive*

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 92–104, 2018.
https://doi.org/10.1007/978-3-319-99259-4_8

drift theorem. However, they were not the first to prove it. The result dates back to Hajek [6], who stated the theorem in a fashion quite different from how it is phrased nowadays. According to Lengler [13], the theorem has been proven even prior to that various times. Since then, many different versions of drift theorems have been proven, the most common ones being the *variable drift theorem* [9] and the *multiplicative drift theorem* [3]. The different names refer to how the drift is bounded other than independent of time: additive means that the drift is bounded by the same value for all states; in a multiplicative scenario, the drift is bounded by a multiple of the current state of the process; and in the setting of variable drift, the drift is bounded by any monotone function with respect to the current state of the process.

At first, the theorems were only stated over finite or discrete search spaces. However, these restrictions are seldom used in the proofs and thus not necessary, as pointed out, for example, by Lehre and Witt [12], who prove a general drift theorem without these restrictions. Nonetheless, up to date, all drift theorems require a bounded search space;[1] Semenov and Terkel [19] state a Theorem very much like an additive drift theorem for unbounded search spaces, but they require the process to have a bounded variance, as they also prove concentration for their result.

The area of randomized search heuristics is, in fact, in strong need of extended drift theorems and a careful discussion of what happens when restrictions are not met. While most search spaces are finite and, thus, the existing drift theorems sufficient, progress will be inhibited whenever search spaces are not naturally finite. Worse yet, the existing drift theorems might be applied where they are not applicable, as happened when in [4, Sect. 4] the additive drift theorem was applied on an unbounded search space.

While previously new drift theorems were proven on a need-to-have basis using whatever restrictions where present in the concrete application, we aim at providing the best possible theorem for any applications to come. For the restrictions that remain, we give examples that show that these restrictions are, in some sense, necessary. In this way, we want to further the understanding of random processes in general and not just for a concrete application; thus, this work should benefit a lot of future work in the area of randomized search heuristics.

Our most important results are the upper and lower bound of the classical additive drift theorem (Theorems 5 and 7, respectively), which we prove for unbounded[2] search spaces. These theorems are used as a foundation for all of our other drift theorems in other settings. Overall, our results can be summarized as follows:

[1] Lengler [13] briefly mentions infinite search spaces and also gives a proof for a restricted version of the additive drift theorem in the setting of an unbounded discrete search space.

[2] For the upper bound, we require the search space to be lower-bounded but not upper-bounded. We still refer to such a setting as unbounded.

For **additive drift**, we prove an upper bound for any nonnegative process (Theorem 5), and a lower bound for processes with bounded *expected* step size (Theorem 7).

For **multiplicative drift** and **variable drift**, we prove upper bounds for any nonnegative process (Corollaries 11 and 12; and Theorems 9 and 10, respectively).

The intention of this paper is to provide a fully-packed reference for very general yet easy-to-apply drift theorems. That is, we try to keep the requirements of the theorems as easy as possible but still state the theorems in the most general way, given the restrictions. Further, we discuss the ideas behind the different theorems and some of the proofs in order to provide insights into how and why drift works, we provide examples, and we discuss prior work at the beginning of each section.

We only consider bounds on the expected first-hitting time, as this is already a vast field to explore. However, we want to mention that drift theory has also brought forth other results than expected first-hitting times, namely, concentration bounds and negative drift, which are related. Both areas bound *the probability* of the first-hitting time taking certain values. Concentration bounds show how unlikely it is for a process to take much longer than the expected first-hitting time [2,10]. On the other hand, negative drift bounds how likely it is for the process to reach the goal although the drift is going the opposite direction [10,16,17]. These results are also very helpful but out of the scope of this paper.

Our paper is structured as follows: in Sect. 2, we start by introducing important notation and terms, which we use throughout the entire paper. Further, we also discuss Theorem 1, which our proofs of the additive drift theorems rely on. In Sect. 3, we discuss additive drift and prove our main results. We then continue with variable drift in Sect. 4 as a generalization of additive drift. In this section, we introduce two different versions of first-hitting time that our results are based on. Last, we consider the scenario of multiplicative drift in Sect. 5.

Due to space limitations, many proofs are omitted in this paper. A version containing all of the proofs can be found on arXiv [11].

2 Preliminaries

We consider the expected *first-hitting time* T of a process $(X_t)_{t\in\mathbb{N}}$ over $\mathbb{R}$, which we call X_t for short. That is, we are interested in the expected time it takes the process to reach a certain value for the first time, which we will refer to as the *target*. Usually, our target is the value 0, that is, we will define the random variable $T = \inf\{t \mid X_t \leq 0\}$ (where we define that $\inf \emptyset := \infty$).

We provide bounds on $\mathrm{E}[T \mid X_0]$ with respect to the *drift* of X_t, which is defined as

$$X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t].$$

Note that $\mathrm{E}[T \mid X_0]$ as well as $\mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t]$ are both random variables. Because of the latter, the drift is a random variable, too. Further note that, if the drift is positive, X_t decreases its value in expectation over time when

considering positive starting values. This is why 0 will be our target most of the time.

We are only interested in the process X_t until the time point T. That is, all of our requirements only need to hold for all $t < T$ (since we also consider $t+1$). While this phrasing is intuitive, it is formally inaccurate, as T is a random variable. We will continue to use it; however, formally, each of our inequalities in each of our requirements should be multiplied with the characteristic function of the event $\{t < T\}$. In this way, the inequalities trivially hold once $t \geq T$ and, otherwise, are the inequalities we state. This is similar to conditioning on the event $\{t < T\}$ but has the benefit of being valid even if $\Pr[t < T] = 0$ holds.

We want to mention that all of our results actually hold for a random process $(X_t)_{t\in\mathbb{N}}$ adapted to a filtration $(\mathcal{F}_t)_{t\in\mathbb{N}}$, where T is a stopping time defined with respect to $\mathcal{F}_t$.[3] Since this detail is frequently ignored in drift theory, we phrase all of our results with respect to the natural filtration, making them look more familiar to usual drift results. For any time point $t \leq T$, we call $X_0, \ldots, X_{t-1}$ the *history* of the process.

Last, we state all of our results conditional on X_0, that is, we bound $\mathrm{E}[T \mid X_0]$. However, by the law of total expectation, one can easily derive a bound for $\mathrm{E}[T] = \mathrm{E}\big[\mathrm{E}[T \mid X_0]\big]$.

2.1 Martingale Theorems

In this section, we state two theorems that we will use in order to prove our results in the next sections. Both theorems make use of *martingales*, a fundamental concept in the field of probability theory. A martingale is a random process with a drift of 0, that is, in expectation, it does not change over time. Further, a *supermartingale* has a drift of at least 0, that is, it decreases over time in expectation, and a *submartingale* has a drift of at most 0, that is, it increases over time in expectation.

The arguably most important theorem for martingales is the Optional Stopping Theorem (Theorem 1). We use a version given by Grimmett and Stirzaker [5, Chap. 12.5, Theorem 9] that can be extended to super- and submartingales.

Theorem 1 (Optional Stopping). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, and let T be a stopping time[4] for X_t. Suppose that*

(a) $\mathrm{E}[T] < \infty$ *and that*
(b) there is some value $c \geq 0$ *such that, for all* $t < T$*, it holds that* $\mathrm{E}\big[|X_{t+1} - X_t| \,\big|\, X_0, \ldots, X_t\big] \leq c$.

Then:

1. *If, for all* $t < T$, $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq 0$, *then* $\mathrm{E}[X_T] \leq \mathrm{E}[X_0]$.
2. *If, for all* $t < T$, $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \leq 0$, *then* $\mathrm{E}[X_T] \geq \mathrm{E}[X_0]$.

[3] More information on filtrations can be found, for example, in *Randomized Algorithms* [15] in the section on martingales.

[4] Intuitively, for the natural filtration, a stopping time T is a random variable over $\mathbb{N}$ such that, for all $t \in \mathbb{N}$, the event $\{t \leq T\}$ is only dependent on $X_0, \ldots, X_t$.

Theorem 1 allows us to bound $\mathrm{E}[X_T]$ independently of its history, which is why our drift results are independent of the history of X_T as well.

Note that case (1) refers to supermartingales, whereas case (2) refers to submartingales. Intuitively, case (1) says that a supermartingale will have, in expectation, a lower value than it started with, which makes sense, as a supermartingale decreases over time in expectation. Case (2) is analogous for submartingales. For martingales, both cases can be combined in order to yield an equality.

Martingales are essential in the proofs of our theorems. We will frequently transform our process such that it results in a supermartingale or a submartingale in order to apply Theorem 1.

Another useful theorem for martingales is the following Azuma–Hoeffding Inequality [1]. This inequality basically is for martingales what a Chernoff bound is for binomial distributions.

Theorem 2 (Azuma-Hoeffding Inequality). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$. Suppose that*

(a) there is some value $c > 0$ such that, for all $t \in \mathbb{N}$, it holds that $|X_t - X_{t+1}| < c$.

If, for all $t \in \mathbb{N}$, $X_t - \mathrm{E}[X_{t+1} \mid X_0, \dots, X_t] \geq 0$, then, for all $t \in \mathbb{N}$ and all $r > 0$, $\Pr[X_t - X_0 \geq r] \leq \mathrm{e}^{-\frac{r^2}{2tc^2}}$.

3 Additive Drift

We speak of additive drift when the drift can be bounded by a value independent of the process itself. That is, the bound is independent of time and state.

When considering the first-hitting time T of a random process $(X_t)_{t\in\mathbb{N}}$ whose drift is *lower*-bounded by a value $\delta > 0$, then $\mathrm{E}[T \mid X_0]$ is *upper*-bounded by X_0/δ. Interestingly, if the drift of X_t is *upper*-bounded by δ, $\mathrm{E}[T \mid X_0]$ is *lower*-bounded by X_0/δ. Thus, if the drift of X_t is exactly δ, that is, we know how much expected progress X_t makes in each step, our expected first-hitting time is equal to X_0/δ. This result is remarkable, as it can be understood intuitively as follows: since we stop once X_t reaches 0, the distance from our start (X_0) to our goal (0) is exactly X_0, and we make an expected progress of δ each step. Thus, in expectation, we are done after X_0/δ steps.

3.1 Upper Bounds

We give a proof for the Additive Drift Theorem, originally published (in a more restricted version) by He and Yao [7,8]. We start by reproving the original theorem (which requires a bounded search space) but in a simpler, more elegant and educational manner. We then greatly extend this result by generalizing it to processes with a bounded step width. Finally, we lift also this restriction.

In all of these cases, we require our random process to only take nonnegative values. The intuitive reason for this is the following: when estimating an upper bound for the expected first-hitting time, we need a lower bound of the drift. This means the larger our bound of the drift, the better our bound for the first-hitting time. Since our process is nonnegative, the drift for values close to 0 provides a natural bound for the drift (which is uniform over the entire search space, since we look at *additive* drift). If our process could take values less than 0, we could artificially increase our lower bound of the drift for values that are now bounded by 0 and, thus, improve our first-hitting time. At the end of this section, we also give an example (Example 6), which shows how our most general drift theorem (Theorem 5) fails if the process can take negative values.

The proof of the following theorem transforms the process into a supermartingale and then uses Theorem 1. However, in order to apply Theorem 1, we have to make sure to fulfill its condition (a), which is the hardest part.

Theorem 3 (Upper Additive Drift, Bounded). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, and let $T = \inf\{t \mid X_t \leq 0\}$. Furthermore, suppose that,*

(a) for all $t \leq T$, it holds that $X_t \geq 0$, that
(b) there is some value $\delta > 0$ such that, for all $t < T$, it holds that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq \delta$, and that
(c) there is some value $c \geq 0$ such that, for all $t < T$, it holds that $X_t \leq c$.

Then $\mathrm{E}[T \mid X_0] \leq \frac{X_0}{\delta}$.

Note that condition (a) means that T can be rewritten as $\inf\{t \mid X_t = 0\}$, that is, we have to hit 0 exactly in order to stop. We show in Example 6 why this condition is crucial.

Condition (b) bounds the expected progress we make each time step. The larger δ, the lower the expected first-hitting time. However, due to condition (a), note that small values of X_t create a natural upper bound for δ, as the progress for such values can be at most $|X_t - 0| = X_t$.

Condition (c) means that we are considering random variables over the interval $[0, c]$. It is a restriction that all previous additive drift theorems have but that is actually not necessary, as we show with Theorem 5. In the following proof, we use this condition in order to show that $\mathrm{E}[T] < \infty$, which is necessary when applying Theorem 1.

Proof (Proof of Theorem 3). We want to use case (1) of the Optional Stopping Theorem in the version of Theorem 1. Thus, we define, for all $t < T$, $Y_t = X_t + \delta t$, which is a supermartingale, since

$$\begin{aligned} Y_t - \mathrm{E}[Y_{t+1} \mid Y_0, \ldots, Y_t] &= X_t + \delta t - \mathrm{E}[X_{t+1} + \delta(t+1) \mid X_0, \ldots, X_t] \\ &= X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] - \delta \geq 0, \end{aligned}$$

as we assume that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq \delta$ for all $t < T$. Note that we can change the condition $Y_0, \ldots, Y_t$ to $X_0, \ldots, X_t$ because the transformation from X_t to Y_t is injective.

We now show that $\mathrm{E}[T \mid X_0] < \infty$ holds in order to apply Theorem 1. Let $r > 0$, and let a be any value such that $\Pr[X_0 \leq a] > 0$. We condition on the event $\{X_0 \leq a\}$, and we consider a time point $t' = (a + r)/\delta$ and want to bound the probability that $X_{t'}$ has not reached 0 yet, that is, the event $\{X_{t'} > 0\}$. We rewrite this event as $\{X_{t'} - a > -a\}$, which is equivalent to $\{Y_{t'} - a > -a + \delta t' = r\}$, by definition of Y and t'.

Note that, for all $t < T$, $|Y_t - Y_{t+1}| < c + \delta + 1$, as we assume that $X_t \leq c$. Thus, the differences of Y_t are bounded and we can apply Theorem 2 as follows, noting that $Y_0 = X_0 \leq a$, due to our condition on $\{X_0 \leq a\}$:

$$\Pr[Y_{t'} - a > r \mid X_0 \leq a] \leq \Pr[Y_{t'} - Y_0 \geq r \mid X_0 \leq a] \leq \mathrm{e}^{-\frac{r^2}{2t'(c+\delta+1)^2}}.$$

If we choose $r \geq a$, we get $t' \leq 2r/\delta$ and, thus,

$$\Pr[Y_{t'} - Y_0 > r \mid X_0 \leq a] \leq \mathrm{e}^{-\frac{r\delta}{4(c+\delta+1)^2}}.$$

This means that the probability that $X_{t'}$ has not reached 0 goes exponentially fast toward 0 as t' (and, hence, r) goes toward ∞. Thus, the expected value of T is finite.

Now we can use case (1) of Theorem 1 in order to get $\mathrm{E}[Y_T \mid X_0] \leq \mathrm{E}[Y_0 \mid X_0]$. In particular, noting that $X_T = 0$ by definition,

$$\begin{aligned} X_0 &= \mathrm{E}[X_0 \mid X_0] = \mathrm{E}[Y_0 \mid X_0] \\ &\geq \mathrm{E}[Y_T \mid X_0] = \mathrm{E}[X_T + \delta T \mid X_0] = \mathrm{E}[X_T \mid X_0] + \delta \mathrm{E}[T \mid X_0] = \delta \mathrm{E}[T \mid X_0]. \end{aligned}$$

Thus, we get the desired bound by dividing by δ. □

Note that the arguments in this proof only need the property of bounded differences in order to apply Theorem 2. Thus, we can relax the condition of a bounded state space into bounded step size, which can be seen in the following theorem.

Theorem 4 (Upper Additive Drift, Bounded Step Size). *Let $(X_t)_{t \in \mathbb{N}}$ be a random process over $\mathbb{R}$, and let $T = \inf\{t \mid X_t \leq 0\}$. Furthermore, suppose that,*

(a) for all $t \leq T$, it holds that $X_t \geq 0$, that
(b) there is some value $\delta > 0$ such that, for all $t < T$, it holds that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq \delta$, and that
(c) there is some value $c \geq 0$ such that, for all $t < T$, it holds that $|X_{t+1} - X_t| \leq c$.

Then $\mathrm{E}[T \mid X_0] \leq \frac{X_0}{\delta}$.

Although the proof of Theorem 3 can be used for Theorem 4 as well, we provide a different proof strategy in the appendix, which we then generalize for our next theorem. This alternative strategy defines a process similar to X_t that behaves like X_t in the limit.

The proof of Theorem 4 makes use of Theorem 3 by artificially bounding the search space for a time that is sufficient in order to bound the expected first-hitting time. This approach can be used in order to let the restriction of the bounded step size fall entirely. Since we cannot make many assumptions about the process in this case anymore, we rely on Markov's inequality in order to show that our process will not leave, with sufficiently high probability, an interval large enough to properly bound the expected first-hitting time.

Theorem 5 (Upper Additive Drift, Unbounded). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, and let $T = \inf\{t \mid X_t \leq 0\}$. Furthermore, suppose that,*

(a) for all $t \leq T$, it holds that $X_t \geq 0$, and that
(b) there is some value $\delta > 0$ such that, for all $t < T$, it holds that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq \delta$.

Then $\mathrm{E}[T \mid X_0] \leq \frac{X_0}{\delta}$.

As we already mentioned before, note that the condition of the process not being negative is important in order to get correct results. The following example highlights this fact.

Example 6. Let $n > 1$, and let $(X_t)_{t\in\mathbb{N}}$ be a random process with $X_0 = 1$ and, for all $t \in \mathbb{N}$, $X_{t+1} = X_t$ with probability $1-1/n$, and $X_{t+1} = -n+1$ otherwise. Let T denote the first point in time t such that the event $X_t \leq 0$ occurs. We have, for all $t < T$, that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] = 1$ and, thus, $\mathrm{E}[T \mid X_0] \leq 1$ if we could apply any of the additive drift theorems. However, since T follows a geometric distribution with success probability $1/n$, we have $\mathrm{E}[T \mid X_0] = n$.

3.2 Lower Bound

In this section, we provide a lower bound for the expected first-hitting time under additive drift. In order to do so, we need an upper bound for the drift. Since we now lower-bound the first-hitting time, a large upper bound of the drift makes the result bad. Thus, we can allow the process to take negative values, as these could only increase the drift's upper bound. However, we need to have some restriction on the step size in order to make sure not to move away from the target. Again, we provide an example (Example 8) showing this necessity at the end of this section.

Theorem 7 (Lower Additive Drift, Expected Bounded Step Size). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, and let $T = \inf\{t \mid X_t \leq 0\}$. Furthermore, suppose that*

(a) there is some value $\delta > 0$ such that, for all $t < T$, it holds that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \leq \delta$, and that
(b) there is some value $c \geq 0$ such that, for all $t < T$, it holds that $\mathrm{E}\big[|X_{t+1} - X_t| \,\big|\, X_0, \ldots, X_t\big] \leq c$.

Then $\mathrm{E}[T \mid X_0] \geq \frac{X_0}{\delta}$.

Proof. We make a case distinction with respect to $\mathrm{E}[T \mid X_0]$ being finite. If $\mathrm{E}[T \mid X_0]$ is infinite, then the theorem trivially holds. Thus, we now assume that $\mathrm{E}[T \mid X_0] < \infty$.

Similar to the proof of Theorem 3, we define, for all $t < T$, $Y_t = X_t + \delta t$, which is a submartingale, since

$$\begin{aligned} Y_t - \mathrm{E}[Y_{t+1} \mid Y_0, \dots, Y_t] &= X_t - \delta t - \mathrm{E}[X_{t+1} - \delta(t+1) \mid X_0, \dots, X_t] \\ &= X_t - \mathrm{E}[X_{t+1} \mid X_0, \dots, X_t] - \delta \leq 0, \end{aligned}$$

as we assume that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \dots, X_t] \leq \delta$ for all $t < T$ and because, again, the transformation of X_t to Y_t is injective.

Since we now assume that both $\mathrm{E}[T \mid X_0] < \infty$ and, further, that $\mathrm{E}\big[|X_{t+1} - X_t| \,\big|\, X_0, \dots, X_t\big] \leq c$ for all $t < T$, we can directly apply case (2) of Theorem 1 and get that $\mathrm{E}[Y_T \mid X_0] \geq \mathrm{E}[Y_0 \mid X_0]$. This yields, noting that $X_T \leq 0$,

$$\begin{aligned} X_0 &= \mathrm{E}[X_0 \mid X_0] = \mathrm{E}[Y_0 \mid X_0] \\ &\leq \mathrm{E}[Y_T \mid X_0] = \mathrm{E}[X_T + \delta T \mid X_0] = \mathrm{E}[X_T \mid X_0] + \delta \mathrm{E}[T \mid X_0] \leq \delta \mathrm{E}[T \mid X_0]. \end{aligned}$$

Thus, we get the desired bound by dividing by δ. □

Note that the step size has to be bounded in some way for a lower bound, as the following example shows.

Example 8. Let $\delta \in (0, 1)$, and let $(X_t)_{t \in \mathbb{N}}$ be a random process with $X_0 = 2$ and, for all $t \in \mathbb{N}$, $X_{t+1} = 0$ with probability $1/2$ and $X_{t+1} = 2X_t - 2\delta$ otherwise. Further, let T denote the first point in time t such that $X_t = 0$. Then T follows a geometric distribution with success probability $1/2$, which yields $\mathrm{E}[T] = 2$. However, we have that $X_t - \mathrm{E}[X_{t+1} \mid X_0, \dots, X_t] = \delta$. If Theorem 7 could be applied to this process (by neglecting the condition of the bounded step size), the theorem would yield that $\mathrm{E}[T] \geq 2/\delta$, which is not true.

4 Variable Drift

In contrast to additive drift, *variable* drift means that the drift can depend on the current state of the process (while still being bounded independently of the time). Interestingly, these more flexible drift theorems can be derived by using additive drift. Intuitively, the reasoning behind this approach is to scale the search space such that the information relevant to the process's history cancels out.

It is important to note that variable drift theorems are commonly phrased such that the first-hitting time T denotes the first point in time such that the random process drops *below* a certain value (our target) – it is not enough to hit that value. However, this restriction is not always necessary. Thus, we also

consider the setting from Sect. 3, where T denotes the first point in time such that we *hit* our target. In this section, our target is no longer 0 but a value $x_{\min}$.

In all of our theorems in this section, we make use of a set D. This set contains (at least) all possible values that our process can take while not having reached the target yet. It is a formal necessity in order to calculate the bound of the first-hitting time (via an integral). However, when applying the theorem, it is usually sufficient to choose $D = \mathbb{R}$ or $D = \mathbb{R}_{\geq 0}$.

The first variable drift theorem was proven by Johannsen [9] and, independently in a different version, by Mitavskiy et al. [14]. It was later refined by Rowe and Sudholt [18]. In all of these versions, bounded search spaces were used. Due to Theorem 5, we can drop this restriction.

Going Below the Target. The following version of the theorem assumes that the process has to drop below the target, denoted by $x_{\min}$. We provide the other version afterward.

Theorem 9 (Upper Variable Drift, Unbounded, Below Target). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, $x_{\min} > 0$, and let $T = \inf\{t \mid X_t < x_{\min}\}$. Additionally, let D denote the smallest real interval that contains at least all values $x \geq x_{\min}$ that, for all $t \leq T$, any X_t can take. Furthermore, suppose that*

(a) $X_0 \geq x_{\min}$ and, for all $t \leq T$, it holds that $X_t \geq 0$ and that
(b) there is a monotonically increasing function $h\colon D \to \mathbb{R}^+$ such that, for all $t < T$, we have $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq h(X_t)$.

Then

$$\mathrm{E}[T \mid X_0] \leq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(z)} \mathrm{d}z.$$

Hitting the Target. As mentioned before, it is not always necessary to drop below the target. For the additive drift, for example, we are interested in the first time reaching the target. Interestingly, the proof for the following theorem is straightforward, as it is almost the same as the proof of Theorem 9. Intuitively, the waiting time for getting below the target, once it is reached, is eliminated from the expected first-hitting time. However, it is important to note that it is now not allowed to get below the target.

Theorem 10 (Upper Variable Drift, Unbounded, Hitting Target). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, $x_{\min} \geq 0$, and let $T = \inf\{t \mid X_t \leq x_{\min}\}$. Additionally, let D denote the smallest real interval that contains at least all values $x \geq x_{\min}$ that, for all $t \leq T$, any X_t can take. Furthermore, suppose that,*

(a) for all $t \leq T$, it holds that $X_t \geq x_{\min}$ and that
(b) there is a monotonically increasing function $h\colon D \to \mathbb{R}^+$ such that, for all $t < T$, we have $X_t - \mathrm{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq h(X_t)$.

Then

$$\mathrm{E}[T \mid X_0] \leq \int_{x_{\min}}^{X_0} \frac{1}{h(z)} \mathrm{d}z.$$

5 Multiplicative Drift

A special case of variable drift is *multiplicative* drift, where the drift can be bounded by a multiple of the most recent value in the history of the process. As before, we provide upper bounds in the two versions of either dropping below the target or hitting it. In this setting, it can be intuitively argued why the version of dropping below the target is useful: consider a sequence of nonnegative numbers that halves its current value each time step. This process will never reach 0 within finite time. However, it drops below any value greater than 0.

Both upper bounds we state are simple applications of the corresponding variable drift theorems from Sect. 4.

Going Below the Target. Corollary 11 has first been stated by Doerr et al. [3] using finite state spaces. However, a closer look at the proof shows that this restriction is not necessary.

Corollary 11 (Upper Multiplicative Drift, Unbounded, Below Target). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, $x_{\min} > 0$, and let $T = \inf\{t \mid X_t < x_{\min}\}$. Furthermore, suppose that*

(a) $X_0 \geq x_{\min}$ and, for all $t \leq T$, it holds that $X_t \geq 0$, and that
(b) there is some value $\delta > 0$ such that, for all $t < T$, it holds that $X_t - \mathrm{E}\big[X_{t+1} \,\big|\, X_0, \dots, X_t\big] \geq \delta X_t$.

Then

$$\mathrm{E}[T \mid X_0] \leq \frac{1 + \ln\left(\frac{X_0}{x_{\min}}\right)}{\delta}.$$

Hitting the Target. By applying Theorem 10 instead of Theorem 9, we get the following theorem. As in the case of Theorem 10, the process now has to be lower-bounded by $x_{\min}$.

Corollary 12 (Upper Multiplicative Drift, Unbounded, Hitting Target). *Let $(X_t)_{t\in\mathbb{N}}$ be a random process over $\mathbb{R}$, $x_{\min} > 0$, and let $T = \inf\{t \mid X_t \leq x_{\min}\}$. Furthermore, suppose that,*

(a) for all $t \leq T$, it holds that $X_t \geq x_{\min}$, and that
(b) there is some value $\delta > 0$ such that, for all $t < T$, it holds that $X_t - \mathrm{E}\big[X_{t+1} \,\big|\, X_0, \dots, X_t\big] \geq \delta X_t$.

Then

$$\mathrm{E}[T \mid X_0] \leq \frac{\ln\left(\frac{X_0}{x_{\min}}\right)}{\delta}.$$

Again, we provide an example that shows that the bounds above are as tight as possible, up to constant factors, for the range of processes we consider. The example describes a process that decreases deterministically, that is, it has a variance of 0.

Example 13. Let $\delta \in (0,1)$ be a value bounded away from 1. Consider the process $(X_t)_{t\in\mathbb{N}}$, with $X_0 > 1$, that decreases each step deterministically such that $X_{t+1} = (1-\delta)X_t$ holds. Let T denote the first point in time such that the process drops below 1. Thus, we get $T = \Theta(-\log_{(1-\delta)} X_0) = \Theta\big(-\ln(X_0)/\ln(1-\delta)\big) = \Theta\big(\ln(X_0)/\delta\big)$, where the last equation makes use of the Taylor expansion of $\ln(1-\delta) = \Theta(-\delta)$, as $1-\delta$ does not converge to 0, by assumption.

References

1. Azuma, K.: Weighted sums of certain dependent random variables. Tohoku Math. J. **19**(3), 357–367 (1967)
2. Doerr, B., Goldberg, L.A.: Adaptive drift analysis. Algorithmica **65**(1), 224–250 (2013)
3. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. Algorithmica **64**(4), 673–697 (2012)
4. Doerr, B., Kötzing, T., Lagodzinski, J.A.G., Lengler, J.: Bounding bloat in genetic programming. In: Proceedings of the GECCO 2017, pp. 921–928 (2017)
5. Grimmett, G.R., Stirzaker, D.R.: Probability and Random Processes. Oxford University Press, Oxford (2001)
6. Hajek, B.: Hitting-time and occupation-time bounds implied by drift analysis with applications. Adv. Appl. Probab. **14**(3), 502–525 (1982)
7. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **127**(1), 57–85 (2001)
8. He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. Nat. Comput. **3**(1), 21–35 (2004)
9. Johannsen, D.: Random combinatorial structures and randomized search heuristics. Ph.D. thesis, Universität des Saarlandes (2010). http://scidok.sulb.uni-saarland.de/volltexte/2011/3529/pdf/Dissertation_3166_Joha_Dani_2010.pdf
10. Kötzing, T.: Concentration of first hitting times under additive drift. Algorithmica **75**(3), 490–506 (2016)
11. Kötzing, T., Krejca, M.S.: First-hitting times under additive drift. CoRR abs/1805.09415 (2018). https://arxiv.org/abs/1805.09415
12. Lehre, P.K., Witt, C.: Concentrated hitting times of randomized search heuristics with variable drift. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 686–697. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13075-0_54
13. Lengler, J.: Drift analysis. CoRR abs/1712.00964 (2017). http://arxiv.org/abs/1712.00964
14. Mitavskiy, B., Rowe, J.E., Cannings, C.: Theoretical analysis of local search strategies to optimize network communication subject to preserving the total number of links. Int. J. Intell. Comput. Cybern. **2**(2), 243–284 (2009)
15. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
16. Oliveto, P.S., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. Algorithmica **59**(3), 369–386 (2011)
17. Oliveto, P.S., Witt, C.: Erratum: simplified drift analysis for proving lower bounds in evolutionary computation. CoRR abs/1211.7184 (2012). http://arxiv.org/abs/1211.7184

18. Rowe, J.E., Sudholt, D.: The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. Theor. Comput. Sci. **545**, 20–38 (2014)
19. Semenov, M.A., Terkel, D.A.: Analysis of convergence of an evolutionary algorithm with self-adaptation using a Stochastic Lyapunov function. Evol. Comput. **11**(4), 363–379 (2003)

Level-Based Analysis of the Population-Based Incremental Learning Algorithm

Per Kristian Lehre(✉) and Phan Trung Hai Nguyen(✉)

University of Birmingham, Birmingham, UK
{p.k.lehre,p.nguyen}@cs.bham.ac.uk

Abstract. The Population-Based Incremental Learning (PBIL) algorithm uses a convex combination of the current model and the empirical model to construct the next model, which is then sampled to generate offspring. The Univariate Marginal Distribution Algorithm (UMDA) is a special case of the PBIL, where the current model is ignored. Dang and Lehre (GECCO 2015) showed that UMDA can optimise LeadingOnes efficiently. The question still remained open if the PBIL performs equally well. Here, by applying the level-based theorem in addition to Dvoretzky–Kiefer–Wolfowitz inequality, we show that the PBIL optimises function LeadingOnes in expected time $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$ for a population size $\lambda = \Omega(\log n)$, which matches the bound of the UMDA. Finally, we show that the result carries over to BinVal, giving the fist runtime result for the PBIL on the BinVal problem.

Keywords: Population-based incremental learning · LeadingOnes BinVal · Running time analysis · Level-based analysis · Theory

1 Introduction

Estimation of distribution algorithms (EDAs) are a class of randomised search heuristics that optimise objective functions by constructing probabilistic models and then sample the models to generate offspring for the next generation. Various variants of EDA have been proposed over the last decades; they differ from each other in the way their models are represented, updated as well as sampled over generations. In general, EDAs are usually categorised into two main classes: *univariate* and *multivariate*. Univariate EDAs take advantage of first-order statistics (i.e. mean) to build a univariate model, whereas multivariate EDAs apply higher-order statistics to model the correlations between the decision variables.

There are only a few runtime results available for EDAs. Recently, there has been a growing interest in the optimisation time of the UMDA, introduced by Mühlenbein and Paaß [11], on standard benchmark functions [4,7,8,13,14]. Recall that the optimisation time of an algorithm is the number of fitness evaluations the algorithm needs before a global optimum is sampled for the first time.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 105–116, 2018.
https://doi.org/10.1007/978-3-319-99259-4_9

Dang and Lehre [4] analysed a variant of the UMDA using truncation selection and derived the first upper bounds of $\mathcal{O}(n\lambda \log \lambda)$ and $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$ on the expected optimisation times of the UMDA on OneMax and LeadingOnes, respectively, where the population size is $\lambda = \Omega(\log n)$. These results were obtained using a relatively new technique called *level-based analysis* [3]. Very recently, Witt [13] proved that the UMDA optimises OneMax within $\mathcal{O}(\mu n)$ and $\mathcal{O}(\mu\sqrt{n})$ when $\mu \geq c \log n$ and $\mu \geq c'\sqrt{n}\log n$ for some constants $c,\ c' > 0$, respectively. However, these bounds only hold when $\lambda = (1 + \Theta(1))\mu$. This constraint on λ and μ was relaxed by Lehre and Nguyen [8], where the upper bound $\mathcal{O}(\lambda n)$ holds for $\lambda = \Omega(\mu)$ and $c \log n \leq \mu = \mathcal{O}(\sqrt{n})$ for some constant $c > 0$.

The first rigorous runtime analysis of the PBIL [1], was presented very recently by Wu et al. [14]. In this work, the PBIL was referred to as a cross entropy algorithm. The study proved an upper bound $\mathcal{O}\left(n^{2+\varepsilon}\right)$ of the PBIL with margins $[1/n, 1 - 1/n]$ on LeadingOnes, where $\lambda = n^{1+\varepsilon}$, $\mu = \mathcal{O}(n^{\varepsilon/2})$, $\eta \in \Omega(1)$ and $\varepsilon \in (0, 1)$. Until now, the known runtime bounds for the PBIL were significantly higher than those for the UMDA. Thus, it is of interest to determine whether the PBIL is less efficient than the UMDA, or whether the bounds derived in the early works were too loose.

This paper makes two contributions. First, we address the question above by deriving a tighter bound $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$ on the expected optimisation time of the PBIL on LeadingOnes. The bound holds for population sizes $\lambda = \Omega(\log n)$, which is a much weaker assumption than $\lambda = \omega(n)$ as required in [14]. Our proof is more straightforward than that in [14] because much of the complexities of the analysis are already handled by the level-based method [3].

The second contribution is the first runtime bound of the PBIL on BinVal. This function was shown to be the hardest among all linear functions for the cGA [5]. The result carries easily over from the level-based analysis of LeadingOnes using an identical partitioning of the search space. This observation further shows that runtime bounds, derived by the level-based method using the canonical partition, of the PBIL or other non-elitist population-based algorithms using truncation selection, on LeadingOnes also hold for BinVal.

The paper is structured as follows. Section 2 introduces the PBIL with margins as well as the level-based theorem, which is the main method employed in the paper. Given all necessary tools, the next two sections then provide upper bounds on the expected optimisation time of the PBIL on LeadingOnes and BinVal. Finally, our concluding remarks are given in Sect. 5.

2 Preliminaries

We first introduce the notations used throughout the paper. Let $\mathcal{X} := \{0, 1\}^n$ be a finite binary search space with dimension n. The univariate model in generation $t \in \mathbb{N}$ is represented by a vector $p^{(t)} := (p_1^{(t)}, \ldots, p_n^{(t)}) \in [0, 1]^n$, where each $p_i^{(t)}$ is called a *marginal*. Let $X_1^{(t)}, \ldots, X_n^{(t)}$ be n independent Bernoulli random variables with success probabilities $p_1^{(t)}, \ldots, p_n^{(t)}$. Furthermore, let $X_{i:j}^{(t)} := \sum_{k=i}^{j} X_k^{(t)}$

be the number of ones sampled from $p_{i:j}^{(t)} := (p_i^{(t)}, \ldots, p_j^{(t)})$ for all $1 \leq i \leq j \leq n$. Each individual (or bitstring) is denoted as $x = (x_1, \ldots, x_n) \in \mathcal{X}$. We aim at maximising an objective function $f : \mathcal{X} \to \mathbb{R}$. We are primarily interested in the optimisation time of these algorithms, so tools to analyse runtime are of importance. We will make use of the level-based theorem [3].

2.1 Two Problems

We consider the two pseudo-Boolean functions: LeadingOnes and BinVal, which are widely used theoretical benchmark problems in runtime analyses of EDAs [4,5,14]. The former aims at maximising the number of leading ones, while the latter tries to maximise the binary value of the bitstring. The global optimum for both functions are the all-ones bitstring. Furthermore, BinVal is an extreme linear function, where the fitness-contribution of the bits decreases exponentially with the bit-position. Droste [5] showed that among all linear functions, BinVal is difficult for the cGA. Given a bitstring $x = (x_1, \ldots, x_n) \in \mathcal{X}$, the two functions are formally defined as follows:

Definition 1. $\textsc{LeadingOnes}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$.

Definition 2. $\textsc{BinVal}(x) := \sum_{i=1}^{n} 2^{n-i} x_i$.

2.2 Population-Based Incremental Learning

The PBIL algorithm maintains a univariate model over generations. The probability of a bitstring $x = (x_1, \ldots, x_n)$ sampled from the current model $p^{(t)}$ is given by

$$\Pr\left(x \mid p^{(t)}\right) = \prod_{i=1}^{n} \left(p_i^{(t)}\right)^{x_i} \left(1 - p_i^{(t)}\right)^{1-x_i}. \tag{1}$$

Let $p^{(0)} := (1/2, \ldots, 1/2)$ be the initial model. The algorithm in generation t samples a population of λ individuals, denoted as $P^{(t)} := \{x^{(1)}, x^{(2)}, \ldots, x^{(\lambda)}\}$, which are sorted in descending order according to fitness. The μ fittest individuals are then selected to derive the next model $p^{(t+1)}$ using the component-wise formula $p_i^{(t+1)} := (1-\eta)\, p_i^{(t)} + (\eta/\mu) \sum_{j=1}^{\mu} x_i^{(j)}$ for all $i \in \{1, 2, \ldots, n\}$, where $x_i^{(j)}$ is the i-th bit of the j-th individual in the sorted population, and $\eta \in (0, 1]$ is the smoothing parameter (sometimes known as the learning rate). The ratio $\gamma_0 := \mu/\lambda \in (0, 1)$ is called the selective pressure of the algorithm. Univariate EDAs often employ margins to avoid the marginals to fix at either 0 or 1. In particular, the marginals are usually restricted to the interval $[1/n, 1 - 1/n]$ after being updated, where the quantities $1/n$ and $1 - 1/n$ are called the lower and upper borders, respectively. The algorithm is called the PBIL with margins. Algorithm 1 gives a full description of the PBIL (with margins).

Algorithm 1. PBIL with margins

$t \leftarrow 0$; $p^{(t)} \leftarrow (1/2, 1/2, \ldots, 1/2)$
repeat
  for $j = 1, 2, \ldots, \lambda$ **do**
    sample an offspring $x^{(j)} \sim \Pr(\cdot \mid p^{(t)})$ as defined in (1)
    evaluate the fitness $f(x^{(j)})$
  sort $P^{(t)} \leftarrow \{x^{(1)}, x^{(2)}, \ldots, x^{(\lambda)}\}$ such that $f(x^{(1)}) \geq f(x^{(2)}) \geq \ldots \geq f(x^{(\lambda)})$
  for $i = 1, 2, \ldots, n$ **do**
    $p_i^{(t+1)} \leftarrow \max\left\{1/n, \min\left\{1 - 1/n, (1-\eta)\, p_i^{(t)} + (\eta/\mu) \sum_{j=1}^{\mu} x_i^{(j)}\right\}\right\}$
  $t \leftarrow t + 1$
until *termination condition is fulfilled*

Algorithm 2. Non-elitist population-based algorithm

$t \leftarrow 0$; create initial population $P^{(t)}$
repeat
  for $i = 1, \ldots, \lambda$ **do**
    sample $P_i^{(t+1)} \sim \mathcal{D}(P^{(t)})$
  $t \leftarrow t + 1$
until *termination condition is fulfilled*

2.3 Level-Based Analysis

Introduced in [3], the level-based theorem is a general tool that provides upper bounds on the expected optimisation time of many non-elitist population-based algorithms on a wide range of optimisation problems [3,4,8]. The theorem assumes that the algorithm to be analysed can be described in the form of Algorithm 2, which maintains a population $P^{(t)} \in \mathcal{X}^\lambda$, where $\mathcal{X}^\lambda$ is the space of all populations with size λ. The theorem is general since it never assumes specific fitness functions, selection mechanisms, or generic operators like mutation and crossover. Furthermore, the theorem assumes that the search space $\mathcal{X}$ can be partitioned into m disjoint subsets $A_1, \ldots, A_m$, which we call *levels*, and the last level A_m consists of all global optima of the objective function. The theorem is formally stated in Theorem 1 [3]. We will use the notation $[n] := \{1, 2, \ldots, n\}$ and $A_{\geq j} := \cup_{k=j}^{m} A_k$.

Theorem 1 (Level-Based Theorem). *Given a partition $(A_i)_{i \in [m]}$ of $\mathcal{X}$, define $T := \min\{t\lambda \mid |P^{(t)} \cap A_m| > 0\}$, where for all $t \in \mathbb{N}$, $P^{(t)} \in \mathcal{X}^\lambda$ is the population of Algorithm 2 in generation t. Denote $y \sim \mathcal{D}(P^{(t)})$. If there exist $z_1, \ldots, z_{m-1}, \delta \in (0, 1]$, and $\gamma_0 \in (0, 1)$ such that for any population $P^{(t)} \in \mathcal{X}^\lambda$,*

(G1) *for each level $j \in [m-1]$, if $|P^{(t)} \cap A_{\geq j}| \geq \gamma_0 \lambda$ then $\Pr(y \in A_{\geq j+1}) \geq z_j$.*
(G2) *for each level $j \in [m-2]$ and all $\gamma \in (0, \gamma_0]$, if $|P^{(t)} \cap A_{\geq j}| \geq \gamma_0 \lambda$ and $|P^{(t)} \cap A_{\geq j+1}| \geq \gamma \lambda$ then $\Pr(y \in A_{\geq j+1}) \geq (1+\delta)\gamma$.*

(***G3***) *and the population size* $\lambda \in \mathbb{N}$ *satisfies* $\lambda \geq \left(\frac{4}{\gamma_0 \delta^2}\right) \ln\left(\frac{128m}{z_* \delta^2}\right)$ *where* $z_* := \min_{j \in [m-1]}\{z_j\}$, *then*

$$\mathbb{E}[T] \leq \left(\frac{8}{\delta^2}\right) \sum_{j=1}^{m-1} \left[\lambda \ln\left(\frac{6\delta\lambda}{4 + z_j \delta\lambda}\right) + \frac{1}{z_j}\right].$$

Algorithm 2 assumes a mapping $\mathcal{D}$ from the space of populations $\mathcal{X}^\lambda$ to the space of probability distributions over the search space. The mapping $\mathcal{D}$ is often said to depend on the current population only [3]; however, it is unnecessarily always the case, especially for the PBIL with a sufficiently large offspring population size λ. The rationale behind this is that in each generation the PBIL draws λ samples from the current model $p^{(t)}$, that correspond to λ individuals in the current population, and if the number of samples λ is sufficiently large, it is highly likely that the empirical distributions for all positions among the entire population cannot deviate too far from the true distributions, i.e. marginals $p_i^{(t)}$. Moreover, the theorem relies on three conditions (G1), (G2) and (G3); thus, as long as these three can be fully verified, the PBIL, whose model is constructed from the current population $P^{(t)}$ in addition to the current model $p^{(t)}$, is still eligible to the level-based analysis.

2.4 Other Tools

In addition to the level-based theorem, we also make use of some other mathematical results. First of all is the Dvoretzky–Kiefer–Wolfowitz inequality [9], which provides an estimate on how close an empirical distribution function will be to the true distribution from which the samples are drawn. The following theorem follows by replacing $\varepsilon = \varepsilon' \sqrt{\lambda}$ into [9, Corollary 1].

Theorem 2 (DKW Inequality). *Let* $X_1, \ldots, X_\lambda$ *be* λ *i.i.d. real-valued random variables with cumulative distribution function* F. *Let* $\hat{F}_\lambda$ *be the empirical distribution function which is defined by* $\hat{F}_\lambda(x) := (1/\lambda) \sum_{i=1}^{\lambda} \mathbb{1}_{\{X_i \leq x\}}$. *For any* $\lambda \in \mathbb{N}$ *and* $\varepsilon > 0$, *we always have*

$$\Pr\left(\sup_{x \in \mathbb{R}} \left|\hat{F}_\lambda(x) - F(x)\right| > \varepsilon\right) \leq 2e^{-2\lambda\varepsilon^2}.$$

Furthermore, properties of majorisation between two vectors are also exploited. The concept is formally defined in Definition 3 [6], followed by its important property (in Lemma 1) that we use intensively throughout the paper.

Definition 3. Given vectors $p^{(1)} := (p_1^{(1)}, \ldots, p_n^{(1)})$ and $p^{(2)} := (p_1^{(2)}, \ldots, p_n^{(2)})$, where $p_1^{(1)} \geq p_2^{(1)} \geq \ldots \geq p_n^{(1)}$ and similarly for the $p_i^{(2)}$s. Vector $p^{(1)}$ is said to *majorise* vector $p^{(2)}$, in symbols $p^{(1)} \succ p^{(2)}$, if $p_1^{(1)} \geq p_1^{(2)}, \ldots, \sum_{i=1}^{n-1} p_i^{(1)} \geq \sum_{i=1}^{n-1} p_i^{(2)}$ and $\sum_{i=1}^{n} p_i^{(1)} = \sum_{i=1}^{n} p_i^{(2)}$.

Lemma 1 ([2]). *Let $X_1, \ldots, X_n$ be n independent Bernoulli random variables with success probabilities $p_1, \ldots, p_n$, respectively. Denote $p := (p_1, p_2, \ldots, p_n)$; let $S(p) := \sum_{i=1}^{n} X_i$ and $D_\lambda := \{p : p_i \in [0,1],\ i \in [n],\ \sum_{i=1}^{n} p_i = \lambda\}$. For two vectors $p^{(1)}, p^{(2)} \in D_\lambda$, if $p^{(1)} \prec p^{(2)}$ then $\Pr\left(S(p^{(1)}) = n\right) \geq \Pr\left(S(p^{(2)}) = n\right)$.*

Lemma 2 (Main lemma). *Let $p^{(1)}$ and $p^{(2)} \in D_\lambda$ be two vectors as defined in Lemma 1, where all components in $p^{(\cdot)}$ are arranged in descending order. Let $z^{(1)} := (z_1^{(1)}, \ldots, z_n^{(1)})$ where each $z_i^{(1)} := (1-\eta)\, p_i^{(1)} + \eta$, and $z^{(2)} := (z_1^{(2)}, \ldots, z_n^{(2)})$, where each $z_i^{(2)} := (1-\eta)\, p_i^{(2)} + \eta$ for any constant $\eta \in (0,1]$. If $p^{(2)} \succ p^{(1)}$, then $z^{(2)} \succ z^{(1)}$.*

Proof. For all $j \in [n-1]$, it holds that $\sum_{i=1}^{j} z_i^{(2)} \geq \sum_{i=1}^{j} z_i^{(1)}$ since $\sum_{i=1}^{j} p_i^{(2)} \geq \sum_{i=1}^{j} p_i^{(1)}$. Furthermore, if $j = n$, then $\sum_{i=1}^{n} z_i^{(2)} = \sum_{i=1}^{n} z_i^{(1)}$ due to $\sum_{i=1}^{n} p_i^{(2)} = \sum_{i=1}^{n} p_i^{(1)}$. By Definition 3, $z^{(2)} \succ z^{(1)}$. □

3 Runtime Analysis of the PBIL on LeadingOnes

We now show how to apply the level-based theorem to analyse the runtime of the PBIL. We use a *canonical partition* of the search space, where each subset A_j contains bitstrings with exactly j leading ones.

$$A_j := \{x \in \{0,1\}^n \mid \textsc{LeadingOnes}(x) = j\}. \tag{2}$$

Conditions (G1) and (G2) of Theorem 1 assume that there are at least $\gamma_0 \lambda$ individuals in levels $A_{\geq j}$ in generation t. Recall $\gamma_0 := \mu/\lambda$. This implies that the first j bits among the μ fittest individuals are all ones. Denote $\hat{p}_i^{(t)} := (1/\lambda) \sum_{j=1}^{\lambda} x_i^{(j)}$ as the frequencies of ones at position i in the current population. We first show that under the assumption of the two conditions of Theorem 1 and with a population size $\lambda = \Omega(\log n)$, the first j marginals cannot be too close to the lower border $1/n$ with probability at least $1 - n^{-\Omega(1)}$.

Lemma 3. *If $|P^{(t)} \cap A_{\geq j}| \geq \gamma_0 \lambda$ and $\lambda \geq c((1+1/\varepsilon)/\gamma_0)^2 \ln(n)$ for any constants c, $\varepsilon > 0$ and $\gamma_0 \in (0,1)$, then it holds with probability at least $1 - 2n^{-2c}$ that $p_i^{(t)} \geq \gamma_0/(1+\varepsilon)$ for all $i \in [j]$.*

Proof. Consider an arbitrary bit $i \in [j]$. Let Q_i be the number of ones sampled at position i in the current population, and the corresponding empirical distribution function of the number of zeros is $F_\lambda(0) = (1/\lambda) \sum_{j=1}^{\lambda} \mathbb{1}_{\{x_i^{(j)} \leq 0\}} = (\lambda - Q_i)/\lambda = 1 - \hat{p}_i^{(t)}$, and the true distribution function is $F(0) = 1 - p_i^{(t)}$. The DKW inequality (see Theorem 2) yields that $\Pr(\hat{p}_i^{(t)} - p_i^{(t)} > \phi) \leq \Pr(|\hat{p}_i^{(t)} - p_i^{(t)}| > \phi) \leq 2e^{-2\lambda\phi^2}$ for all $\phi > 0$. Therefore, with probability at least $1 - 2e^{-2\lambda\phi^2}$ we have $\hat{p}_i^{(t)} - p_i^{(t)} \leq \phi$ and, thus, $p_i^{(t)} \geq \hat{p}_i^{(t)} - \phi \geq \gamma_0 - \phi$ since $\hat{p}_i^{(t)} \geq \gamma_0 \lambda/\lambda = \gamma_0$ due to $|P^{(t)} \cap A_{\geq j}| \geq \gamma_0 \lambda$. We then choose $\phi \leq \varepsilon\gamma_0/(1+\varepsilon)$ for some constant $\varepsilon > 0$ and $\lambda \geq c((1+1/\varepsilon)/\gamma_0)^2 \ln(n)$. Putting everything together, it holds that $p_i^{(t)} \geq \gamma_0(1 - \varepsilon/(1+\varepsilon)) = \gamma_0/(1+\varepsilon)$ with probability at least $1 - 2n^{-2c}$. □

Given the μ top individuals having at least j leading ones, we now estimate the probability of sampling j leading ones from the current model $p^{(t)}$.

Lemma 4. *For any non-empty subset $I \subseteq [n]$, define $C_I := \left\{x \in \{0,1\}^n \mid \prod_{i\in I} x_i = 1\right\}$. If $|P^{(t)} \cap C_I| \geq \gamma_0\lambda$ and $\lambda \geq c((1+1/\varepsilon)/\gamma_0)^2 \ln(n)$ for any constants $\varepsilon > 0$, $\gamma_0 \in (0,1)$, then it holds with probability at least $1-2n^{-2c}$ that $q^{(t)} := \prod_{i\in I} p_i^{(t)} \geq \gamma_0/(1+\varepsilon)$.*

Proof. We prove the statement using the DKW inequality (see Theorem 2). Let $m = |I|$. Given an offspring sample $Y \sim p^{(t)}$ from the current model, let $Y_I := \sum_{i\in I} Y_i$ be the number of one-bits in bit-positions I. By the assumption $|P^{(t)} \cap C_I| \geq \gamma_0\lambda$ on the current population, the empirical distribution function of Y_I must satisfy $\hat{F}_\lambda(m-1) = \frac{1}{\lambda}\sum_{i=1}^{\lambda} \mathbb{1}_{\{Y_{I,i}\leq m-1\}} \leq 1-\hat{q}^{(t)}$, where $\hat{q}^{(t)} \geq \gamma_0$ is the fraction of individuals in the current population with j leading ones, and the true distribution function satisfies $F(m-1) = 1-q^{(t)}$. The DKW inequality yields that $\Pr(\hat{q}^{(t)} - q^{(t)} > \phi) \leq \Pr(|\hat{q}^{(t)} - q^{(t)}| > \phi) \leq 2e^{-2\lambda\phi^2}$ for all $\phi > 0$. Therefore, with probability at least $1-2e^{-2\lambda\phi^2}$ it holds $\hat{q}^{(t)} - q^{(t)} \leq \phi$ and, thus, $q^{(t)} \geq \hat{q}^{(t)} - \phi \geq \gamma_0 - \phi$. Choosing $\phi := \varepsilon\gamma_0/(1+\varepsilon)$, we get $q^{(t)} \geq \gamma_0(1-\varepsilon/(1+\varepsilon)) = \gamma_0/(1+\varepsilon)$ with probability at least $1-2e^{-2\phi^2\lambda} \geq 1-2n^{-2c}$. □

Given the current level is j, we speak of a *success* if the first j marginals never drop below $\gamma_0/(1+\varepsilon)$; otherwise, we speak of a *failure*. If there are no failures at all, let us assume that $\mathcal{O}\left(n\log\lambda + n^2/\lambda\right)$ is an upper bound on the expected number of generations of the PBIL on LeadingOnes. The following lemma shows that this is also the expected optimisation time of the PBIL on LeadingOnes.

Lemma 5. *If the expected number of generations required by the* PBIL *to optimise* LeadingOnes *in case of no failure is at most $t^* \in \mathcal{O}\left(n\log\lambda + n^2/\lambda\right)$ regardless of the initial probability vector of the* PBIL*, the expected number of generations of the* PBIL *on* LeadingOnes *is at most $4(1+o(1))t^*$.*

Proof. From the point when the algorithm starts, we divide the time into identical phases, each lasting t^* generations. Let $\mathcal{E}_i$ denote the event that the i-th interval is a failure for $i \in \mathbb{N}$. According to Lemma 3, $\Pr(\mathcal{E}_i) \leq 2n^{-2c}\ \mathcal{O}(n\log\lambda + n^2/\lambda) = \mathcal{O}(n^{-c'+2})$ by union bound for another constant $c' > 0$ when the population is of at most exponential size, that is $\lambda \leq 2^{\alpha n}$ where $\alpha > 0$ is a constant with respect to n, and the constant c large enough such that $c' > 2$, and $\Pr\left(\overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) \geq 1 - \Pr(\mathcal{E}_1) - \Pr(\mathcal{E}_2) \geq 1 - \mathcal{O}(n^{-c'+2})$ by union bound. Let T be the number of generations performed by the algorithm until a global optimum is found for the first time. We know that $\mathbb{E}\left[T \mid \wedge_{i\in\mathbb{N}} \overline{\mathcal{E}}_i\right] \leq t^*$, and $\Pr\left(T \leq 2t^* \mid \wedge_{i\in\mathbb{N}} \overline{\mathcal{E}}_i\right) \geq 1/2$ since $\Pr\left(T \geq 2t^* \mid \wedge_{i\in\mathbb{N}} \overline{\mathcal{E}}_i\right) \leq 1/2$ by Markov's inequality [10]. We now consider each pair of two consecutive phases. If there is a failure in a pair of phases, we wait until that pair has passed by and then

repeat the arguments above as if no failure has ever happened. It holds that

$$\begin{aligned}\mathbb{E}\left[T \,|\, \overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right] &\leq 2t^* \Pr\left(T \leq 2t^* \mid \overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) + (2t^* + \mathbb{E}\left[T\right]) \Pr\left(T \geq 2t^* \mid \overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) \\ &= 2t^* + \Pr\left(T \geq 2t^* \mid \overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) \mathbb{E}\left[T\right] \\ &\leq 2t^* + (1/2)\mathbb{E}\left[T\right]\end{aligned}$$

since $\Pr\left(T \leq 2t^* \mid \overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) \geq \Pr\left(T \leq 2t^* \mid \wedge_{i\in\mathbb{N}} \overline{\mathcal{E}}_i\right) \geq 1/2$. Substituting the result into the following yields

$$\begin{aligned}\mathbb{E}\left[T\right] &= \Pr\left(\overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) \mathbb{E}\left[T \mid \overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right] + \Pr\left(\mathcal{E}_1 \vee \mathcal{E}_2\right) (2t^* + \mathbb{E}\left[T\right]) \\ &\leq \Pr\left(\overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) (2t^* + (1/2)\mathbb{E}\left[T\right]) + \Pr\left(\mathcal{E}_1 \vee \mathcal{E}_2\right) (2t^* + \mathbb{E}\left[T\right]) \\ &= 2t^* + ((1/2)\Pr\left(\overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) + \Pr\left(\mathcal{E}_1 \vee \mathcal{E}_2\right))\mathbb{E}\left[T\right] \\ &= 2t^* + \mathbb{E}\left[T\right] - (1/2)\Pr\left(\overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right)\mathbb{E}\left[T\right].\end{aligned}$$

Thus, $\mathbb{E}\left[T\right] \leq 4t^*/\Pr\left(\overline{\mathcal{E}}_1 \wedge \overline{\mathcal{E}}_2\right) = 4t^*\left(1 + o(1)\right) = 4(1 + o(1))t^*$. □

By the result of Lemma 5, the phase-based analysis that is exploited until there is a pair with no failure only leads to a multiplicative constant in the expectation. We need to calculate the value of t^* that will also asymptotically be the overall expected number of generations of the PBIL on LeadingOnes. We now give our runtime bound for the PBIL on LeadingOnes with sufficiently large population λ. The proof is very straightforward compared to that in [14]. The floor and ceiling functions of $x \in \mathbb{R}$ are $\lfloor x \rfloor$ and $\lceil x \rceil$, respectively.

Theorem 3. *The* PBIL *with margins and offspring population size* $\lambda \geq c \log n$ *for a sufficiently large constant* $c > 0$*, parent population size* $\mu = \gamma_0\lambda$ *for any constant* γ_0 *satisfying* $\gamma_0 \leq \eta^{\lceil \xi \rceil + 1}/((1+\delta)e)$ *where* $\xi = \ln(p_0)/(p_0 - 1)$ *and* $p_0 := \gamma_0/(1+\varepsilon)$ *for any positive constants* δ*,* ε *and smoothing parameter* $\eta \in (0, 1]$*, has expected optimisation time* $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$ *on* LeadingOnes.

Proof. We strictly follow the procedure recommended in [3].
Step 1: Recall that we use the canonical partition, defined in (2), in which each subset A_j contains individuals with exactly j leading ones. There are a total of $m = n + 1$ levels ranging from A_0 to A_n.
Step 2: Given $|P^{(t)} \cap A_{\geq j}| \geq \gamma_0\lambda = \mu$ and $|P^{(t)} \cap A_{\geq j+1}| \geq \gamma\lambda$, we prove that the probability of sampling an offspring in $A_{\geq j+1}$ in generation $t + 1$ is lower bounded by $(1 + \delta)\gamma$ for some constant $\delta > 0$.

Lemma 1 asserts that if we can find a vector $z^{(t)} = (z_1^{(t)}, \ldots, z_j^{(t)})$ that majorises $p_{1:j}^{(t)}$, then the probability of obtaining j successes from a Poisson-binomial distribution with parameters j and $p_{1:j}^{(t)}$ is lower bounded by the same distribution with parameters j and $z^{(t)}$. Following [14], we compare $X_1^{(t)}, \ldots, X_j^{(t)}$ with another sequence of independent Bernoulli random variables $Z_1^{(t)}, \ldots, Z_j^{(t)}$ with success probabilities $z_1^{(t)}, \ldots, z_j^{(t)}$. Note that $Z^{(t)} := \sum_{k=1}^{j} Z_k^{(t)}$. Define $m := \lfloor (\sum_{i=1}^{j} p_i^{(t)} - jp_0)/(1 - \frac{1}{n} - p_0) \rfloor$ where $p_0 := \frac{\gamma_0}{1+\varepsilon}$,

and let $Z_1^{(t)}, \ldots, Z_m^{(t)}$ all have success probability $z_1^{(t)} = \ldots = z_m^{(t)} = 1 - \frac{1}{n}$, $Z_{m+2}^{(t)}, \ldots, Z_j^{(t)}$ get p_0 and possibly a random variable $Z_{m+1}^{(t)}$ takes intermediate value $[p_0, 1 - \frac{1}{n}]$ to guarantee $\sum_{i=1}^{j} p_i^{(t)} = \sum_{i=1}^{j} z_i^{(t)}$.

Since $\sum_{i=1}^{j} p_i^{(t)} \geq j \cdot (\prod_{i=1}^{j} p_i^{(t)})^{1/j} \geq j \cdot p_0^{1/j}$ by the Arithmetic Mean-Geometric Mean inequality (see Lemma 7 in the Appendix) and Lemma 4, we get $m \geq \lfloor j(p_0^{1/j} - p_0)/\left(1 - \frac{1}{n} - p_0\right) \rfloor$. Let us consider the following function:

$$g(j) = j \cdot \frac{p_0^{1/j} - p_0}{1 - p_0} - j = j \cdot \frac{p_0^{1/j} - 1}{1 - p_0}.$$

This function has a horizontal asymptote at $y = -\xi$, where $\xi := \frac{\ln p_0}{p_0 - 1}$ (see calculation in the Appendix). Thus, $m \geq j - \lceil \xi \rceil$ for all $j \geq 0$.

Note that we have just performed all calculations on the current model in generation t. The PBIL then updates the current model $p^{(t)}$ to obtain $p^{(t+1)}$ using the component-wise formula $p_i^{(t+1)} = (1 - \eta) p_i^{(t)} + \frac{\eta}{\mu} \sum_{k=1}^{\mu} x_i^{(k)}$. For all $i \in [j]$, we know that $\sum_{k=1}^{\mu} x_i^{(k)} = \mu$ due to the assumption of condition (G2). After the model is updated, we obtain

- $z_i^{(t+1)} = 1 - \frac{1}{n}$ for all $i \leq j - \lceil \xi \rceil$,
- $z_i^{(t+1)} \geq (1 - \eta)\ p_0 + \eta \geq \eta$ for all $j - \lceil \xi \rceil < i \leq j$, and
- $p_{j+1}^{(t+1)} \geq (1 - \eta)\ p_{j+1}^{(t)} + \eta \frac{\gamma}{\gamma_0} \geq \eta \frac{\gamma}{\gamma_0}$ due to $\sum_{k=1}^{\mu} x_{j+1}^{(k)} = \gamma\lambda$.

Let us denote $z_i^{(t+1)} = (1-\eta) z_i^{(t)} + \eta$. Lemmas 1 and 2 assert that $z^{(t+1)}$ majorises $p_{i:j}^{(t+1)}$, and $\Pr(X_{1:j}^{(t+1)} = j) \geq \Pr(Z^{(t+1)} = j)$. In words, the probability of sampling an offspring in $A_{\geq j}$ in generation $t+1$ is lower bounded by the probability of obtaining j successes from a Poisson-binomial distribution with parameters j and $z^{(t+1)}$. More precisely, at generation $t+1$,

$$\begin{aligned}\Pr(X_{1:j+1}^{(t+1)} = j+1) &\geq \Pr(X_{1:j}^{(t+1)} = j) \cdot \Pr(X_{j+1}^{(t+1)} = 1) \\ &\geq \Pr(Z^{(t+1)} = j) \cdot p_{j+1}^{(t+1)} \geq (1 - 1/n)^{j - \lceil \xi \rceil} \eta^{\lceil \xi \rceil + 1} \gamma/\gamma_0 \geq (1+\delta)\gamma,\end{aligned}$$

where $\left(1 - \frac{1}{n}\right)^{j - \lceil \xi \rceil} \geq \frac{1}{e}$ and $\gamma_0 \leq \frac{\eta^{\lceil \xi \rceil + 1}}{e(1+\delta)}$ for any constant $\delta > 0$. Thus, condition (G2) of Theorem 1 is verified.

Step 3: Given that $|P^{(t)} \cap A_{\geq j}| \geq \gamma_0 \lambda$, we aim at showing that the probability of sampling an offspring in $A_{\geq j+1}$ in generation $t+1$ is at least z_j. Note in particular that Lemma 4 yields $\Pr(X_{1:j}^{(t+1)} = j) \geq \frac{\gamma_0}{1+\varepsilon}$. The probability of sampling an offspring in $A_{\geq j+1}$ in generation $t+1$ is lower bounded by

$$\Pr(X_{1:j}^{(t+1)} = j) \cdot \Pr(X_{j+1}^{(t+1)} = 1) \geq \frac{\gamma_0}{1+\varepsilon} \cdot \frac{1}{n} =: z_j.$$

where $\Pr(X_{j+1}^{(t+1)} = 1) = p_{j+1}^{(t+1)} \geq \frac{1}{n}$. Therefore, condition (G1) of Theorem 1 is satisfied with $z_j = z_* = \frac{\gamma_0}{(1+\varepsilon)n}$.

Step 4: Condition (G3) of Theorem 1 requires a population size $\lambda = \Omega(\log n)$. This bound matches with the condition on $\lambda \geq c \log n$ for some sufficiently large constant $c > 0$ from the previous lemmas. Overall, $\lambda = \Omega(\log n)$.

Step 5: When $z_j = \frac{\gamma_0}{(1+\varepsilon)n}$ where $\gamma_0 \leq \frac{\eta^{\lceil \xi \rceil + 1}}{(1+\delta)e}$ and $\lambda \geq c \log n$ for some constants $\varepsilon > 0$, $\eta \in (0, 1]$ and sufficiently large $c > 0$, all conditions of Theorem 1 are verified. Using that $\ln\left(\frac{6\delta\lambda}{4+\delta\lambda z_j}\right) < \ln\left(\frac{3\delta\lambda}{2}\right)$ an upper bound on the expected optimisation time of the PBIL on LeadingOnes is guaranteed as follows.

$$\left(\frac{8}{\delta^2}\right) \sum_{j=0}^{n-1} \left[\lambda \ln\left(\frac{3\delta\lambda}{2}\right) + \frac{1}{z_j}\right] < \frac{8}{\delta^2} n\lambda \log \lambda + \frac{8(1+\varepsilon)}{\delta^2 \gamma_0} n^2 + o\left(n^2\right) \in \mathcal{O}\left(n\lambda \log \lambda + n^2\right).$$

Hence, the expected number of generations t^* is $\mathcal{O}\left(n \log \lambda + \frac{n^2}{\lambda}\right)$ for a sufficiently large λ in the case of no failure and, thus, meets the assumption in Lemma 5. The expected optimisation time of the PBIL on LeadingOnes is still asymptotically $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$. This completes the proof. □

Our improved upper bound of $\mathcal{O}\left(n^2\right)$ on the optimisation time of the PBIL with population size $\lambda = \Theta(\log n)$ on LeadingOnes is significantly better than the previous bound $\mathcal{O}\left(n^{2+\varepsilon}\right)$ from [14]. Our result is not only stronger, but the proof is much simpler as most of the complexities of the population dynamics of the algorithm is handled by Theorem 1 [3]. Furthermore, we also provide specific values for the multiplicative constants, i.e. $\frac{32}{\delta^2}$ and $\frac{32(1+\varepsilon)}{\delta^2\gamma_0}$ for the terms $n\lambda \log \lambda$ and n^2, respectively (see Step 5 in Theorem 3). Moreover, the result also matches the runtime bound of the UMDA on LeadingOnes for a small population $\lambda = \Theta(\log n)$ [4].

Note that Theorem 3 requires some condition on the selective pressure, that is $\gamma_0 \leq \frac{\eta^{\lceil \xi \rceil + 1}}{(1+\delta)e}$ where $\xi = \frac{\ln p_0}{p_0 - 1}$ and $p_0 := \frac{\gamma_0}{1+\varepsilon}$ for any positive constants δ, ε and smoothing parameter $\eta \in (0, 1]$. Although for practical applications, we have to address these constraints to find a suitable set of values for γ_0, this result here tells us that there exists some settings for the PBIL such that it can optimise LeadingOnes within $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$ time in expectation.

4 Runtime Analysis of the PBIL on BinVal

We first partition the search space into non-empty disjoint subsets $A_0, \ldots, A_n$.

Lemma 6. *Let us define the levels as* $A_j := \{x \in \{0,1\}^n \mid \sum_{i=1}^{j} 2^{n-i} \leq \textsc{BinVal}(x) < \sum_{i=1}^{j+1} 2^{n-i}\}$, *for* $j \in [n] \cup \{0\}$, *where* $\sum_{i=1}^{0} 2^{n-i} = 0$. *If a bit-string* x *has exactly* j *leading ones, i.e.* $\textsc{LeadingOnes}(x) = j$, *then* $x \in A_j$.

Proof. Consider a bitstring $x = 1^j 0\{0,1\}^{n-j-1}$. The fitness contribution of the first j leading ones to BinVal(x) is $\sum_{i=1}^{j} 2^{n-i}$. The $(j+1)$-th bit has no contribution, while that of the last $n-j-1$ bits ranges from zero to $\sum_{i=j+2}^{n} 2^{n-i} = \sum_{i=0}^{n-j-2} 2^i = 2^{n-j-1} - 1$. So overall, $\sum_{i=1}^{j} 2^{n-i} \leq \text{BinVal}(x) \leq \sum_{i=1}^{j+1} 2^{n-i} - 1 < \sum_{i=1}^{j+1} 2^{n-i}$. Hence, the bitstring x belongs to level A_j. □

In both problems, all that matters to determine the level of a bitstring is the position of the first 0-bit when scanning from the most significant to the least significant bits. Now consider two bitstrings in the same level for BinVal, their rankings after the population is sorted are also determined by some other less significant bits; however, the proof of Theorem 3 never takes these bits into account. Thus, the following corollary yields the first upper bound on the expected optimisation time of the PBIL and the UMDA (when $\eta = 1$) for BinVal.

Corollary 1. *The* PBIL *with margins and offspring population size $\lambda \geq c \log n$ for a sufficiently large constant $c > 0$, parent population size $\mu = \gamma_0 \lambda$ for any constant γ_0 satisfying $\gamma_0 \leq \eta^{\lceil \xi \rceil + 1}/((1+\delta)e)$ where $\xi = \ln(p_0)/(p_0 - 1)$ and $p_0 := \gamma_0/(1+\varepsilon)$ for any positive constants δ, ε and smoothing parameter $\eta \in (0, 1]$, has expected optimisation time $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$ on* BinVal.

5 Conclusions

Runtime analyses of EDAs are scarce. Motivated by this, we have derived an upper bound of $\mathcal{O}\left(n\lambda \log \lambda + n^2\right)$ on the expected optimisation time of the PBIL on both LeadingOnes and BinVal for a population size $\lambda = \Omega(\log n)$. The result improves upon the previously best-known bound $\mathcal{O}\left(n^{2+\varepsilon}\right)$ from [14], and requires a much smaller population size $\lambda = \Omega(\log n)$, and uses relatively straightforward arguments. We also presents the first upper bound on the expected optimisation time of the PBIL on BinVal.

Furthermore, our analysis demonstrates that the level-based theorem can yield runtime bounds for EDAs whose models are updated using information gathered from the current and previous generations. An additional aspect of our analysis is the use of the DKW inequality to bound the true distribution by the empirical population sample when the number of samples is large enough. We expect these arguments will lead to new results in runtime analysis of evolutionary algorithms.

Appendix

Lemma 7 (AM-GM Inequality [12]). *Let $x_1, \ldots, x_n$ be n non-negative real numbers. It always holds that*

$$\frac{x_1 + x_2 + \cdots + x_n}{n} \geq \sqrt[n]{x_1 \cdot x_2 \cdots x_n},$$

and equality holds if and only if $x_1 = x_2 = \cdots = x_n$.

Proof (of horizontal asymptote). The function can be rewritten as $g(j) = \frac{1}{1-p_0} \cdot \frac{p_0^{1/j}-1}{1/j}$. Denote $t := 1/j$, we obtain $g(t) = \frac{1}{1-p_0} \cdot \frac{p_0^t-1}{t}$. Applying L'Hôpital's rule yields:

$$\lim_{j\to+\infty} g(j) = \lim_{t\to 0^+} g(t) = \frac{\lim_{t\to 0^+}(p_0^t \ln p_0)}{1-p_0} = \frac{\ln p_0}{1-p_0} = -\frac{\ln p_0}{p_0-1}.$$

□

References

1. Baluja, S.: Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon University (1994)
2. Boland, P.J., Proschan, F.: The reliability of k out of n systems. Ann. Probab. **11**(3), 760–764 (1983)
3. Corus, D., Dang, D.C., Eremeev, A.V., Lehre, P.K.: Level-based analysis of genetic algorithms and other search processes. IEEE Trans. Evol. Comput. **PP**(99), 1 (2017)
4. Dang, D.C., Lehre, P.K.: Simplified runtime analysis of estimation of distribution algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, pp. 513–518 (2015)
5. Droste, S.: A rigorous analysis of the compact genetic algorithm for linear functions. Nat. Comput. **5**(3), 257–283 (2006)
6. Gleser, L.J.: On the distribution of the number of successes in independent trials. Ann. Probab. **3**(1), 182–188 (1975)
7. Krejca, M.S., Witt, C.: Lower bounds on the run time of the univariate marginal distribution algorithm on OneMax. In: Proceedings of the Foundation of Genetic Algorithms, FOGA 2017, pp. 65–79 (2017)
8. Lehre, P.K., Nguyen, P.T.H.: Improved runtime bounds for the univariate marginal distribution algorithm via anti-concentration. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 1383–1390 (2017)
9. Massart, P.: The tight constant in the Dvoretzky-Kiefer-Wolfowitz inequality. Ann. Probab. **18**(3), 1269–1283 (1990)
10. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
11. Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 178–187. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61723-X_982
12. Steele, J.M.: The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities. Cambridge University Press, Cambridge (2004)
13. Witt, C.: Upper bounds on the runtime of the univariate marginal distribution algorithm on onemax. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 1415–1422 (2017)
14. Wu, Z., Kolonko, M., Möhring, R.H.: Stochastic runtime analysis of the cross-entropy algorithm. IEEE Trans. Evol. Comput. **21**(4), 616–628 (2017)

Precise Runtime Analysis for Plateaus

Denis Antipov[1(✉)] and Benjamin Doerr[2]

[1] ITMO University, 49 Kronverkskiy prosp., 197101 Saint-Petersburg, Russia
antipovden@yandex.ru
[2] École Polytechnique, CNRS, Laboratoire d'Informatique (LIX), Palaiseau, France
doerr@lix.polytechnique.fr

Abstract. To gain a better theoretical understanding of how evolutionary algorithms cope with plateaus of constant fitness, we analyze how the $(1 + 1)$ EA optimizes the n-dimensional $\textsc{Plateau}_k$ function. This function has a plateau of second-best fitness in a radius of k around the optimum. As optimization algorithm, we regard the $(1 + 1)$ EA using an arbitrary unbiased mutation operator. Denoting by α the random number of bits flipped in an application of this operator and assuming $\Pr[\alpha = 1] = \Omega(1)$, we show the surprising result that for $k \geq 2$ the expected optimization time of this algorithm is

$$\frac{n^k}{k! \Pr[1 \leq \alpha \leq k]}(1 + o(1)),$$

that is, the size of the plateau times the expected waiting time for an iteration flipping between 1 and k bits. Our result implies that the optimal mutation rate for this function is approximately k/en. Our main analysis tool is a combined analysis of the Markov chains on the search point space and on the Hamming level space, an approach that promises to be useful also for other plateau problems.

Keywords: Runtime analysis · Theory · Markov chains · Mutation

1 Introduction

This work aims at making progress on several related subjects—we aim at understanding how evolutionary algorithms optimize non-unimodal[1] fitness functions, what mutation operators to use in such settings, how to analyze the behavior of evolutionary algorithms on large plateaus of constant fitness, and in particular, how to obtain runtime bounds that are precise including the leading constant.

The recent work [14] observed that a large proportion of the theoretical work in the past concentrates on analyzing of how evolutionary algorithms optimize unimodal fitness functions and that this can lead to misleading recommendations how to design evolutionary algorithms. Based on a precise analysis of how

[1] As common in optimization, we reserve the notion *unimodal* for objective functions such that each non-optimal search point has a strictly better neighbor.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 117–128, 2018.
https://doi.org/10.1007/978-3-319-99259-4_10

the $(1+1)$ EA optimizes jump functions, it was observed that the classic recommendation to use standard bit mutation with mutation rate $\frac{1}{n}$ is far from optimal for this function class. For jump size k, a speed-up of order $k^{\Theta(k)}$ can be obtained from using a mutation rate of $\frac{k}{n}$.

Jump functions are difficult to optimize, because the optimum is surrounded by a large set of search points of very low fitness (all search points in Hamming distance 1 to $k-1$ from the optimum). However, this is not the only reason for fitness functions being difficult. Another challenge for most evolutionary algorithms are large plateaus of constant fitness. On such plateaus, the evolutionary algorithm learns little from evaluating search points and consequently performs an unguided random walk. To understand this phenomenon in more detail, we propose a class of fitness function very similar to jump functions. A *plateau function* with plateau parameter k is identical to a jump function with jump size k except that the $k-1$ Hamming levels around the optimum do not have a small fitness, but have the same second-best fitness as the k-th Hamming level. Consequently, these functions do not have true local optima (in which an evolutionary algorithm could get stuck for longer time), but only a plateau of constant fitness. Our hope is that this generic fitness function with a plateau of scalable size may aid the understanding of plateaus in evolutionary computation in a similar manner as the jump functions have led to many useful results about the optimization of functions with true local optima, e.g., [4–6,8,15,18].

When trying to analyze how evolutionary algorithms optimize plateau functions, we observe that the active area of theoretical analyses of evolutionary algorithms has produced many strong tools suitable to analyze how evolutionary algorithms make true progress (e.g., various form of the fitness level method [7,24,25] or drift analysis [13,17,20]), but much less is known on how to analyze plateaus. This is not to mean that plateaus have not been analyzed previously, see, e.g., [3,10,16], but these results appear to be more ad hoc and less suitable to derive generic methods for the analysis of plateaus. In particular, with the exception of [16], we are not aware of any results that determine the runtime of an evolutionary algorithm on a fitness function with non-trivial plateaus precise including the leading constant (whereas a decent number of very precise results have recently appeared for unimodal fitness functions, e.g., [2,9,21,26]).

Such precise results are necessary for our further goal of understanding the influence of the mutation operator on the efficiency of the optimization process. Mutation is one of the most basic building blocks in evolutionary computation and has, consequently, received significant attention also in the runtime analysis literature. We refer to the discussion in [14] for a more extensive treatment of this topic and note here only already small changes of the mutation operator or its parameters can lead to a drastic change of the efficiency of the algorithm [11,12]

Our Results: Our main result is a very general analysis of how the simplest mutation-based evolutionary algorithm, the $(1+1)$ EA, optimizes the n-dimensional plateau function with plateau parameter $k \in \mathbb{N}$. We allow the algorithm to use any unbiased mutation operator (including, e.g., 1-bit flips, standard-bit mutation with an arbitrary mutation rate, or the fast mutation

operator of [14]) as long as the operator flips exactly one bit with probability at least some positive constant. This assumption is natural, but also necessary to ensure that the algorithm can reach all points on the plateau. Denoting the number of bits flipped in an application of this operator by the random variable α, we prove that the expected optimization time (number of fitness evaluations until the optimum is visited) is

$$\frac{n^k}{k! \Pr[1 \leq \alpha \leq k]}(1 + o(1)).$$

This result, tight apart from lower order terms only, is remarkable in several respects. It shows that the performance depends very little on the particular mutation operator, only the probability to flip between 1 and k bits has an influence. The absolute runtime is also surprising—it is the size of the plateau times the waiting time for a one-to-k bit flip.

A similar-looking result was obtained in [16], namely that the expected runtime of the $(1+1)$ EA with 1-bit mutation and with standard-bit mutation with rate $\frac{1}{n}$ on the needle function is (apart from lower order terms) the size of the plateau times the probability to flip a positive number of bits (which is 1 for 1-bit mutation and $(1 - o(1))(1 - \frac{1}{e})$ for standard bit mutation with rate $1/n$). Our result thus complements this result (valid for two specific mutation operators and for the plateau of radius n around the unique optimum) with an analogous result for constrained plateaus of arbitrary (constant) radius $k \geq 2$ around the optimum and for arbitrary unbiased mutation operators.

We note that there is a substantial difference between the case $k = n$ and k constant. Since the needle function consists of a plateau containing the whole search space apart from the optimum, the optimization time in this case is just the hitting time of a particular search point when doing an undirected random walk (via repeated mutation) on the hypercube $\{0, 1\}^n$. For the function considered in this paper, the plateau has a large boundary. More precisely, almost all[2] search points of the plateau lie on its outer boundary and furthermore, all these search points have almost all their neighbors outside the plateau. Hence a large number of iterations (namely almost all) are lost in the sense that the mutation operator generates a search point outside the plateau (and different from the optimum), which is not accepted. Interestingly, as our result shows, the optimization of such restricted plateaus is not necessarily significantly more difficult (relative to the plateau size) than the optimization of the unrestricted needle plateau.

Our precise runtime analysis allows to deduce a number of particular results. For example, when using standard bit mutation, the optimal[3] mutation rate is $\frac{k}{en}$. This is by a constant factor less than the optimal rate of $\frac{k}{n}$ for the jump function with jump size k, but again a factor of $\Theta(k)$ larger than the classic recommendation of $\frac{1}{n}$, which is optimal for many unimodal fitness functions.

[2] In the usual asymptotic sense, that is, meaning all but a lower order fraction.

[3] We call a mutation rate optimal when it differs from the truly optimal rate at most by lower order terms, that is, e.g. a factor of $(1 \pm o(1))$.

Hence our result confirms that the optimal mutation rates can be significantly higher for non-unimodal fitness functions. While the optimal mutation rates for jump and plateau functions are similar, the effect of using the optimal rate is very different. For jump functions, an $k^{\Theta(k)}$ factor speed-up (compared to the standard recommendation of $\frac{1}{n}$) was observed, here the influence of the mutation operator is much smaller, namely the factor $\Pr[1 \leq \alpha \leq k]$, which is trivially at most 1, but which was assumed to be at least some positive constant. Interestingly, our results imply that the fast mutation operator described in [14] is not more effective than other unbiased mutation operators, even though it was proven to be significantly more effective for jump functions and it has shown good results in some practical problems [23].

So one structural finding, which we believe to be true for larger classes of problems and which fits to the result [16] for needle functions, is that the mutation rate, and more generally, the particular mutation operator which is used, is less important while the evolutionary algorithm is traversing a plateau of constant fitness.

The main technical novelty in this work is that we model the optimization process via two different Markov chains describing the random walk on the plateau, namely the chain defined on the $\Theta(n^k)$ elements of the plateau (plus the optimum) and the chain obtained from aggregating these into the total mass on the Hamming levels. Due to the symmetry of the process, one could believe that it suffices to regard only the level chain. The chain defined on the elements, however, has some nice features which the level chain is missing, among others, a symmetric transition matrix (because for any two search points x and y on the plateau, the probability of going from x to y is the same as the probability of going from y to x). For this reason, we find it fruitful to switch between the two chains. Exploiting the interplay between the two chains and using classic methods from linear algebra, we find the exact expression for the expected runtime.

The abstract idea of switching between the chain of all the elements of the plateau and an aggregated chain exploiting symmetries of the process as well as the linear algebra arguments we use are not specific to our particular problem. For this reason, we are optimistic that our techniques may be applied as well to other optimization processes involving plateaus[4].

2 Problem Statement

We consider the maximization of a function that resembles the OneMax function, but has a plateau of second-highest fitness of radius k around the optimum. We call this function $\textsc{Plateau}_k$ and define it as follows.

[4] For reasons of space, not all mathematical proofs could fit into this extended abstract. The proofs can be found in [1].

$$\textsc{Plateau}_k(x) := \begin{cases} \textsc{OneMax}(x), & \text{if } \textsc{OneMax}(x) \leq n-k, \\ n-k, & \text{if } n-k < \textsc{OneMax}(x) < n, \\ n, & \text{if } \textsc{OneMax}(x) = n, \end{cases}$$

where $\textsc{OneMax}(x)$ is the number of one-bits in x.

Notice that the plateau of the function $\textsc{Plateau}_k(x)$ consists of all bit-strings that have at least $n-k$ one-bits, except the optimal bit-string $x^* = (1, \ldots, 1)$. See Fig. 1 for an illustration of $\textsc{Plateau}_k$. Since a reviewer asked for it, we note that the unary unbiased black-box complexity (see [19] for the definition) of $\textsc{Plateau}_k$ is $\Theta(n \log n)$. Here the lower bound follows from the $\Omega(n \log n)$ lower bound for the unary unbiased black-box complexity of $\textsc{OneMax}$, see again [19], and the fact that $\textsc{Plateau}_k = f \circ \textsc{OneMax}$ for a suitable function f, hence any algorithm solving $\textsc{Plateau}_k$ can also solve $\textsc{OneMax}$. The upper bound follows along the same lines as the $O(n \log n)$ upper bound for the unary unbiased black-box complexity of $\textsc{Jump}_k$, see [8].

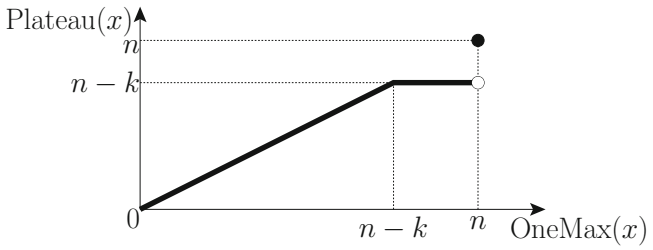


Fig. 1. Graph of the $\textsc{Plateau}$ function. As a function of unitation, the function value of a search point x depends only on the value $\|x\|_1$ of the $\textsc{OneMax}$ function.

As the optimization algorithm we consider the $(1+1)$ EA, shown in Algorithm 1, using an unbiased mutation operator. A mutation operator $\textsc{Mutate}$ for bit-string representations is called *unbiased* if it is symmetric in the bit-positions $[1..n]$ and in the bit-values 0 and 1. This is equivalent to saying that for all $x \in \{0,1\}^n$ and all automorphisms σ of the hypercube $\{0,1\}^n$ (respecting Hamming neighbors) we have $\sigma^{-1}(\textsc{Mutate}(\sigma(x)) = \textsc{Mutate}(x)$ (and this is an equality of distributions). The notation of unbiasedness was introduced (also for higher-arity operators) in the seminal paper [19].

For our purposes, it suffices to know that the set of unbiased mutation operators consists of all operators which can be described as follows. First, we choose a number $\alpha \in [0..n]$ according to some probability distribution and then we flip exactly α bits chosen uniformly at random. Examples for unbiased operators are the operator of Random Local Search, which flips a random bit, or standard bit mutation, which flips each bit independently with probability $\frac{1}{n}$. Note that in the first case α is always equal to one, whereas in the latter α follows a binomial distribution with parameters n and $\frac{1}{n}$.

Algorithm 1. The (1+1) EA with unary unbiased mutation operator MUTATE maximizing the function f

```
1: x ← random bit string of length n
2: repeat
3:     y ← Mutate(x)
4:     if f(y) ≥ f(x) then
5:         x ← y
6:     end if
7: until forever.
```

Additional Assumptions: The class of unbiased mutation operators contains a few operators which are unable to solve even very simple problems. For example, operators that always flips exactly two bits never finds the optimum of any function with unique optimum if the initial individual has an odd Hamming distance from the optimum. To avoid such difficulties, we only consider unbiased operators that have at least a constant (positive) probability to flip exactly one bit.

As usual in runtime analysis, we are interested in the optimization behavior for large problem size n. Formally, this means that we view the runtime $T = T(n)$ as a function of n and aim at understanding its asymptotic behavior for n tending to infinity. We aim at sharp results (including finding the leading constant), that is, we try to find a simple function $\tau : \mathbb{N} \to \mathbb{N}$ such that $T(n) = (1 + o(1))\tau(n)$, which is equivalent to saying that $\lim_{n\to\infty} T(n)/\tau(n) = 1$. In this limit sense, however, we treat k as a constant, that is, k is a given positive integer and not also a function of n. Since the case $k = 1$ is well-understood (PLATEAU_1 is the well-known ONEMAX function), we always assume $k \geq 2$.

3 Preliminaries and Notation

As long as the unbiased operator with constant probability flips exactly one bit, the expected time to reach the plateau is $O(n \log n)$. Since the time for leaving the plateau (as shown in this paper) is $\Omega(n^k)$, we only consider the runtime of the algorithm after it has reached the plateau.

For our precise runtime analysis on the plateau we consider the plateau in two different ways. The first way is to regard a Markov chain that contains $N = \sum_{i=0}^{k-1} \binom{n}{k-i}$ states, where each state represents one element of the plateau. Note that $N = \frac{n^k}{k!} + o(n^k)$, since $\binom{n}{j} = \frac{n^j}{j!}(1 + o(1))$ for all $j \in [1..k]$. Since we only regard unbiased mutation operators, the transition probability between two elements depends only on the Hamming distance between these elements. The transition matrix P_{ind} for this chain is large and inconvenient to work with. However this matrix is symmetric due to unbiasedness of the operator that implies for mutating from individual x to individual y we need to flip exactly the same set of bits as for mutating from individual y to individual x. The symmetry of this matrix will give us some simplifications in our analysis. For

example we will use the fact that the eigenvectors of this matrix are orthogonal. We intentionally do not include optimum into this chain to understand, how the algorithm behaves before escaping the plateau. As a result, the sum of each row in P_{ind} may be less than one. We call this chain the *individual chain* and call the space of real vectors of dimension N the *individual space*, since it is the space of all possible current individuals, when the algorithm is on the plateau.

To define the second Markov chain we shall use, let us first define the *i-th level* as the set of all the search points that have exactly $n-k+i$ one-bits. Then the plateau is the union of levels 0 to $k-1$ and the optimum is level k. Notice that the i-th level contains exactly $\binom{n}{k-i}$ elements. For every i and $j \in [0..k]$ and any element of the i-th level the probability to mutate to the j-th level is the same due to the unbiasedness of the operator. Therefore we can regard a Markov chain of k states, where i-th state represents all the elements of the i-th level. We call the transition matrix of this chain P. This matrix has a size of $k \times k$. Matrix P (unlike P_{ind}) is not symmetric. Like in the individual chain, since the optimum is not represented by any state, the sum of all the outgoing transition probabilities for each state may be less than one. We call this chain the *level chain* and we call the space of real vectors of length k the *level space*. The level chain is illustrated in Fig. 2.

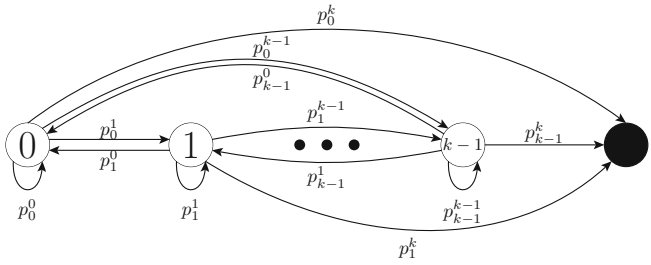


Fig. 2. Illustration of the level chain. The black circle represents the optimum that is not considered as a part of the chain, states $[0..k-1]$ represent the levels of the plateau surrounding the optimum.

There is a natural mapping from the level space to the individual space. Every vector $x = (x_0, \ldots, x_{k-1})$ can be mapped to vector $\phi(x) = (y_0, \ldots, y_{N-1})$, where $y_i = x_j / \binom{n}{k-j}$, if i-th element belongs to the j-th level. If x is a distribution over the levels, that is, $x \in [0,1]^k$ and $\|x\|_1 = 1$, then $\phi(x)$ is the distribution over the elements of the plateau which is uniform on the levels and which has the same total mass on each level as x.

This mapping has several useful properties.

1. This mapping is linear, that is, we have $\phi(\alpha x + \beta y) = \alpha\phi(x) + \beta\phi(y)$ for all $x, y \in \mathbb{R}^k$ and all $\alpha, \beta \in \mathbb{R}$. This property follows right from the definition of ϕ.

2. For all $x \in \mathbb{R}^k$ we have $\phi(xP) = \phi(x)P_{\text{ind}}$. To justify this property, let us notice two facts. First, if some mass vector y from the individual space has a uniform distribution of the mass inside each level, then after applying matrix P_{ind} to this vector, this property will remain true due to symmetry. Second, the transition of mass between levels in the level chain is the same as in the individual chain. Therefore, if we regard $\phi(xP)$ as the mass x that firstly was transferred over the level chain by matrix P and then distributed uniformly inside each level and we regard $\phi(x)P_{\text{ind}}$ as the mass x that firstly was distributed inside each level and then transferred between levels by matrix P_{ind}, then we see that it is the same vector.
3. From the previous properties we see that if x is an eigenvector of P, then $\phi(x)$ is an eigenvector of P_{ind} with the same eigenvalue. Therefore, the spectrum $\sigma(P)$ of the matrix P is a subset of the spectrum $\sigma(P_{\text{ind}})$ of matrix P_{ind}.
4. For all $x \in \mathbb{R}^k$, the Manhattan norm is invariant under ϕ, that is, $\|x\|_1 = \|\phi(x)\|_1$. This follows from the fact that all components of $\phi(x)$ that are from the same level have the same sign. Notice that an analogous property does not hold for the Euclidean norm $\|\cdot\|_2$.

4 The Spectrum of the Transition Matrix

Let p_i^k be the probability to find the optimum in one iteration if the current individual is in level i and let p_i^j for all $i, j \in [0..k-1]$ be the elements of P, that is, the transition matrix for the level chain. For all $i \in [0..k-1]$ and $j \in [0..k]$ we have

$$p_i^j = \begin{cases} \sum_{m=0}^{k-j} \binom{k-i}{j-i+m}\binom{n-k+i}{m}\binom{n}{j-i+2m}^{-1} \Pr[\alpha = j-i+2m], & \text{if } j > i, \\ \sum_{m=0}^{k-i} \binom{k-i}{m}\binom{n-k+i}{i-j+m}\binom{n}{i-j+2m}^{-1} \Pr[\alpha = i-j+2m], & \text{if } j < i, \\ 1 - \sum_{m=0, m\neq i}^{k} p_i^m, & \text{if } j = i. \end{cases}$$

The main result of this section is the following Lemma.

Lemma 1. *All the eigenvalues of matrix P are real. The largest eigenvalue of P is $\lambda_0 = 1 - o(1)$. If we have $\Pr[\alpha = 1] > c$, where c is some constant, then there exists a constant $\varepsilon > 0$ such that any other eigenvalue λ' of P satisfies $|\lambda'| < 1 - \varepsilon$.*

The fact that all the eigenvalues are real follows from the third property of ϕ and the fact that all the eigenvalues of the symmetric matrix P_{ind} are real. λ_0, that is, largest eigenvalue of P, is bounded by the minimal and the maximal row sum of P (see Perron-Frobenius Theorem [22]), which are both $1 - o(1)$. However the fact that all other eigenvalues are less than $1 - \varepsilon$ for some constant ε requires a precise analysis of the characteristic polynomial of P. The details are omitted for reasons of space.

5 Runtime Analysis

The Perron-Frobenius Theorem [22] states that for positive matrices the largest eigenvalue has a one-dimensional eigenspace. Also this theorem asserts that both left and right eigenvectors that correspond to the largest eigenvalue have all components with the same sign and they do not have any zero component. Let π^* be such a left eigenvector with positive components for P and let it be normalized in such way that $\|\pi^*\|_1 = 1$. We view π^* as distribution over the levels of the plateau and call it the *conditional stationary distribution of* P since it does not change in one iteration under the condition that the algorithm does not find the optimum. Also let $u = (u_0, \dots, u_{k-1})$ be the probability distribution in the level space that is uniform on the whole plateau, that is,

$$u_i = \binom{n}{k-i} \Big/ \sum_{j=0}^{k-1} \binom{n}{k-j} = \binom{n}{k-i} N^{-1}.$$

In the remainder, we need the following basis of the level space.

Lemma 2. *There exists a basis of the level space* $\{e^i\}_{i=0}^{k-1}$ *with the following properties.*

1. $\pi^* = e^0$;
2. e^i *is an eigenvector of* P *for all* $i \in [0..k-1]$;
3. *all* $\phi(e^i)$ *are orthogonal in the individual space.*

The first two statements are satisfied by any basis of eigenvectors of P. The third statement is not automatically satisfied if some eigenvalues have an eigenspace of dimension greater than one, however the orthogonality can be ensured via standard means from linear algebra. The details are omitted for reasons of space.

Having the basis from Lemma 2 we can prove the following relation between the vectors π^* and u.

Lemma 3. *For all* $j \in [0..k-1]$, *we have* $\pi^*_j = u_j(1 \pm O(1/\sqrt{n}))$.

The proof is again omitted for reasons of space. From Lemmas 2 and 3, we obtain our main result (with again the proof omitted for reasons of space).

Theorem 1. *The expected runtime of the* $(1+1)$ *EA using any unbiased mutation operator with constant probability to flip exactly one bit on the plateau of* $\textsc{Plateau}_k$ *function is* $N\left(\Pr[1 \le \alpha \le k]\right)^{-1}(1 + o(1))$.

6 Corollaries

We now exploit Theorem 1 to see how the choice of the mutation operator influences the runtime. Since, by Theorem 1 the expected runtime depends only on the probability to flip between 1 and k bits, this is an easy task. The proofs

of the theorems formulated in this section are omitted, since they trivially follow from Theorem 1.

We first observe that for all the unbiased operators with constant probability to flip exactly one bit, the expected optimization time is $\Theta(N)$. Hence all these mutation operators lead to asymptotically the same runtime.

The best runtime, obviously, is obtained from mutation operators which flip only between 1 and k bits. It implies that the uniformly most effective algorithm for every plateau function is Random Local Search (RLS), as for this algorithm $\Pr[1 \leq \alpha \leq k] = \Pr[\alpha = 1] = 1$.

We now analyze the runtime resulting from using standard-bit mutation as in the classic $(1+1)$ EA and from using the fast genetic algorithm, that is, the $(1+1)$ EA with a heavy-tailed mutation operator.

Recall that in standard-bit mutation, each bit is flipped independently with probability γ/n, where γ usually is a constant. Recall further that the size of the plateau is $N = \sum_{i=0}^{k-1} \binom{n}{n-k+i} = (1 \pm o(1))n^k/k!$.

Theorem 2. *Let γ be some arbitrary positive constant and $k \geq 2$. Then the $(1+1)$ EA with mutation rate γ/n optimizes* $\textsc{Plateau}_k$ *in an expected number of $N/(e^{-\gamma}\sum_{i=1}^{k} \frac{\gamma^i}{i!})(1+o(1))$ iterations. This time is asymptotically minimal for $\gamma = \sqrt[k]{k!} \approx k/e$.*

The fast genetic algorithm recently proposed in [14] is simply a (1+1) EA that uses the unbiased mutation operator such that $\Pr[\alpha = i] = 0$ for every $i > n/2$ and $i = 0$ and $\Pr[\alpha = i] = i^{-\beta}/H_{n/2,\beta}$ otherwise, where β is a parameter of the algorithm that is greater than one and $H_{n/2,\beta} := \sum_{i=1}^{n/2} i^{-\beta}$ is a generalized harmonic number. The parameter β is considered to be a constant over the problem size.

Theorem 3. *The expected runtime of the fast genetic algorithm on* $\textsc{Plateau}_k$ *is $C_{kn}N$, where C_{kn} can be bounded by constants, namely $C_{kn} \in \left[\frac{1}{\beta-1} - o(1)}{H_{k,\beta}}, \frac{\frac{1}{\beta-1}+1}{H_{k,\beta}}\right]$.*

7 Conclusion

In this paper we introduced a new method to analyze the runtime of evolutionary algorithms on plateaus. This method does not depend on the particular mutation operator used by the EA as long as there is a constant positive probability to flip a single random bit. We performed a very precise analysis on the particular class of plateau functions, but we are optimistic that similar methods can be applied for the analysis of other plateaus. For example, Lemmas 1, 2 and 3 remain true for those plateaus of the function XDIVK (that is defined as $\lfloor \textsc{OneMax}(x)/k \rfloor$ for some parameter k) that are in a constant Hamming distance from the optimum (and these are the plateaus which contribute most to the runtime). That said, the proof of Lemma 1 would need to be adapted to these plateaus different from the one of our plateau function. We are optimistic that this can be done, but leave it as an open problem for now.

The inspiration for our analysis method comes from the observation that the algorithm spends a relatively long time on the plateau. So regardless of the initial distribution on the plateau, the distribution of the individual converges to the conditional stationary distribution long before the algorithm leaves the plateau. This indicates that our method is less suitable to analyze how evolutionary algorithms leave plateaus which are easy to leave, but such plateaus usually present not bigger problems in optimization.

On the positive side, our analysis method can also be used to give runtime estimates for functions having less symmetric plateaus than our PLATEAU functions. For example, assume that $f : \{0,1\}^n \to \mathbb{R}$ is a function that agrees with PLATEAU_k on all search points x with $\text{PLATEAU}(x) = \text{ONEMAX}(x)$, but has only the restriction $n - k \leq f(x) \leq n$ for the other search points. Such functions can have plateaus of arbitrary shape inside the plateau of second-best fitness of PLATEAU_k. It is easy to see that the runtime T of the $(1+1)$ EA with arbitrary unbiased mutation operator satisfies the same asymptotic upper bound $N/\Pr[\alpha \in [1..k]](1 + o(1))$ that we have proven for the PLATEAU_k function.

Overall, we are optimistic that our main analysis method, switching between the level chain and the individual chain, which might be the first attempt to devise a general analysis method for EAs on plateaus, finds further applications.

References

1. Antipov, D., Doerr, B.: Precise runtime analysis for plateaus (2018). http://arxiv.org/abs/1806.01331
2. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN 2010. LNCS, vol. 6238, pp. 1–10. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_1
3. Brockhoff, D., Friedrich, T., Hebbinghaus, N., Klein, C., Neumann, F., Zitzler, E.: On the effects of adding objectives to plateau functions. IEEE Trans. Evol. Comput. **13**(3), 591–603 (2009)
4. Buzdalov, M., Doerr, B., Kever, M.: The unrestricted black-box complexity of jump functions. Evol. Comput. **24**(4), 719–744 (2016)
5. Dang, D.-C., et al.: Emergence of diversity and its benefits for crossover in genetic algorithms. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 890–900. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_83
6. Dang, D.C., et al.: Escaping local optima with diversity mechanisms and crossover. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2016, pp. 645–652. ACM (2016)
7. Dang, D., Lehre, P.K.: Runtime analysis of non-elitist populations: from classical optimisation to partial information. Algorithmica **75**(3), 428–461 (2016)
8. Doerr, B., Doerr, C., Kötzing, T.: Unbiased black-box complexities of jump functions. Evol. Comput. **23**(4), 641–670 (2015)
9. Doerr, B., Fouz, M., Witt, C.: Sharp bounds by probability-generating functions and variable drift. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, pp. 2083–2090. ACM (2011)

10. Doerr, B., Hebbinghaus, N., Neumann, F.: Speeding up evolutionary algorithms through asymmetric mutation operators. Evol. Comput. **15**, 401–410 (2007)
11. Doerr, B., Jansen, T., Sudholt, D., Winzen, C., Zarges, C.: Mutation rate matters even when optimizing monotonic functions. Evol. Comput. **21**(1), 1–27 (2013)
12. Doerr, B., Jansen, T., Klein, C.: Comparing global and local mutations on bit strings. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008, pp. 929–936. ACM (2008)
13. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. Algorithmica **64**(4), 673–697 (2012)
14. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 777–784. ACM (2017). http://arxiv.org/abs/1703.03334
15. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**(1–2), 51–81 (2002)
16. Garnier, J., Kallel, L., Schoenauer, M.: Rigorous hitting times for binary mutations. Evol. Comput. **7**(2), 173–203 (1999)
17. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **127**, 51–81 (2001)
18. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms–a proof that crossover really can help. Algorithmica **34**(1), 47–66 (2002)
19. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**(4), 623–642 (2012)
20. Lehre, P.K., Witt, C.: Concentrated hitting times of randomized search heuristics with variable drift. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 686–697. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13075-0_54
21. Lissovoi, A., Oliveto, P.S., Warwicker, J.A.: On the runtime analysis of generalised selection hyper-heuristics for pseudo-boolean optimisation. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 849–856. ACM (2017)
22. Meyer, C.D. (ed.): Matrix Analysis and Applied Linear Algebra. Society for Industrial and Applied Mathematics, Philadelphia (2000)
23. Mironovich, V., Buzdalov, M.: Hard test generation for maximum flow algorithms with the fast crossover-based evolutionary algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2015, pp. 1229–1232. ACM (2015)
24. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Trans. Evol. Comput. **17**, 418–435 (2013)
25. Wegener, I.: Theoretical aspects of evolutionary algorithms. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 64–78. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_6
26. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Comb. Probab. Comput. **22**(2), 294–318 (2013)

Ring Migration Topology Helps Bypassing Local Optima

Clemens Frahnow(✉) and Timo Kötzing

Hasso Plattner Institute, Prof.-Dr.-Helmert-Straße 2-3, Potsdam 14482, Germany
hpi-info@hpi.de
https://hpi.de/friedrich

Abstract. Running several evolutionary algorithms in parallel and occasionally exchanging good solutions is referred to as island models. The idea is that the independence of the different islands leads to diversity, thus possibly exploring the search space better. Many theoretical analyses so far have found a complete (or sufficiently quickly expanding) topology as underlying migration graph most efficient for optimization, even though a quick dissemination of individuals leads to a loss of diversity.

We suggest a simple fitness function Fork with two local optima parametrized by $r \geq 2$ and a scheme for composite fitness functions. We show that, while the $(1+1)$ EA gets stuck in a bad local optimum and incurs a run time of $\Theta(n^{2r})$ fitness evaluations on Fork, island models with a complete topology can achieve a run time of $\Theta(n^{1.5r})$ by making use of rare migrations in order to explore the search space more effectively. Finally, the ring topology, making use of rare migrations and a large diameter, can achieve a run time of $\tilde{\Theta}(n^r)$, the black box complexity of Fork. This shows that the ring topology can be preferable over the complete topology in order to maintain diversity.

Keywords: Evolutionary computation · Island models
Ring topology · Run time analysis

1 Introduction

In heuristic optimization, evolutionary algorithms are a technique that is capable of finding good solutions by employing strategies inspired by evolution [2]. One way to understand why some optimization algorithms are more successful than others is to prove rigorous run time bounds on test functions which embody a typical challenge occurring in realistic optimization problems. For example, the famous OneMax function, which assigns the number of 1s of a bit string x as the fitness of x, embodies the challenge of solving independent problems concurrently. The LeadingOnes function, counting the number of leading 1s of a bit string, embodies the problem of solving otherwise independent problems sequentially (where the order is typically unknown). Thus, OneMax and

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 129–140, 2018.
https://doi.org/10.1007/978-3-319-99259-4_11

LeadingOnes are fruitful test functions to analyze search heuristics on, they simulate important properties of realistic search spaces in an analyzable way.

We introduce a new representative fitness function Fork which poses a choice of two possible directions, the fork. One of the directions is a dead end, a local and not global optimum; we call this the *valley*. The other is the global optimum. We use a parameter $r > 1$ and formalize Fork by using OneMax, but assigning two elements with (disjoint) sets of r 0s to be the global optimum and the valley. Thus, once trapped in the valley, it is hard to find the global optimum. Since the probability of finding the global optimum before the valley is exactly $1/2$ due to the symmetry of the search space (see also Lemma 4), making random restarts with the well-known (1+1) EA (or about $\log n$ independent runs) can efficiently find the global optimum.

However, for realistic optimization problems, forks can happen not just as the last step of the search, but over and over again. The probability that a run will succeed and choose the right path each time decreases exponentially with the number of fork decisions to be made. We formalize this with *composite fitness functions*. We give a general scheme for building fitness functions out of base functions by dividing the bit string into blocks. As an example for solving k successive Fork functions, we can divide the bit string of length n into k equal parts. Each part contributes to the total fitness with its Fork-value, but only if all previous blocks are already optimized; this scheme was already used in essence by [11]. Intuitively, the resulting composite fitness function is like LeadingOnes, where each bit is a Fork function on bit strings of length n/k. Clearly, the (1+1) EA as well as independent runs on such a succession of Fork functions are unlikely to succeed.

Exactly to deal with such fitness landscapes, different techniques have been introduced. One possible way in evolutionary computation is to employ *island models*, meaning multiple computing agents (so called islands) that run the same algorithm in parallel and which can share information. Various analyses of island models have been made that show its usefulness: Alba used parallel evolutionary algorithms to achieve super-linear speedups [1]. Lässig and Sudholt gave a formal analysis of island models for many different migration topologies in [12] showing how one can gain a speedup from parallelism; Badkobeh, Lehre and Sudholt could even show where the cut-off points are from which on linear speedup is no longer possible and discovered some bounds on different topologies [3]. In 2017, Lissovoi and Witt explored the performance of a parallel approach on dynamic optimization problems [13]. In all these works, the idea is to exploit the computing power that comes along with multiple islands, gaining a speedup from parallelism. Intuitively, a set up where each islands sends its best solution to all islands (called a *complete migration topology*) as often as possible leads to the smallest run times, since all islands can share the progress of all others. Doerr et al. showed that if one considers the communication between islands also as time consuming, Rumor Spreading or Binary Trees perform even better on OneMax and LeadingOnes [4], but still the emphasis is on informing all islands as efficiently as possible about every improvement found.

An essentially different work was given by Lässig and Sudholt in [11]. They used a composite fitness function where each component tries to trick the algorithm to walk up a path leading to a local optimum, which is hard to escape (similar to Fork introduced above, but here we have paths that lead to the local optima). They give an island setting that can efficiently optimize this composite function, while simple hill climbers get stuck with high probability. Note that the complete topology also performs well in this setting, even though such high connectivity typically implies the loss of diversity, which was found important in many areas of heuristic optimization. Here the diversity was maintained by focusing on rare migration and making sure that migration only occurs at opportune times.

In this paper we want to show that a high connectivity in a topology, for any frequency of migration, can lead to a loss of diversity and therefore to worse run times on Fork. In contrast to this we will show that the ring topology allows to maintain diversity. We choose the ring on λ vertices, since it is the unique graph with maximal diameter among all vertex transitive graphs with λ vertices, in contrast to the complete graph, which has minimal diameter, thus highlighting the role of a large diameter (which implies a slow spread of migrants).

First, in Sect. 2, we introduce the algorithms we deal with. Section 3 introduces the fitness functions more formally, especially Fork and our scheme for composite fitness functions; here we also give a general result for the (1+1) EA applied to such composite fitness functions which is of independent interest.

In Sect. 4 we show that the (1+1) EA fails to optimize Fork efficiently, with an expected run time of $\Theta(n^{2r})$ (see Theorem 5). Independent runs of the (1+1) EA similarly fail for compositions of Fork-functions.

Regarding island models, while it is typical to consider as optimization time the time until just one island has found the optimum, we consider the time until *all* islands have found the optimum: consider the case of optimizing k successive Fork functions as introduced above. In order to be able to continue optimization after the first Fork function has been optimized, we need a sufficient number of islands which have passed this first phase; if we were to lose a constant fraction for each Fork function, then quickly all islands would be used up and the algorithm will get stuck in a local optimum. If *all* islands make it to the next stage, then the optimization can proceed as in the first stage. We leave the rigorous argument to future work and contend ourselves with finding the time until all islands find the optimum of Fork, showing in what way the ring topology can be beneficial and, in fact, preferable to the complete topology.

In Sect. 5 we consider an island model with the complete topology, that is, the different islands run a (1+1) EA, but they occasionally share their best individual with *all* other islands. In particular, we use a parameter τ such that each round with probability $1/\tau$ each island sends its best (and only current) individual to all other islands, continuing the search with the best individual among all incoming and own individuals.[1] For optimal choice of τ and the number of islands λ, the

[1] Note that in some papers migration is considered to happen deterministically every τ rounds.

time until all islands have found the optimum here is $\Theta(n^{1.5r})$ fitness evaluations (see Corollary 17).

Next, in Sect. 6, we show that the ring topology requires only $O(n^r(\log n)^2)$ fitness evaluations until all islands have found the optimum (see Corollary 24), which equals, up to polylogarithmic factors, the black-box complexity of FORK, which is $\Theta(n^r)$ (see Proposition 1). In this sense the ring topology achieves the best possible optimization time over all black-box algorithms (up to the factor of $(\log n)^2$).

Finally, in Sect. 7, we conclude the paper with some final remarks. Almost all proofs are omitted due to space constraints, but they can be found in "Ring Migration Topology Helps Bypassing Local Optima" on https://arxiv.org.

2 Algorithms

The island model makes use of the (1+1) Evolutionary Algorithm ((1+1) EA for brevity). The goal of that algorithm is to maximize a given fitness function by trying different search points and remembering the one that gave the best result so far. The fitness function is defined on bit strings of a specific length n. The algorithm starts with a bit string chosen uniformly at random. A new individual for the input is generated each step by taking the best known input and flipping every bit independently with a probability of $\frac{1}{n}$ (*standard bit mutation*). If the fitness function yields a value that is not smaller than the best known so far, it becomes the new best individual. Algorithm 1 makes this more formal.

Algorithm 1. (1+1) EA optimizing f.

```
1 t ← 0;
2 x ← solution drawn u.a.r. from {0,1}^n;
3 while termination criterion not met do
4 |   t ← t + 1;
5 |   y ← flip each bit of x independently w/ prob. 1/n;
6 |   if f(y) ≥ f(x) then x ← y ;
```

Usually the termination criterion is met when the optimum is found. In later chapters we let this algorithm run in parallel multiple times until the optimum is found everywhere. In this case we change the termination criterion accordingly. The run time of the (1+1) EA is determined by the value of t after the algorithm terminates. For our research we use the *island model* as an approach on parallel evolutionary computation, cf. [12,14,15]. The topology is defined by an undirected graph $G = (V, E)$, where $\lambda = |V|$. Every vertex, called *island*, represents an independent agent running the (1+1) EA using standard bit mutation. Like in the (1+1) EA, the initial bit string is chosen uniformly at random. This happens independently on every island. All islands run in lockstep, meaning

they all make the same amount of fitness evaluations in the same time. Copies of the best found individual so far are shared along the edges of G whenever a migration step happens. An island overwrites its best solution when a received individual has a fitness that is not smaller than the resident best individual. Ties among incoming migrants (with maximum fitness) are broken uniformly at random. With a probability of $\frac{1}{\tau}$, every island sends its best individual to all of its neighbors. Algorithm 2 makes this more formal, where $\mathbf{x}^{(j)}$ denotes the best individual on island j. Again, t determines the run time (*optimization time*) we are mainly interested in, as it counts the number of fitness evaluations of a single island. If multiplied by λ, one gains the total number of fitness evaluations.

Algorithm 2. Island model with migration topology $G = (V, E)$ on λ islands and migration probability $1/\tau$.

$t \leftarrow 0$;
for $1 \leq j \leq \lambda$ **in parallel do**
  $\mathbf{x}^{(j)} \leftarrow$ solution drawn u.a.r. from $\{0,1\}^n$;
while *termination criterion not met* **do**
  $t \leftarrow t + 1$;
  $m \leftarrow$ **true** with probability $1/\tau$ else **false**;
  for $1 \leq j \leq \lambda$ **in parallel do**
    $\mathbf{y}^{(j)} \leftarrow$ flip each bit of $\mathbf{x}^{(j)}$ independently w/ prob. $1/n$;
    if $f(\mathbf{y}^{(j)}) \geq f(\mathbf{x}^{(j)})$ **then** $\mathbf{x}^{(j)} \leftarrow \mathbf{y}^{(j)}$;
    if m **then**
      Send $\mathbf{x}^{(j)}$ to all islands k with $\{j, k\} \in E$;
      $N = \{\mathbf{x}^{(i)} \mid \{i, j\} \in E\}$;
      $M = \{\mathbf{x}^{(i)} \in N \mid f(\mathbf{x}^{(i)}) = \max_{\mathbf{x} \in N} f(\mathbf{x})\}$;
      $\mathbf{y}^{(j)} \leftarrow$ solution drawn u.a.r. from M;
      if $f(\mathbf{y}^{(j)}) \geq f(\mathbf{x}^{(j)})$ **then** $\mathbf{x}^{(j)} \leftarrow \mathbf{y}^{(j)}$;

Observe that the final value of t is a random variable; in this paper whenever we use T we refer to this random variable.

All bounds in this work will be in terms of n, λ, τ and r simultaneously. We consider $\lambda = \lambda(n)$ and $\tau = \tau(n)$ as positive, non-decreasing, integer-valued functions whereas r is a fixed but arbitrary constant that has to be at least 2. The bounds we give describe the univariate asymptotics of the expected optimization times with respect to n for any choices of λ and τ within the given boundaries.

3 Fitness Functions

In this paper we investigate the maximization of pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}_{\geq 0}$ on bit strings $\mathbf{x} = \mathbf{x}_0\mathbf{x}_1 \dots \mathbf{x}_{n-1}$ of length n. When we talk about the *fitness* of a bit string $\mathbf{x}$ it refers to $f(\mathbf{x})$.

We want to create composites of fitness functions by nesting them into each other. To start, we will examine the run times of the two functions below. Let $|\mathbf{x}|_1$ denote the number of bits set to 1 within $\mathbf{x}$. For $r \geq 2$ and $n \geq 2r$ we define

$$\textsc{LeadingOnes}(\mathbf{x}) = \sum_{i=0}^{n-1} \prod_{j=0}^{i} \mathbf{x}_j \quad \textsc{Fork}_r^n(\mathbf{x}) = \begin{cases} n+1, & \text{if } \mathbf{x} = \{0\}^r\{1\}^{n-r}; \\ n+2, & \text{if } \mathbf{x} = \{1\}^{n-r}\{0\}^r; \\ |\mathbf{x}|_1, & \text{otherwise.} \end{cases}$$

In our work when we talk about the $\textsc{Fork}_r^n$ fitness function we will call the bit strings of fitness $n+1$ *valley*, as getting to the optimum from there is harder than it is from any other fitness. This definition of Fork is not related to the one in the work of Gießen [9].

We start by considering the general difficulty of optimizing $\textsc{Fork}_r^n$ as formalized by the *unrestricted black-box complexity* [7], for which we consider the class of $\textsc{Fork}_r^n$ composed with any automorphism of the hypercube.

Proposition 1. *For constant r, the unrestricted black-box complexity of the $\textsc{Fork}_r^n$ function class is $\Theta(n^r)$.*

Proof. The lower bound comes from having to find at least one out of two search points among all search points with Hamming distance r to the third-best individual, of which there are $\Theta(n^r)$ many. The upper bound comes from being able to find the third-best individual efficiently, for example with a (1+1) EA, and then testing in a fixed order all search points with Hamming distance r to this point. □

3.1 Composite Fitness Function

Here we define composite fitness functions formally. Let $f = (f^n)_{n\in\mathbb{N}}$ be a family of fitness functions such that, for all $n \in \mathbb{N}$, $f^n : \{0,1\}^n \to \mathbb{R}_{\geq 0}$. For this construction, we suppose that 1^n is the unique optimum (if a different bit string is the unique optimum, we apply a corresponding bit mask to make 1^n the unique optimum, but will not mention it any further). With *LeadingOnes with k-block f* we denote the fitness function which divides the input $\mathbf{x}$ into bit strings of length k $((\mathbf{x}_0\mathbf{x}_1\dots\mathbf{x}_{k-1}), (\mathbf{x}_k\mathbf{x}_{k+1}\dots\mathbf{x}_{2k-1}), \dots)$. The blocks contribute to the total fitness using fitness function f^k, but only if all previous blocks have reached the unique optimum 1^k. More formally,

$$LO_k^f(\mathbf{x}) = \sum_{i=0}^{\frac{n}{k}-1} f^k(\mathbf{x}_{ik}\mathbf{x}_{ik+1}\dots\mathbf{x}_{ik+k-1}) \cdot \prod_{j=0}^{ik-1} \mathbf{x}_j.$$

Similarly, OneMax with k-block f is defined as

$$OM_k^f(\mathbf{x}) = \sum_{i=0}^{\frac{n}{k}-1} f^k(\mathbf{x}_{ik}\mathbf{x}_{ik+1}\dots\mathbf{x}_{ik+k-1}).$$

In this paper we only analyze the run times around the LeadingOnes version. Note that it equals the LOB_b function in the work of Jansen and Wiegand [10]. In general, also other fitness functions can be used as the outer function of that nesting method, as exemplified in our definition of OneMax with k-block.

3.2 Run Time of LeadingOnes with k-block f

In the following we develop a general approach for proving run times on LeadingOnes with k-block f. We make use of the observation that there is always exactly one block that contributes to the overall fitness except all previous blocks that are already optimal.

Theorem 2. *Let T be the run time of a* (1+1) EA *Algorithm on LO_k^f to optimize a bit string of length n. We have*

$$E(T) = E\left(T_k^n\right) \frac{\left(\frac{n}{n-1}\right)^n - 1}{\left(\frac{n}{n-1}\right)^k - 1} \in \Theta\left(E\left(T_k^n\right) \frac{n}{k}\right)$$

where T_k^n is the run time to optimize f^k with bit flip probability $\frac{1}{n}$.

With this Theorem as a tool we can now derive exact bounds on functions like LeadingOnes. The following equation was already shown by Sudholt [16, Corollary 1]. Here we use the approach of composite functions, which generalizes to many other fitness functions, but note that the underlying proof idea is the same.

Corollary 3. *The expected run time of a* (1+1) EA *on* LeadingOnes *is exactly*

$$E(T_{LO}(n)) = \frac{\left(\frac{n}{n-1}\right)^{n-1} + \frac{1}{n} - 1}{2} n^2.$$

4 No Migration

In the next sections we will frequently use that m islands have to make a jump from the same fitness level to another one, where for each bit string in the current level the probability to do the jump is the same. When we call E_j the expected run time for a single island to do the jump, we can bound the expected run time $E(T)$ until one of them succeeds by

$$\frac{E_j}{2m} \leq E(T) \leq \frac{E_j}{m} + 1. \tag{1}$$

This holds due to the fact that $E(T)$ is distributed geometrically with success probability $p = \frac{1}{E(T)}$ and because $\frac{pm}{1+pm} \leq 1 - (1-p)^m \leq \frac{2pm}{1+pm}$ as shown by Badkobeh et al. [3]. We already gave an intuition of the following lemma that will be used in several proofs.

Lemma 4. *For any run of the* (1+1) EA *on* Fork_r^n *let V denote the event that the valley string occurs as a best solution before the optimum. Then,* $\Pr(V) = \frac{1}{2}$.

4.1 The (1+1) EA

For the (1+1) EA we can use Lemma 4 to see that it will be trapped with probability 1/2, which leads to the following two theorems.

Theorem 5. *The expected optimization time of the* (1+1) EA *on* $\textsc{Fork}_r^k$ *with bit flip probability* $\frac{1}{n}$ *and* $n \geq k$ *is* $E(T(k)) \in \Theta\left(n^{2r}\right)$.

Theorem 6. *The expected optimization time of the* (1+1) EA *on* LeadingOnes *with k-block* $\textsc{Fork}_r^n$ *is* $E(T) \in \Theta\left(\frac{1}{k} n^{2r+1}\right)$.

4.2 Independent Runs

In this section we examine the performance of λ islands in isolation, all running the (1+1) EA on $\textsc{Fork}_r^n$ or LeadingOnes with k-block $\textsc{Fork}_r^n$. Isolation of the islands means there is no migration between them at all. The next lemma will help us to get to the final bounds.

Lemma 7. *Let* λ *be the number of islands running the* (1+1) EA *optimizing* $\textsc{Fork}_r^n$. *Let* T_{all}^{λ} *denote the run time for all islands to get to the optimum, valley or* 1^n *as their best solution respectively, regardless of the migration topology and policy. If* λ *is polynomial in* n, $E\left(T_{all}^{\lambda}\right) \in \mathrm{O}(n \log n)$.

Theorem 8. *For* $\lambda \leq n^r$ *isolated islands the expected time to optimize* $\textsc{Fork}_r^n$ *is* $E(T) \in \mathrm{O}\left(n \log(n) + \frac{n^{2r}}{\lambda 2^{\lambda}} + \frac{n^r}{\lambda}\right)$.

Corollary 9. *If we use* $r \log n \leq \lambda \leq n^r$ *islands,* $E(T) \in \mathrm{O}\left(n \log(n) + \frac{n^r}{\lambda}\right)$.

Corollary 10. *The expected number of evaluations until all* $\lambda \leq n^r$ *islands get the optimum is in* $\Omega\left(\lambda n^{2r} \log \lambda\right)$.

Proof. To achieve this, all islands have to find the optimum. Observe that we expect half of the islands to get trapped. The probability that there are more than half as much is - by using Chernoff bounds - asymptotically more than a constant. Therefore we expect to need at least $\Omega\left(n^{2r} \log \lambda\right)$ rounds, which results in the given number of evaluations. □

Theorem 11. *For* $k \leq \frac{n}{\log \lambda}$, *the expected run time of* λ *islands optimizing* LeadingOnes *with k-block* $\textsc{Fork}_r^n$ *by running the* (1+1) EA *can be bound by* $E(T) \in \Omega\left(\frac{n^{2r}}{\lambda}\right)$.

Proof. Let V denote the event that every island gets trapped in a valley at least once during a run. From Lemma 4 one can derive that $\Pr(V) = \left(1 - \frac{1}{2^{\frac{n}{k}}}\right)^{\lambda}$.

Again we apply the law of total expectation and use the lower bound $\frac{E}{2\lambda}$ on the expected run time of λ islands until one of them makes a jump where E is the expected number of steps for a single island (Eq. (1)).

$$E(T) = E(T \mid V) \Pr(V) + E\left(T \mid \overline{V}\right) \Pr\left(\overline{V}\right) \geq \frac{n^{2r}}{2\lambda} \Pr(V) = \frac{n^{2r}}{2\lambda} \left(1 - \frac{1}{2^{\frac{n}{k}}}\right)^{\lambda}$$

Using that $k \leq \frac{n}{\log(\lambda)}$ finally gives $E(T) \geq \frac{n^{2r}}{2\lambda} \left(1 - \frac{1}{\lambda}\right)^{\lambda} \in \Omega\left(\frac{n^{2r}}{\lambda}\right)$. □

5 Complete Topology

The disadvantage of the complete topology when optimizing $\textsc{Fork}_r^n$ is that if at least one island gets the chance to migrate the valley, all islands get trapped. To get to the bounds, we first investigate the worst run time that could appear. Second, we calculate a bound on the probability for one island to find the optimum or the valley during the time it takes all islands to get to 1^n, the valley or the optimum.

In this and the following sections we frequently use the worst case run time that can occur if we do not find the optimum early enough.

Lemma 12. *For $\lambda \in \mathrm{O}\left(\frac{n^{2r-1}}{\log n}\right)$ islands being polynomial in n and any migration topology and/or policy, the expected run time to optimize $\textsc{Fork}_r^n$ can be bounded by $E(T) \in \mathrm{O}\left(\frac{n^{2r}}{\lambda}\right)$.*

To get a lower bound on the run time we want to concentrate on the case that all islands come to a state where every island has 1^n as its solution. That is why in the next lemma we show how likely it is that the optimum or the valley is generated before that state is reached.

Lemma 13. *The probability p_{ov} for a single island to generate the optimum or the valley during its way to 1^n while optimizing $\textsc{Fork}_r^k$ with (1+1) EA $_{1/n}$ is $p_{ov} \in \mathrm{O}\left(\frac{1}{k^{r-1}}\right)$ for $k \leq n$.*

Next we give a lemma that we use for the upper and the lower bound, which considers the event that and islands finds the valley and broadcasts it, thus drowning diversity.

Lemma 14. *Under the condition of at least one of $\lambda \in \mathrm{O}\left(n^{r-1}\right)$ island having 1^n and all others the valley as solution, the probability for the event Q that one island will find the valley and a migration will be made before the optimum is found by any island is $\frac{c}{2}\frac{n^r}{n^r+\tau\lambda} \leq \Pr(Q) \leq \frac{n^r+\frac{\lambda}{2}}{2n^r+\tau\lambda}$ for a constant $0 < c < 1$.*

We continue with the lower bound for the complete topology, followed by the upper bound. The restriction for λ to be in $\mathrm{O}\left(\frac{n^{r-1}}{\log n}\right)$ in the next theorems is useful since the expected number of derived optima during the first $\mathrm{O}(n \log n)$ steps is constant if we chose $\lambda \in \Theta\left(n^{r-1}\right)$. This follows from Lemmas 7 and 13.

Theorem 15. *The expected run time of $2r \log n \leq \lambda \in \mathrm{O}\left(\frac{n^{r-1}}{\log n}\right)$ islands optimizing $\textsc{Fork}_r^n$ on a complete graph is $E(T) \in \Omega\left(\frac{n^{3r}+\tau\lambda n^r}{\lambda n^r+\tau\lambda^2}\right)$.*

Theorem 16. *The expected run time of $2 \log n \leq \lambda \in \mathrm{O}\left(\frac{n^{r-1}}{\log n}\right)$ islands optimizing $\textsc{Fork}_r^n$ on a complete graph is in $E(T) \in \mathrm{O}\left(\frac{n^{3r}+\tau\lambda n^r}{\lambda n^r+\tau\lambda^2} + \frac{n^{2r+1}\log n}{\tau\lambda}\right)$.*

Corollary 17. *If we choose $\tau \in \Omega(n \log n)$ and $2r \log n \leq \lambda \in \mathrm{O}(n^{r-1-\varepsilon})$ for $\varepsilon > 0$ constant, then the optimization time for $\textsc{Fork}_r^n$ on a complete graph is $E(T) \in \Theta\left(\frac{n^{3r}+\tau\lambda n^r}{\lambda n^r+\tau\lambda^2}\right)$.*

Proof. We already have shown the lower bound in Theorem 15. The second term of the upper bound in Theorem 16 is dominated by the rest if $\tau \in \Omega(n \log n)$. Therefore it matches the lower bound. □

Corollary 18. *The number of fitness evaluations to spread the optimum to all islands is in $\Theta\left(n^{1.5r}\right)$ for the best choice of parameters λ and τ.*

Proof. This can be shown by recalling that the number of evaluations to get there is in $\Omega\left(\frac{n^{3r}+\tau\lambda n^r}{n^r+\tau\lambda} + \tau\lambda\right)$ and in $\mathrm{O}\left(\frac{n^{3r}+\tau\lambda n^r}{n^r+\tau\lambda} + \frac{n^{2r+1}\log n}{\tau} + \tau\lambda\right)$ (Theorems 15 and 16). There is no way to choose $\tau\lambda$ to get below $\Theta\left(n^{1.5r}\right)$. □

6 Ring Topology

We expect the ring topology to perform better than the complete graph due to the fact that even if one island finds the valley, there is enough time for the others to come up with the optimum before they would get informed of the valley by a neighbor. In this section we want to prove this assumption.

First we show that we expect just a small number of islands to find the valley before all other islands get to 1^n. After that we prove that we can choose a migration probability so that valleys are unlikely to be shared too often. As final step we show that all other islands have enough time to find the optimum so that we get an upper bound of the expected run time.

For those steps we define two events that can occur.

Definition 19. *Let V_b be the event that the valley was generated on at most b islands during the time until all other islands have 1^n, the valley or the optimum as their solution.*

Definition 20. *Let B_c be the event that after the time until all islands have 1^n, the valley or the optimum as their solution, there is a maximum of $c \log n$ valleys.*

Lemma 21. *If $b \geq 1 + \frac{r}{\varepsilon}$ is constant and $\lambda \in \mathrm{O}\left(n^{r-1-\varepsilon}\right)$, where $\varepsilon > 0$ is a constant, it holds that the expected run time of to optimize $\textsc{Fork}_r^n$ on any island is $E(T) \leq E(T \mid V_b) + \mathrm{O}(n)$.*

Another event that could possibly lead to a high run time is when one of the found valleys is shared too fast to all other islands. Like before we will show that this case is unlikely enough to not dominate the run time.

Lemma 22. *Let $\varepsilon > 0$, $b \geq 1 + \frac{r}{\varepsilon}$ and $c \geq 7rb$ be constants and $\lambda \in \mathrm{O}\left(n^{r-1-\varepsilon}\right)$. When $\frac{1}{\tau}$ denotes the migration probability and $\tau \in \Omega(n \log n)$, the expected run time to optimize $\textsc{Fork}_r^n$ on any island is $E(T) \leq E(T \mid V_b \cap B_c) + \mathrm{O}\left(\frac{n^r}{\lambda}\right)$.*

From the previous lemmas we can derive that if we choose τ large enough and λ small enough, we get an upper bound on the expected run time by just looking at the case of $B_c \cap V_b$ and adding $\mathrm{O}\left(\frac{n^r}{\lambda}\right)$.

Theorem 23. *For* $\tau \in \Omega(n \log n)$, $12r \log n \leq \lambda \in \mathrm{O}\left(n^{r-1-\varepsilon}\right)$ *and* $\lambda^2 \tau \geq 17r^2 n^r \log^2 n$, *the expected optimization time for* $\textsc{Fork}_r^n$ *on a Ring topology is* $E(T) \in \mathrm{O}\left(\frac{n^r}{\lambda}\right)$.

The next corollary sums up our findings and shows performance for the optimal choice of parameters.

Corollary 24. *The expected number of fitness evaluations to have all islands at the optimum is in* $\mathrm{O}\left(n^r \log^2 n\right)$ *if* τ *and* λ *are set appropriately.*

Proof. If we use the results from Theorem 23, we see that there are $\mathrm{O}\left(n^r + \tau\lambda^2\right)$ evaluations until all islands are informed. If we consider that $\tau\lambda^2 \in \mathrm{O}\left(n^r \log^2 n\right)$ we get the bound. □

7 Conclusion

The results we obtained regarding the expected number of fitness evaluations of Fork by different algorithms (until all islands have the optimum, in case of an island model) and on optimal parameter settings are

- (1+1) EA– $\Theta\left(n^{2r}\right)$;
- Independent runs – $\Omega\left(\lambda n^{2r} \log \lambda\right)$;
- Complete – $\Theta\left(n^{1.5r}\right)$;
- Ring – $\mathrm{O}\left(n^r \log^2 n\right)$.

To show this we exploited that less diversity can mean to be trapped, but it also gives advantages if there is migration between the islands. Note that a ring delays migration, since in every migration step an individual can only proceed by one island along the ring, so the total time to inform all islands is highly concentrated around the expectation. This is different in the complete topology with a high value of τ: the expected time to inform all individuals might be the same, but the concentration is weaker.

We showed when a ring topology outperforms a topology with faster dissemination of individuals due to the increase in diversity. It would be interesting to see what other properties of the search space can also gain from a ring topology. Furthermore, one could wonder whether always one of the two extremes, complete and ring, is the best choice. There may be effects that can benefit for example a two-dimensional torus over both the ring and the complete graph.

We also discussed the method of composing different fitness functions, based on nesting one fitness function in another. We focused on the case that the outer fitness function is LeadingOnes and all inner fitness functions are Fork of the same length, but in principle one can consider inner fitness function that

differ from each other, possibly also in their length and also other options for out fitness functions such as OneMax. We gave a general but precise tool to calculate run times on LeadingOnes-composite fitness functions when using the (1+1) EA.

References

1. Alba, E.: Parallel evolutionary algorithms can achieve super-linear performance. Inf. Process. Lett. **82**, 7–13 (2002)
2. Bäck, T., Fogel, D.B., Michalewicz, Z.: Handbook of evolutionary computation. Release **97**(1), B1 (1997)
3. Badkobeh, G., Lehre, P.K., Sudholt, D.: Black-box complexity of parallel search with distributed populations. In: Proceedings of the 2015 FOGA XIII. pp. 3–15. ACM (2015)
4. Doerr, B., Fischbeck, P., Frahnow, C., Friedrich, T., Kötzing, T., Schirneck, M.: Island models meet rumor spreading. In: Proceedings of the GECCO 2017, pp. 1359–1366. ACM (2017)
5. Doerr, B., Goldberg, L.A.: Adaptive drift analysis. Algorithmica **65**(1), 224–250 (2013)
6. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. Algorithmica **64**, 673–697 (2012)
7. Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Theory Comput. Syst. **39**(4), 525–544 (2006)
8. Eisenberg, B.: On the expectation of the maximum of IID geometric random variables. Stat. Probab. Lett. **78**(2), 135–143 (2008)
9. Gießen, C.: Hybridizing evolutionary algorithms with opportunistic local search. In: Proceedings of the GECCO 2013, pp. 797–804. ACM (2013)
10. Jansen, T., Wiegand, R.P.: Exploring the explorative advantage of the cooperative coevolutionary (1+1) EA. In: Cantú-Paz, E. (ed.) GECCO 2003. LNCS, vol. 2723, pp. 310–321. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_37
11. Lässig, J., Sudholt, D.: Design and analysis of migration in parallel evolutionary algorithms. Soft Comput. **17**(7), 1121–1144 (2013)
12. Lässig, J., Sudholt, D.: General upper bounds on the runtime of parallel evolutionary algorithms. Evol. Comput. **22**, 405–437 (2014)
13. Lissovoi, A., Witt, C.: A runtime analysis of parallel evolutionary algorithms in dynamic optimization. Algorithmica **78**(2), 641–659 (2017)
14. Neumann, F., Oliveto, P.S., Rudolph, G., Sudholt, D.: On the effectiveness of crossover for migration in parallel evolutionary algorithms. In: Proceedings of the GECCO 2011, pp. 1587–1594 (2011)
15. Ruciński, M., Izzo, D., Biscani, F.: On the impact of the migration topology on the Island model. Parallel Comput. **36**, 555–571 (2010)
16. Sudholt, D.: General lower bounds for the running time of evolutionary algorithms. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN 2010. LNCS, vol. 6238, pp. 124–133. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_13

Runtime Analysis of Evolutionary Algorithms for the Knapsack Problem with Favorably Correlated Weights

Frank Neumann[1(✉)] and Andrew M. Sutton[2]

[1] Optimisation and Logistics, School of Computer Science, The University of Adelaide, Adelaide, Australia
frank.neumann@adelaide.edu.au
[2] Department of Computer Science, University of Minnesota Duluth, Duluth, USA

Abstract. We rigorously analyze the runtime of evolutionary algorithms for the classical knapsack problem where the weights are favorably correlated with the profits. Our result for the $(1+1)$ EA generalizes the one obtained in [1] for uniform constraints and shows that an optimal solution in the single-objective setting is obtained in expected time $O(n^2(\log n + \log p_{\max}))$, where $p_{\max}$ is the largest profit of the given input. Considering the multi-objective formulation where the goal is to maximize the profit and minimize the weight of the chosen item set at the same time, we show that the Pareto front has size $n+1$ whereas there are sets of solutions of exponential size where all solutions are incomparable to each other. Analyzing a variant of GSEMO with a size-based parent selection mechanism motivated by these insights, we show that the whole Pareto front is computed in expected time $O(n^3)$.

1 Introduction

Evolutionary algorithms [2] and other bio-inspired algorithms have been applied to a wide range of combinatorial optimization and engineering problems. They imitate the evolution process in nature in order to generate good solutions for a given optimization or design problem. The advantage of evolutionary computation methods lies in their easy applicability to new problems and they often provide satisfying solutions to new problems at hand.

Evolutionary algorithms make uses of random decisions in their main operators such as mutation and selection and the area of runtime analysis considers bio-inspired computing methods as a special class of randomized algorithms. Substantial progress has been made over the last twenty years regarding the theoretical understanding of bio-inspired computing techniques (see [3–5] for comprehensive presentations). One of the classical problems studied quite early in the literature are linear pseudo-Boolean functions. Although linear functions are easy to optimize, the first proof that showed that a simple $(1+1)$ EA optimizes any pseudo-Boolean linear function in expected time $O(n \log n)$ was quite

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 141–152, 2018.
https://doi.org/10.1007/978-3-319-99259-4_12

involved [6]. Later on, the result was considerably improved and simplified by using drift analysis [7–9].

Considering linear functions with a linear constraint leads to the classical NP-hard knapsack problem which in general can be solved in pseudo-polynomial time by dynamic programming (see for example [10]). Although some early studies on the runtime behavior of evolutionary algorithms have been carried out for constrained combinatorial optimization problems including the knapsack problem [11], the area has been somehow neglected in the area of runtime analysis until quite recently. The mentioned results mainly concentrated on special isolated problem instances such as trap like problems. Considering general knapsack instances it was shown in [12] that a multi-objective approach is able to achieve a 1/2-approximation for the knapsack problem when using helper objectives. Furthermore, the approximation ability of a specific multi-objective approach in terms of the structure of knapsack instances was investigated in [13].

Recently, linear functions with a uniform constraint were studied in [1]. This is equivalent to the knapsack problem where all items have unit weight. We extend these studies to the class of knapsack problem with favorably correlated weights, i.e. for any two items i and j and with profits $p_i \geq p_j$ implies $w_i \leq w_j$. We study the single-objective setting where the profit of the knapsack should be maximized subject to the weights of the items meeting a given capacity bound W. Furthermore, we investigate the multi-objective setting where the goal is to maximize the profit and minimize the weight of the chosen items simultaneously. For the single-objective setting, we generalize the result on uniform weights given in [1] to knapsack instances with favorably correlated weights.

We subsequently investigate the multi-objective problem of maximizing profit and minimizing weight for the knapsack problem with favorably correlated weights. We study a variant of Global SEMO (GSEMO) [14,15] which is a baseline algorithm frequently investigated in the area of runtime analysis for evolutionary multi-objective optimization [16–19]. Investigating the multi-objective setting, we show that even favorably correlated weights can lead to an exponentially large set of search points that are all incomparable to each other. This can potentially lead to a large population in evolutionary multi-objective algorithms such as GSEMO that store at each time step all incomparable search points found so far. Based on this, we introduce size-based parent selection mechanism and show that GSEMO using this parent selection method computes the whole Pareto front for the multi-objective setting in expected time $O(n^3)$.

The outline of the paper is as follows. In Sect. 2, we introduce the knapsack problem with favorably correlated weights and the single- and multi-objective formulations studied in this paper. We investigate the structure of the objective space for the multi-objective setting and characterize optimal solutions for the single-objective problem in Sect. 3. In Sect. 4, we analyze the $(1 + 1)$ EA with respect to its runtime behavior and analyze the runtime of a multi-objective approach to compute the whole Pareto front in Sect. 5. Finally, we finish with some conclusions.

Algorithm 1. $(1+1)$ EA

```
Choose x ∈ {0,1}^n uniformly at random.;
repeat
    Obtain x' from x by flipping each bit with probability 1/n.;
    If f(x') ≥ f(x), set x := x';
until stop;
```

2 Algorithms and Problems

We consider the classical knapsack problem. The input is given by n items. Each item i has an integer profit $p_i \geq 1$ and an integer weight $w_i \geq 1$, $1 \leq i \leq n$. For our investigations, we consider the special class of instances where the weights are favorably correlated to the profits, i.e. for two items i and j, $p_i \geq p_j$ implies $w_i \leq w_j$. W.l.o.g, we assume $p_1 \geq p_2 \geq \ldots \geq p_n \geq 1$ and $1 \leq w_1 \leq w_2 \ldots \leq w_n$. This means that item i dominates each item j with $j > i$. We consider the search space $\{0,1\}^n$ and for a search point $x \in \{0,1\}^n$, we have $x_i = 1$ if item i is selected and $x_i = 0$ if it is not selected. We denote by $p(x) = \sum_{i=1}^{n} p_i x_i$ the profit and by $w(x) = \sum_{i=1}^{n} w_i x_i$ the weight of x. Furthermore, we denote by $p_{\max} = p_1$ the largest profit and by $w_{\max} = w_n$ the largest weight of the given input.

The knapsack problem with favorably correlated weights is a generalization of the problem of optimizing a linear function with a uniform constraint investigated in [1] where we have $w_i = 1$, $1 \leq i \leq n$. As for the case of a uniform constraint, the knapsack problem with favorably correlated weights can be easily solved by a greedy algorithm which includes the items as they appear in the bit string. However, analyzing the behavior of evolutionary algorithms is interesting for this problem as it clarifies the working behavior of this type of algorithm for the problem.

2.1 Single-Objective Optimization

In the single-objective case, we have given a weight bound W in addition to the given weights and profits. The goal is to compute a solution x with weight $w(x) \leq W$ and whose profit $p(x)$ is maximal among all solution meeting this weight constraint.

We investigate the classical $(1+1)$ EA shown in Algorithm 1. It starts with a solution $x \in \{0,1\}^n$ chosen uniformly at random. In each iteration an offspring x' is produced by flipping each bit of the current solution x with probability $1/n$. The offspring replaces the current solution if it is not worth with respect to fitness. For the $(1+1)$ EA, we consider the single-objective fitness function

$$f(x) = (c(x), p(x))$$

where $c(x) = \min\{W - w(x), 0\}$. Note, that $c(x)$ is strictly negative if x is infeasible and that the absolute value of $c(x)$ denotes the amount of constraint violation. We maximize f with respect to the lexicographic order, i.e. we have

$$f(x) \geq f(y) \Leftrightarrow (c(x) > c(y)) \vee ((c(x) = c(y)) \wedge (p(x) \geq p(y))).$$

This implies that for infeasible solutions the weight is reduced until a feasible solution is obtained. Furthermore, each feasible solution has a better fitness than any infeasible solution.

Analyzing the runtime of the $(1+1)$ EA, we consider the expected number of fitness evaluations until an optimal search point has been obtained for the first time. This is called the expected optimization time of the algorithm.

2.2 Multi-objective Optimization

In the multi-objective setting, we aim to maximize the profit and minimize the weight at the same time. We consider the multi-objective fitness function $f'(x) = (w(x), p(x))$ which gives the profit and weight of a given solution x. A solution y (weakly) dominates a solution x ($y \succeq x$) iff $p(y) \geq p(x)$ and $w(y) \leq w(x)$. A solution y strongly dominates x if $y \succeq x$ and $f'(x) \neq f'(y)$. The notion of dominance translates to the objective vector.

The classical goal in multi-objective optimization is to compute for each non-dominated objective vector v a solution x with $f'(x) = v$. The set of all non-dominated objective vectors is called the Pareto front of the given problem.

We consider the algorithm called GSEMO given in Algorithm 2. The algorithm has been frequently investigated in the area of runtime analysis of evolutionary multi-objective optimization [15,16]. It can be seen as a generalization of the $(1+1)$ EA to the multi-objective case. It starts with a solution x chosen uniformly at random from the considered search space and stores in its population P for each non-dominated objective vector found so far a corresponding solution. In each iteration an individual $x \in P$ is chosen uniformly at random for mutation and x produces then an offspring x' by flipping each bit with probability $1/n$. In the selection step, x' is added to the population if it is not strongly dominated by any other individual in P. If x' is added to the population all individuals that are (weakly) dominated by x' are removed from P.

For many multi-objective optimization problems the number of non-dominated objective vectors can grow exponentially in the problem size n (these problems are often referred to as *intractable* multi-objective optimization problems [20]). We will show that the Pareto front for the knapsack problem with favorably correlated weights has size at most $n+1$. However, we will show in Sect. 3 that there are sets of solutions with size exponential in n that are all incomparable to each other.

These exponentially many trade-offs motivate the following parent selection mechanism to focus the search of GSEMO. We call the size of a solution x the number of items it contains, i.e. the size is given by the number of its 1-bits $|x|_1$. In our size-based parent selection (see Algorithm 3), we determine the number of 1-bits j that a parent should have by choosing it uniformly at random from the available sizes. Afterwards, we choose the solution with maximum profit from the solutions in P having exactly j 1-bits. GSEMO with size-based parent selection differs from GSEMO by using Algorithm 3 instead of line 5 in Algorithm 2.

Algorithm 2. GSEMO Algorithm

```
Choose x ∈ {0,1}^n uniformly at random;
Determine f'(x);
P ← {x};
repeat
    Choose x ∈ P uniformly at random;
    Create x' by flipping each bit x_i of x with probability 1/n;
    Determine f'(x');
    if x' is not strongly dominated by any other search point in P then
        Include x' into P;
        Delete all other solutions z ∈ P with f'(z) ⪯ f'(x') from P
until stop;
```

Algorithm 3. Size-based Parent Selection

Let $P_i = \{x \in P \mid |x|_1 = i\}$, $0 \leq i \leq n$ and $I = \{i \mid P_i \neq \emptyset\}$.;
Choose $j \in I$ uniformly at random and choose parent $x = \arg\max\{p(y) \mid y \in P_j\}$ with the largest profit.

Note that the population of the algorithm may still grow exponentially with the problem size as the survival selection is not changed.

Analyzing the runtime of GSEMO with size-based parent selection, we consider the expected number of iterations until for each Pareto optimal objective vector a corresponding solution has been obtained. This is called the expected optimization time of the multi-objective algorithm.

3 Structure of the Objective Space

We now examine our class of problems in terms of the structure of solutions in the multi-objective space. Our investigations will also point out how optimal solutions look like in the single-objective setting.

We first show that the Pareto front for any choice of the profit and weights of our class of instances of the knapsack problem has a Pareto front of size $n+1$. We define the set $X^* = \{x \mid x = 1^k 0^{n-k}, 0 \leq k \leq n\}$ of size $n+1$, containing all strings that start with some (or no) 1-bits and have all remaining bits set to 0.

Theorem 1. *The Pareto front consists of the set of objective vectors* $F = \{f(x) \mid x \in X^*\}$.

Proof. Let x be any search point with $|x|_1 = k$ which is not of the form $1^k 0^{n-k}$. We have $p(x) \leq p(1^k 0^{n-k})$ and $w(x) \geq w(1^k 0^{n-k})$ as $p_1 \geq \ldots \geq p_n$ and $w_1 \leq \ldots \leq w_n$ which implies that x is (weakly) dominated by $1^k 0^{n-k}$. As every search point x with $|x|_1 = k$ is (weakly) dominated by $1^k 0^{n-k}$, $0 \leq k \leq n$, and all solutions in X^* are incomparable to each other due to strictly positive weights and profits, the Pareto optimal objective vectors are given by the set F. □

Let $x^* = \arg\max\{p(x) \mid x \in X^* \wedge w(x) \leq W\}$ be the feasible search point with the largest profit in X^*. This search point has the property that it is the feasible search point with the largest number of 1-bits in X^*. The following corollary shows that x^* is an optimal solution of the constrained single-objective problem.

Corollary 1. *Let $x^* \in X^*$ be the feasible solution with the largest number of 1-bits that is feasible. Then x^* is an optimal solution to the constrained single-objective knapsack problem with favorably correlated weights.*

Proof. From the proof of Theorem 1, we know that any search point is (weakly) dominated by a search point in X^*. x^* is the feasible solution with the largest profit in X^*. This implies that x^* has the maximum profit among all feasible solutions as no feasible dominated solution can have a larger profit than x^*. □

The previous observations show that all Pareto optimal objective vectors as well as the maximum profit for the constrained single-objective problem have corresponding solutions that all start with some (or no) 1-bits and have all remaining bits set to 0.

An important questions that arises when considering evolutionary multi-objective optimization is whether there can be many incomparable solutions even if the Pareto front is small. This might cause difficulties to the search of an evolutionary multi-objective algorithm. We now construct an exponentially large set of solutions and corresponding objective vectors that are incomparable to each other. We set $p_i = 2^{n-i}$ and $w_i = 2^{i-1}$, $1 \leq i \leq n$. Let $n = 4k$, $k \geq 1$ a positive integer. We define

$$S(k) = \{0110, 1001\}, \text{ if } k = 1,$$

and

$$S(k) = \{01x10, 10x01 \mid x \in S(k-1)\}, \text{ if } k \geq 2.$$

This recursively defines sets of solutions for a fixed value of k based on sets for $k-1$. Sets for the k are obtained from sets for $k-1$ by adding two bits to the left and two bits to the right of each string. The values of these bits are chosen in the way that the strings obtained do not dominate each other. This can be done as profits are exponentially decreasing and weights are exponentially increasing according to their position in the string. An illustration of the (incomparable) objective vectors for $S(4)$ with $|S(4)| = 2^4 = 16$ and the corresponding Pareto front for $n = 16$ is given in Fig. 1.

We now prove that the size of $S(k)$ grows exponentially in $n = 4k$ and that all solutions in the set are incomparable to each other.

Theorem 2. *Let $n = 4k$, $k \geq 1$ a positive integer. Then $S(k)$ contains $2^k = 2^{n/4}$ search points which are all incomparable among each other.*

Proof. The proof is by induction on k. For $k = 1$, we have two strings 0110 and 1001. We have $p(0110) = 6 < p(1001) = 9$ and $w(0110) = 6 < w(1001) = 9$

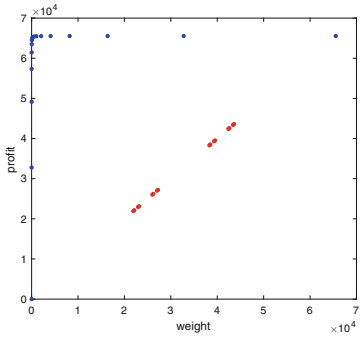


Fig. 1. Illustration of Pareto front (blue) and incomparable objective vectors of $S(4)$ (red) for $p_i = 2^{n-i}$ and $w_i = 2^{i-1}$, $1 \leq i \leq n$ when $n = 16$. (Color figure online)

which implies that the two strings are incomparable. For the induction step we assume that the set $S(k)$ consists only of incomparable solutions and show that $S(k+1)$ also only includes incomparable solutions. For each string of $S(k)$ we add 2 digits to the left and 2 digits to the right and produce two new strings in $S(k+1)$ for each string in $S(k)$. Hence, we have $|S(k+1)| = 2 \cdot S(k)| = 2^{(n/4)+1}$. Based on the assumption that all solution of $S(k)$ are incomparable, we show that all solutions in $S(k+1)$ are incomparable.

Assuming that the added bits all take on a value of 0, the profit and the weight of each solution in $S(k)$ increases by a factor of 4 where both values have been less than $2^n - 1$ and are therefore less than $4 \cdot (2^n - 1) = 2^{n+2} - 4$ after the four 0-bits have been added. We now have to take into account the effect of the 1-bits added in the two cases ($01x10$ and $10x01$). Adding 01 to the left and 10 to the right furthermore increases the profit by $2^{n+2} + 2$ and the weight by $2 + 2^{n+2}$. Adding 10 to the left and 01 to the right increases the profit by $2^{n+3} + 1$ and the weight by $1 + 2^{n+3}$. All solutions of $S(k+1)$ where the same pattern has been added are again incomparable after the addition as the profit and weight increased by at least $2^{n+2} + 2$ which is greater than the profit and weight of any solution in $S(n)$.

For the pattern $01x10$ all solutions have profit at most $2^{n+3} - 1$ and weight at most $2^{n+3} - 1$. For the pattern $10x01$ all solutions have profit at least $2^{n+3} + 1$ and weight at least $2^{n+3} + 1$. Hence, solutions belonging to different patterns are incomparable. This implies that all solutions of $S(k+1)$ are incomparable which completes the proof. □

Although the number of trade-offs that might occur is exponential, each non Pareto optimal solution can be improved towards the Pareto front. The reason for this is that any solution that is not Pareto optimal can be improved by a certain number of 2-bit flips. Let x with $|x|_1 = k$ be a non Pareto optima solution. We can transfer it into the Pareto optimal solution $1^k 0^{n-k}$ by a set of 2-bit flips where the first bit that is flipped consists of an arbitrary 0-bit among the first k bits and the second bit that is flipped consists of an arbitrary 1-bit

among the last $n-k$ bits in x. Clearly each of these 2-bit flips is accepted as they produce an offspring that dominates x. This means that any evolutionary algorithm flipping two randomly chosen bits can obtain a some progress towards the Pareto front even if it has to work with a large population.

4 Runtime Analysis of (1 + 1) EA

We first investigate the runtime behavior of the classical (1+1) EA and study the time to obtain an optimal solution. The proof considers two phases. In the first phase a feasible solution is obtained. After having obtained a feasible solution, the expected time until for the first time an optimal solution has been obtained is analyzed.

Theorem 3. *The expected optimization time of the (1 + 1) EA on the knapsack problem with favorably correlated weights is* $O(n^2(\log n + \log p_{\max}))$.

Proof. We first show that the expected time until the $(1+1)$ EA has produced a feasible solution is $O(n^2)$ by adapting the $O(n^2)$ bound for optimizing linear functions. Let x be an infeasible solution and consider the function

$$g(x) = \left(\sum_{i=1}^{n} w_i\right) - w(x)$$

giving the weight of the bits set to 0 in x. A feasible solution has value

$$W - w(x) = W - \sum_{i=1}^{n} w_i + g(x) \geq 0.$$

For technical reasons, set $w_{n+1} = 0$. We define fitness layer

$$A_i = \left\{ x \,\middle|\, \sum_{j=(n+1)-i}^{n+1} w_j \leq g(x) < \sum_{j=(n+1)-(i+1)}^{n+1} w_j \right\}, 0 \leq i \leq n-1,$$

dependent on the value of $g(x)$. Having an infeasible solution of layer $x \in A_i$, one of the last $n-i-1$ bits of x is currently set to 1. Flipping this bit produces an offspring $x' \in A_j$ with $j > i$ as it increases $g(x)$ by at least w_{n-i-1}. The probability for such a mutation step is at least $1/(en)$. There are at most n improvements until a feasible solution with $W - w(x) = W - \sum_{i=1}^{n} w_i + g(x) \geq 0$ has been obtained which implies that a feasible solution is obtained after an expected number of $O(n^2)$ steps.

Having reached a feasible solution, we consider the difference

$$\Delta(x) = p(1^k 0^{n-k}) - p(x),$$

where $1^k 0^{n-k}$ is an optimal solution according to Corollary 1. We consider the drift on $\Delta(x)$. As done in [1] for the case of uniform weights, we define

$$\text{loss}(x) = \sum_{i=1}^{k} p_i \bar{x}_i, \text{ and surplus}(x) = \sum_{i=k+1}^{n} p_i x_i.$$

Note that $\Delta(x) = \text{loss}(x) - \text{surplus}(x)$. Let r be the number of zeros among the first k bits in x (contributing to the loss) and s be the number of ones among the last $n-k$ bits in x (contributing to the surplus). Have have $p_i \geq p_j$ and $w_i \leq w_j$ iff $i \leq j$. This implies that each item belonging to the first k bits has a profit at least as high as any profit of the last $n-k$ bits and a weight at most as high as any of the last $n-k$ bits.

We consider different situations depending on the values of r and s. Our analysis has similarities to the analysis of the $(1+1)$ EA for the minimum spanning tree problem carried out in [21].

If $r > s$, then any of the r missing bits among the first k bits can be flipped and will be accepted. The sum of these gains is $\text{surplus}(x) \geq \Delta(x)$ and the expected progress in this situation is at least $\frac{r}{en} \cdot \Delta(x) \geq \frac{1}{en} \cdot \Delta(x)$. If $r = s$, then any of the missing r bits among the first k bits and any of the 1-bits among the last $n-k$ bits can be flipped to achieve an accepted solution. The sum of the progress obtainable by these 2-bit flips is at least $\Delta(x)$ and the expected progress in this situation is at least $\frac{r^2}{en^2} \cdot \Delta(x) \geq \frac{1}{en^2} \cdot \Delta(x)$. The situation $r < s$ is not possible for a feasible solution x as otherwise the search point $1^{k+1}0^{n-k-1}$ would be a feasible solution contradicting that 1^k0^{n-k} is an optimal solution.

Overall, we have at each time step t where the $(1+1)$ EA has obtained a feasible but non-optimal solution x of value $\Delta_t(x)$

$$E[\Delta_{t+1}(x) \mid \Delta_t(x)] \leq \left(1 - \frac{1}{en^2}\right) \Delta_t(x).$$

Using the multiplicative drift theorem [8] and the upper bound $np_{\max}$ and lower bound 1 on $\Delta(x)$ for any non-optimal solution x, we get the upper bound of $O(n^2(\log n + \log p_{\max}))$ on the expected time until the $(1+1)$ EA has obtained an optimal solution. □

It should be noted that $\Omega(n^2)$ is a lower bound for the $(1+1)$ EA for knapsack instances with favorably correlated weights as this bounds already holds for special instances where the weights are all 1 [1]. The reason for this lower bound are special 2-bit flips that are necessary in the case that a current non-optimal solution has a maximal number of 1-bits.

5 Runtime Analysis of GSEMO

We now analyze the expected runtime until GSEMO with size-based parent selection has computed the whole Pareto front. The proof works by considering a first phase in which a Pareto optimal solution is obtained. Afterwards missing Pareto optimal solutions can be produced by flipping a specific bit to obtain a still missing Pareto optimal objective vector.

Theorem 4. *The expected optimization GSEMO with size-based parent selection on the knapsack problem with favorably correlated weights is* $O(n^3)$.

Proof. We first upper bound the time until the algorithm has produced the search point 1^n. This solution is Pareto optimal as it has the largest possible profit and once obtained will not be removed from the population. We follow the proof for the $O(n^2)$ bound of optimizing linear pseudo-Boolean functions and use fitness-based partitions weight respect to the profit of the solutions and define fitness layer

$$A_i = \left\{ x \left| \sum_{j=1}^{i} p_j \leq p(x) \leq \sum_{j=i+1}^{n} p_j \right. \right\}, 0 \leq i \leq n-1,$$

as the set of all search points whose profit is at least as high as the profit of the first i profits and whose profit is less than the profits of the first $i+1$ profits. Furthermore, the search point 1^n constitute the optimal layer A_n of profit $p(1^n) = \sum_{i=1}^{n} p_i$.

Consider the solution with the largest profit in the population. The probability to choose this solution x for mutation is at least $1/(n+1)$ as the set I contains at most $n+1$ values and once it has been determined how many 1-bits the parent should have the individual with that number of 1-bits having the largest profit is selected. Assume that the solution of largest profit currently belongs to layer A_i, $0 \leq i \leq n-1$. In order to obtain a solution belonging to layer A_j, $j > i$, one of the leading $i+1$ bits is not set to 1 and can be flipped to obtain a profit increase of at least p_{i+1}. As $x \in A_i$ before the mutation, this leads to an offspring whose profit is increased by at least p_{i+1} and therefore belonging to layer A_j with $j > i$. The probability to select the individual with the largest profit in the population for mutation is $1/(n+1)$ and flipping the bit leading to an improvement in terms of fitness levels has probability at least $1/(en)$. There are $n+1$ different fitness layers which implies that the search point 1^n is produced after an expected number of $O(n^3)$ steps.

In the following, we work under the assumption that the search point 1^n has already been included in the population. The search point 1^n is Pareto optimal as it has the largest possible profit and will stay in the population once it has been obtained. Furthermore, for each Pareto optimal solution there does not exist any other solution in the population that has the same number of ones. As long as not all Pareto optimal objective vectors have been obtained, a new Pareto optimal objective vector can be obtained by selecting a Pareto optimal solution x with $|x| = i$ for which Pareto optimal objective vector with solution size $i+1$ or $i-1$ does not exist yet in the population. Flipping the 0-bit of x corresponding to a largest profit not yet included in the solution leads to a Pareto optimal solution y with $|y| = i+1$ for $0 \leq i \leq n-1$. Similarly, flipping the 1-bit of x corresponding to a largest profit selected in the solution leads to a Pareto optimal solution y with $|y| = i-1$ for $1 \leq i \leq n$. Choosing such an individual x for mutation has probability at least $1/(n+1)$ and flipping the bit necessary to increase the number of Pareto optimal objective vectors obtained

has probability at least $1/(en)$. Hence a new Pareto optimal objective vector is produced after an expected number of $O(n^2)$ steps. There are at most n Pareto optimal objective vectors that have not been obtained after the search point 1^n has been included in the population. Hence, after an expected number of $O(n^3)$ steps the population consists of $n + 1$ solutions, one for each Pareto optimal objective vector. □

It should be noted that using size-based parent selection where in each step the solution with the smallest weight (instead of the largest profit) is selected would lead to the same result. Here one would show that the search point 0^n is included in the population after an expected number of $O(n^3)$ steps (maximizing $(\sum_{i=1}^{n} w_i) - w(x)$ and considering always the solution with the smallest weight in the population) and show that the other Pareto optimal objective vectors are included after an additional phase of an expected number of $O(n^3)$ steps.

6 Conclusions

Constrained combinatorial optimization problems play a crucial role in real-world applications and evolutionary algorithms have been widely applied to constrained problems. With this paper, we have contributed to the theoretical understanding of evolutionary algorithms for constrained optimization problems by means of rigorous runtime analysis. We generalized the result for the $(1 + 1)$ EA obtained for uniform weights given in [1] to favorably correlated weights. Furthermore, we investigated the multi-objective formulation of the knapsack problem. Our results show that although the Pareto front has size $n + 1$, there can be exponentially large sets of non Pareto optimal objective vectors that are all incomparable. Motivated by these insights, we introduced a size-based parent selection mechanism and have shown that GSEMO using this parent selection method is able to compute the whole Pareto front in expected time $O(n^3)$.

For future work, it would be interesting to analyze GSEMO with its standard uniform parent selection. We conjecture that this algorithm would also be able to obtain the whole Pareto front in expected polynomial time as each non Pareto optimal solution can always make good progress towards the Pareto front.

Acknowledgment. This work has been supported through Australian Research Council (ARC) grant DP160102401.

References

1. Friedrich, T., Kötzing, T., Lagodzinski, G., Neumann, F., Schirneck, M.: Analysis of the (1+1) EA on subclasses of linear functions under uniform and linear constraints. In: Proceedings of the Fourteenth Conference on Foundations of Genetic Algorithms (FOGA), pp. 45–54. ACM (2017)
2. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-44874-8

3. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics: Foundations and Recent Developments, vol. 1. World Scientific, Singapore (2011)
4. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16544-3
5. Jansen, T.: Computational complexity of evolutionary algorithms. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 815–845. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9_26
6. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**(1–2), 51–81 (2002)
7. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **127**(1), 57–85 (2001)
8. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. Algorithmica **64**, 673–697 (2012)
9. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Comb. Probab. Comput. **22**, 294–318 (2013)
10. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24777-7
11. Zhou, Y., He, J.: A runtime analysis of evolutionary algorithms for constrained optimization problems. IEEE Trans. Evol. Comput. **11**(5), 608–619 (2007)
12. He, J., Mitavskiy, B., Zhou, Y.: A theoretical assessment of solution quality in evolutionary algorithms for the knapsack problem. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 141–148. IEEE (2014)
13. Kumar, R., Banerjee, N.: Running time analysis of a multiobjective evolutionary algorithm on simple and hard problems. In: Wright, A.H., Vose, M.D., De Jong, K.A., Schmitt, L.M. (eds.) FOGA 2005. LNCS, vol. 3469, pp. 112–131. Springer, Heidelberg (2005). https://doi.org/10.1007/11513575_7
14. Laumanns, M., Thiele, L., Zitzler, E.: Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions. IEEE Trans. Evol. Comput. **8**(2), 170–182 (2004)
15. Giel, O.: Expected runtimes of a simple multi-objective evolutionary algorithm. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 1918–1925. IEEE (2003)
16. Neumann, F., Wegener, I.: Minimum spanning trees made easier via multi-objective optimization. Nat. Comput. **5**(3), 305–319 (2006)
17. Friedrich, T., Neumann, F.: Maximizing submodular functions under matroid constraints by evolutionary algorithms. Evol. Comput. **23**(4), 543–558 (2015)
18. Kratsch, S., Neumann, F.: Fixed-parameter evolutionary algorithms and the vertex cover problem. Algorithmica **65**(4), 754–771 (2013)
19. Qian, C., Yu, Y., Tang, K., Yao, X., Zhou, Z.: Maximizing non-monotone/non-submodular functions by multi-objective evolutionary algorithms. CoRR abs/1711.07214 (2017)
20. Ehrgott, M.: Multicriteria Optimization. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-27659-9
21. Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theor. Comput. Sci. **378**(1), 32–40 (2007)

Theoretical Analysis of Lexicase Selection in Multi-objective Optimization

Thomas Jansen and Christine Zarges(✉)

Department of Computer Science, Aberystwyth University,
Aberystwyth SY23 3DB, UK
{t.jansen,c.zarges}@aber.ac.uk

Abstract. Lexicase selection is a parent selection mechanism originally introduced for genetic programming that has also been considered in the context of multi-objective optimization. This is the first theoretical runtime analysis of lexicase selection showing results for the bi-objective leading ones trailing zeroes benchmark problem. The lexicase selection operator is embedded into a simple hillclimbing algorithm and compared with different selection operators from the literature that are based on the classical dominance relationship. Strengths and weaknesses of the operators are demonstrated providing insights into their working principles. Results of experiments accompany the theoretical findings and point towards interesting questions for future research.

Keywords: Runtime analysis · Multi-objective optimisation
Selection operators

1 Introduction

The choice of appropriate selection operators is a crucial step in the design of evolutionary algorithms. Selection occurs twice in the typical evolutionary cycle—as selection for reproduction when selecting search points as parents and as selection for survival when deciding which search points will form the next population. In principle, the same mechanisms can be used for both scenarios and thus, it makes sense to consider common operators for both settings. Classical examples for selection operators include uniform, fitness-proportional, tournament and truncation selection (see [13] for an overview). A more recent proposal in the context of genetic programming is lexicase selection [22], where fitness evaluation is based on a number of test cases. However, it has been noted that lexicase selection lends itself naturally to multi-objective optimisation [15].

We consider lexicase selection in the context of a pseudo-Boolean bi-objective optimisation problem and compare it with three other multi-objective selection mechanisms based on the classical dominance relationship. Lexicase selection has a performance that is comparable to the best of three simple dominance-based selection mechanisms and clearly outperforms two of them.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 153–164, 2018.
https://doi.org/10.1007/978-3-319-99259-4_13

We discuss motivation and background of this study in more detail in the next section. We introduce the concrete algorithms and problems we study in a more formal way in Sect. 3. Section 4 is devoted to the theoretical analysis and contains our main results. We present results of empirical analysis in Sect. 5 that confirm the theoretical results and lead to open questions by considering a modified version of the algorithm. We summarise our findings and discuss directions for possible future research in Sect. 6.

2 Background

Our main goal is to perform a theoretical analysis of lexicase selection in the context of discrete multi-objective optimization. Lexicase selection is a parent selection mechanism that was introduced by Spector [22] to improve the performance of genetic programming in situations where fitness evaluation is based on (potentially a large number of) test cases. It is designed in a way that allows it to be used in place of tournament selection (or any other parent selection mechanism) without any other changes to the overall genetic programming algorithm. It has since been used in applications [9,11] and analysed in some detail. This includes comparing its performance to other selection mechanisms [7,10,12,18], studying its impact on population diversity [6,8], and considering its effects and performance in detail for a specific problem [19] (see also [5]). Since it can be less effective for problems in the continuous domain a variant called ε-lexicase selection for such continuous-valued problems has been introduced [15] and also analysed [14]. To the best of our knowledge the work by La Cava et al. [14] is the first theoretical analysis of a lexicase selection variant and it applies to the special version for continuous domains. We present the first theoretical analysis of the original lexicase selection method for discrete-valued problem and its impact on the expected optimization time.

Despite its origin in genetic programming lexicase selection is a general parent selection mechanism that can be employed in any kind of search algorithm that performs a step that is essentially parent selection. This includes for example evolutionary algorithms [1] and artificial immune systems [23]. We consider a simple hillclimber and compare lexicase selection with other selection mechanisms that have been used in this context. We will discuss details of all algorithms and selection operators we consider in the next section.

We consider a well known benchmark problem for multi-objective optimisation, the leading ones trailing zeros problem (formally defined as LOTZ in the following). It was introduced by Laumanns et al. [16] and was the first problem to be used for a theoretical runtime analysis. This motivates its use in this study which is a much more specific first in theoretical runtime analysis. LOTZ has been analysed for a number of simple evolutionary algorithms for multi-objective optimization (e. g., see [3,17]) and has also motivated the introduction of other, similar benchmark problems (e. g., see [4,17]). We will formally introduce the problem in the next section.

3 Algorithms, Problems, and Definitions

Our analysis focuses on lexicase selection, a parent selection method that was introduced by Spector [22]. It aims at genetic programming where the fitness of an individual is defined by its performance on a number of test cases. We consider it here as a general parent selection method for multi-objective optimization and formulate it in this context. We consider a multi-objective function f with c components so that for all x we have $f(x) = (f_1(x), f_2(x), \ldots, f_c(x))$. Without loss of generality we assume that the optimisation goal is to maximise each of the c different components. For such a multi-objective function f we describe lexicase selection as a method that selects one out of a pool P of parents. It goes through the c different objectives in random order (selecting the order randomly for each new selection operation) and eliminates for the current objective all search points that are inferior to other search points in the parent pool P. When this reduces the number of remaining potential parents to one the remaining parent is returned as selected. If at the end there is more than one potential parent left one is selected uniformly at random. We give a formal description as Algorithm 1.

Algorithm 1. Lexicase Selection

1 **for** $i \in \{1, 2, \ldots, c\}$ *in random order* **do**
2 | Remove from P all x with $f_i(x) < \max_{y \in P} \{f_i(y)\}$.
3 | **if** $|P| = 1$ **then**
4 | | return remaining $x \in P$ as selected
5 Select $x \in P$ uniformly at random and return x.

Evolutionary algorithms make use of selection in two different places in the evolutionary cycle: they use parent selection and selection for survival (e.g., see [13] for a general description). In some sense selection for survival is the opposite to parent selection since normally we select a worst individual to be removed from the population. We can do the same with lexicase selection simply by replacing the maximum by the minimum in line 2 of Algorithm 1. This follows in spirit the use of negative tournament selection in a well-known standard GP algorithm, called TinyGP, in the field guide to GP [21]. We refer to this version as negative lexicase selection and would use it whenever we need to decide which member of a population should be discarded. In the context of this work, we will not be needing this.

In multi-objective optimisation one usually considers Pareto optimality in some form as optimisation goal. For a multi-objective function $f\colon \{0,1\}^n \to \mathbb{R}^c$ with $f = (f_1, f_2, \ldots, f_c)$ we say that x weakly dominates y ($x \succeq y$) if $f_i(x) \geq f_i(y)$ for all $i \in \{1, 2, \ldots, c\}$. Note that we do not distinguish between dominance between x and y and $f(x)$ and $f(y)$ here. Clearly, the same terminology can be applied to both. We say that x dominates y ($x \succ y$) if $x \succeq y$ and there exists

some $i \in \{1, 2, \ldots, c\}$ with $f_i(x) > f_i(y)$. If neither $x \succeq y$ nor $y \succeq x$ we say that x and y are non-comparable. We say that $x \in \{0,1\}^n$ is non-dominated if there is no $y \in \{0,1\}^n$ such that $y \succ x$. The Pareto set is the set of all non-dominated search points in $\{0,1\}^n$. Its image under f is called the Pareto front.

When using an optimization heuristic for a multi-objective function one is usually interested in not only finding some solution that belongs to the Pareto set but a set of solutions such that the image of this set equals (or approximates) the Pareto front.

We analyse two different points of time. Let $x_1, x_2, \ldots$ be the sequence of search points that a multi-objective heuristic optimization algorithm generates. First, we consider the first point of time when a search point that belongs to the Pareto set is discovered (i.e., the smallest T_1 such that x_{T_1} belongs to the Pareto set and all x_i with $i < T_1$ do not belong to the Pareto set). Second, we consider the first point of time when the whole Pareto front is discovered (i.e., the smallest T_2 such that $\{f(x_1), f(x_2), \ldots, f(x_{T_2})\}$ is the Pareto front and for all $i < T_2$ we have that $\{f(x_1), f(x_2), \ldots, f(x_i)\}$ is not the Pareto front. Clearly, T_1 and T_2 are random variables if the optimization algorithm makes random choices. We will analyze their expectation in the following. Note that the definitions of T_1 and T_2 do not impose any requirements on the algorithm. It is not necessary that the algorithm knows that T_1 or T_2 have been reached. For T_2, it is not necessary that the algorithm keeps a representative for each point of the Pareto front somewhere. This allows us not to discuss the use of populations and archives and still analyse the performance of any multi-objective optimisation algorithm. If in an application it is desirable to output the Pareto front once it is found an archive that stores a representative for each non-dominated solution could easily be used.

Since this is a first theoretical run time analysis of lexicase selection it makes sense to start with an algorithm that is as simple as possible. In the context of evolutionary algorithms this is often the so-called $(1+1)$ EA. It is similar to local search because it has a population of size only 1 and creates one offspring in each round, selecting the better of parent and offspring for survival. Different from local search it uses standard bit mutations, a global mutation operator that flips each bit independently with probability $1/n$ (where n is the number of bits). While in expectation only 1 bit flips (the same number as for local search) any number of bits can flip so that a single mutation can reach any point in the search space with positive probability. While local search and the $(1+1)$ EA are often similar it is known that the global mutation can be the cause of very different behaviour [2]. In particular in the context of multi-objective optimization it is known that global mutations make the analysis considerably more difficult. This is the reason why usually results for SEMO, a simple evolutionary optimizer employing the same local steps as local search, are much easier to obtain than for GEMO, the same optimizer but with the same global mutations as used in the $(1+1)$ EA [17]. This is the reason why we consider random local search here. For the sake of completeness we give a formal description as Algorithm 2.

Algorithm 2. Random Local Search (RLS)

```
1 Select x ∈ {0,1} uniformly at random.
2 while termination criterion not met do
3 |   y := x
4 |   Select one bit in y uniformly at random and flip this bit.
5 |   Use a selection mechanism to determine which of {x, y} replaces x.
```

When considering RLS and analysing T_1 and T_2 it is important to note that we analyse the sequence of search points x that RLS generates. We do not take into account any search points y that do not replace x as new current search point.

Clearly, lexicase selection is not the only possibility to turn an evolutionary algorithm (or other heuristic optimiser like RLS) into an algorithm that can be used for multi-objective optimisation. Giel and Lehre [4] consider four different operators, three very simple and straightforward ones and one slightly more complicated on a different benchmark function. We adopt the three simple selection mechanisms to have a baseline for comparison here.

Definition 1. *The strong selection operator prefers a new search point y over x if y dominates x. The weak selection operator prefers a new search point y over x if y weakly dominates x. The weakest selection operator prefers a new search point y over x if y weakly dominates x or x and y are not comparable.*

We see that all three selection operators are different from lexicase selection and we will see how this influences the performance of RLS in the next section. We will refer to the four different algorithms as RLS with lexicase selection, RLS with strong selection, RLS with weak selection, and RLS with weakest selection in the following.

As mentioned earlier we use LOTZ as our benchmark function. The function LOTZ was introduced by Laumanns et al. [16] to facilitate theoretical analysis and it was important in the development of a theory-grounded understanding of evolutionary multi-objective optimization. This is the reason why we use the same function here. Our results will be specific to LOTZ, of course. We hope they will serve as a useful first step.

LOTZ is a bi-objective function and similar in structure to the well-known leading ones problem (e. g., see [13] for some background on this example problem). It asks to maximize the number of leading 1-bits and trailing 0-bits at the same time.

Definition 2. *The function* LOTZ$\colon \{0,1\}^n \to \mathbb{N}^2$ *is defined as* $\left(\sum_{i=1}^{n}\prod_{j=1}^{i} x[j], \sum_{i=1}^{n}\prod_{j=i}^{n}(1-x[j])\right)$.

It is well known and not difficult to see that the Pareto front of LOTZ equals $\{(0,n),(1,n-1),(2,n-2),\ldots,(n-1,1),(n,0)\}$ and that the Pareto set equals

$\{000\cdots000, 100\cdots000, 110\cdots000, \ldots, 111\cdots110, 111\cdots111\}$. The function is extreme in the sense that Pareto set and Pareto front have equal size. Each member of the Pareto front has only one representative in the search space.

4 Analysis

We analyse the performance of RLS with all four different selection operators (lexicase selection, strong selection, weak selection, weakest selection) on LOTZ. We consider both points of time that we defined in the previous section, the first point of time when a point on the Pareto front is found and also the first point of time when all points on the Pareto front have been found at least once.

Lemma 1. *The expected time until RLS with any of the four different selection operators (lexicase selection, strong selection, weak selection, weakest selection) finds a point on the Pareto front of* LOTZ *is* $O(n^2)$.

Proof. We consider the current search point x. Let $l \in \{0, 1, \ldots, n-2\}$ denote the number of leading 1-bits in x (i.e., $x[1] = x[2] = \cdots = x[l] = 1$ and $x[l+1] = 0$), let $r \in \{0, 1, \ldots, n-2\}$ denote the number of trailing 0-bits in x (i.e., $x[n] = x[n-1] = \cdots = x[n-r+1] = 0$ and $x[n-r] = 1$). Note that $l > n-2$ or $r > n-2$ imply that x is already at the Pareto front (where $r + l = n$ holds).

Let l' and r' denote the corresponding numbers in the new search point y. The probability to increase either l or r by 1 in a mutation equals $2/n$ because it suffices to flip either $x[l+1]$ or $x[n-r]$. In this case we have either $l' = l$ and $r' = r + 1$ or $l' = l + 1$ and $r' = r$.

In both cases each of the four selection mechanisms will prefer y over x because y is not inferior in any criterion and strictly better in one.

After at most n such steps we have $r + l = n$ and the Pareto front is reached. The expected waiting for each event equals $n/2$, thus the total expected waiting time is bounded by $n^2/2 = O(n^2)$. □

It is not hard to make Lemma 1 more precise and prove that the expected time is actually $\Theta(n^2)$, i.e., the bound $O(n^2)$ is asymptotically tight. However, since we are more interested in the time it takes to explore the Pareto front there is no point in making this more precise. The next results will prove that the time to find the first point of the Pareto front is insignificant in comparison to the time it takes to explore the whole Pareto front.

Theorem 1. *RLS with strong selection and RLS with weak selection never find more than a single point of the Pareto front of* LOTZ.

Proof. We consider the situation after the first point of the Pareto front of LOTZ has been found. By definition of the Pareto front, no other search point can dominate this search point. This implies that RLS with strong selection is stuck at this search point and will never go to a second search point on the Pareto front. By definition of LOTZ, no other search point can weakly dominate this search point because each point of the Pareto front of LOTZ has only

one representative point in the search point. This implies that RLS with weak selection is stuck at this search point and will never go to a second search point on the Pareto front. □

We see that RLS with strong and weak selection fail to optimise LOTZ in the sense that both are not able to find the whole Pareto front, even when given infinite time. The following result shows that RLS with lexicase selection is efficient in finding the complete Pareto front on LOTZ.

Theorem 2. *The expected time until RLS with lexicase selection finds each point of the Pareto front of* LOTZ *is* $\Theta(n^3)$.

Proof. We first prove an upper bound of $O(n^3)$ for the time to find the whole Pareto front of LOTZ. We start at a point of time when the current search point x is at the Pareto front. In the same way as in the proof of Lemma 1 let l denote the number of leading 1-bits in x and r denote the number of trailing 0-bits in x. Note that $l + r = n$ since x is at the Pareto front. Let l' and r' denote the corresponding numbers in the new search point y that is created as a mutant of x.

There are at least one and at most two positions such that either $l' = l - 1$ and $r' = r + 1$ or $l' = l + 1$ and $r' = r - 1$. For all other positions we have either $l' < l$ and $r' = r$ or $l' = l$ and $r' < r$. In this latter case the lexicase selection will prefer x over y.

We now consider the other case, the one that happens with probability at least $1/n$ and at most $2/n$. In both cases it depends on the order of selection criteria if x or y is preferred. In both cases there is one order where x is preferred over y and one order where y is preferred over x. Thus, we see that y replaces x with probability $1/2$ in this case and in total with probability at least $1/(2n)$ and at most $1/n$. Since the situation is completely symmetric x is replaced with a y that has a smaller or greater number of 1-bits with equal probability (except for the fringe cases $l = n$ and $r = n$). This implies that we can identify the changes of l (or, equivalently r) as an unbiased random work on $\{0, 1, \ldots, n\}$ where each step is taken with equal probability (except for the fringe cases $l = n$ and $r = n$) and the expected waiting time between two steps is between n and $2n$. It is well known (e.g., see [20, Chap. 6.5]) that the time to have visited each state (also know as cover time) is at most $2n \cdot 2n^2 = 4n^3$. This implies the upper bound of $O(n^3)$.

For the lower bound it suffices to notice that for the chain graph that corresponds to the unbiased random work on $\{0, 1, \ldots, n\}$ the cover time is $\Theta(n^2)$ and all the probabilities we considered were tight. Thus, the expected time is $\Theta(n^3)$. □

While strong and weak selection are both unsuccessful on LOTZ lexicase is not the only selection mechanism that is successful. The weakest selection mechanism has the same asymptotic run time as the next theorem shows.

Theorem 3. *The expected time until RLS with weakest selection finds each point of the Pareto front of* LOTZ *is* $\Theta(n^3)$.

Proof. If we consider two different points on the Pareto front they are always not comparable. Thus, in the situation where the current search point x is on the Pareto front and the new search point y is also on the Pareto front employing the weakest selection mechanism implies that y will replace x because it will prefer the new search point (see Definition 1). If the new search point is not the Pareto front it will be dominated by the old search point. Thus, RLS with the weakest selection performs the same kind of random walk on the Pareto front as RLS with lexicase selection and the same runtime bounds apply. □

It is worth noting that with lexicase selection a new search point at the Pareto front will only be accepted with probability 1/2 while with the weakest selection it will certainly be accepted. We therefore expect RLS with weakest selection to be twice as fast as RLS with lexicase selection. We remark that the same asymptotic runtime for LOTZ has been proven for the simple evolutionary multi-objective optimizer (SEMO) [17].

5 Experimental Supplements

We present the results of experiments to accompany our theoretical findings. We perform 100 runs for RLS (Algorithm 2) with lexicase selection (Algorithm 1) and with weakest selection (Definition 1) for $n = 10, 20, \ldots, 300$. We visualise the results using boxplots[1] in Figs. 1 and 2. For both algorithms we consider the time to reach the Pareto front (T_1) and the time until all points on the Pareto front have been sampled at least once (cover time T_2). Recall that both algorithms reach the Pareto front in time $O(n^2)$ (Lemma 1). Once the Pareto front is reached, only points on the Pareto front will be accepted and the random walk on the Pareto front has a cover time of $\Theta(n^3)$ for both algorithms (Theorems 2 and 3). The observed runtimes in our experiments nicely match these theoretical bounds as can be seen in Fig. 3 where we plot the mean observed number of function evaluations against a fitted polynomial based on our theoretical bounds. As expected, RLS with weakest selection is about twice as fast as RLS with lexicase selection when considering cover time. It should be noted that both algorithms exhibit a large variance when considering the cover time for the Pareto front. Thus, we observe a considerable number of outliers in our experiments. All statistical operations have been performed with R[2].

Our theoretical results only consider mutations that flip exactly one bit. As discussed previously, moving to global mutations such as standard bit mutations in the $(1+1)$ EA can have dramatic consequences. From our experiments, we can indeed see that using standard bit mutations instead of one bit mutations has a significant influence on the performance of the algorithm. We visualise results for $n = 10, 20, \ldots, 70$ using boxplots in Fig. 4. While the increase in time needed

[1] Boxplots depict the minimum, maximum, median and first and third quartiles of the observed runtimes. Circles indicate outliers, which are observations outside 1.5 times the interquartile range above the upper quartile and below the lower quartile.

[2] http://www.r-project.org

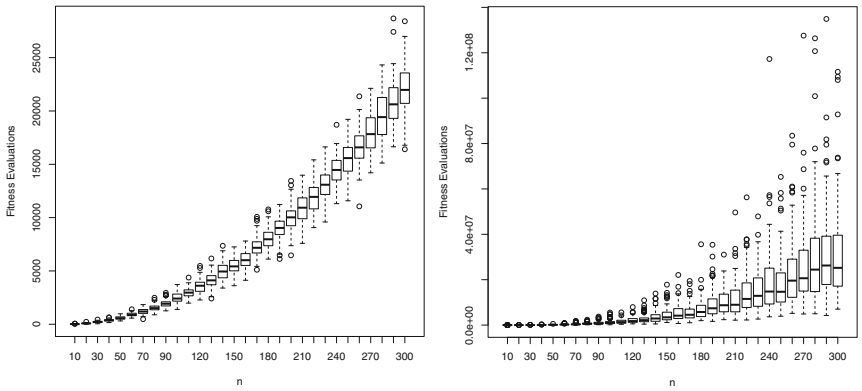


Fig. 1. Experimental results for RLS with lexicase selection over 100 runs. Boxplots show the number of fitness function evaluations until the Pareto front is reached (left) and until the whole Pareto front has been sampled (right).

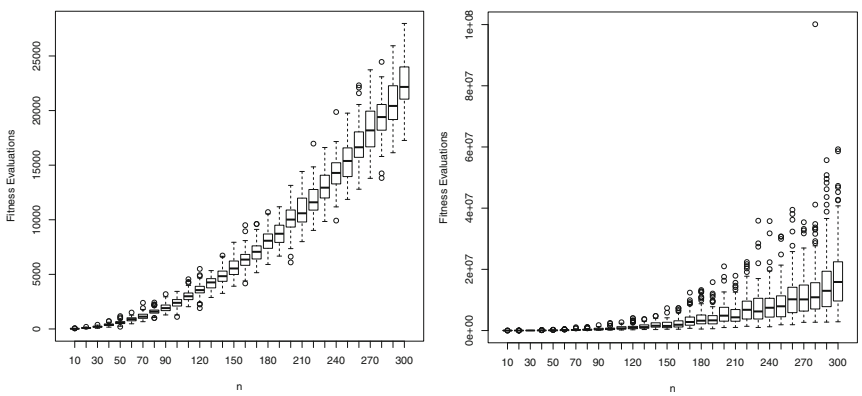
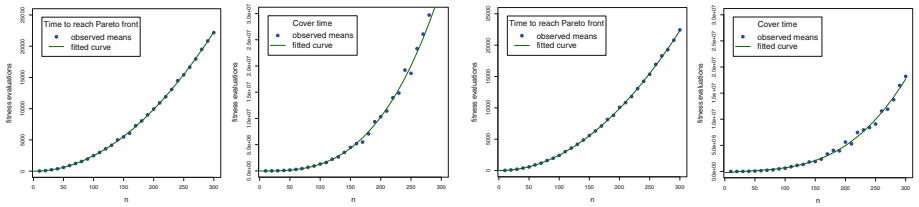


Fig. 2. Experimental results for RLS with weakest selection over 100 runs. Boxplots show the number of fitness function evaluations until the Pareto front is reached (left) and until the whole Pareto front has been sampled (right).



(a) RLS with lexicase selection: (b) RLS with weakest selection

Fig. 3. The average number fitness function evaluations observed in our experiments together with the fitted polynomials matching our theoretical results. For RLS with lexicase selection we obtain $0.2476x^2 - 11.41$ (T_1) and $1.285x^3 - 21\,190$ (T_2). For RLS with weakest selection we obtain $0.2481x^2 - 20.3$ (T_1) and $0.6479x^3 - 50\,520$ (T_2).

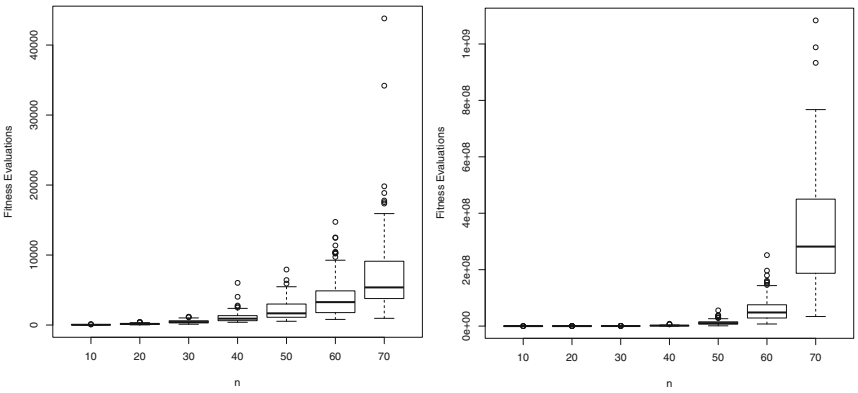


Fig. 4. Experimental results for the $(1+1)$ EA with lexicase selection over 100 runs. Boxplots show the number of fitness function evaluations until the Pareto front is reached (left) and until the whole Pareto front has been sampled (right).

to find the Pareto front seems not too different (but with larger outliers), we see that already for $n = 70$ the average number of fitness function evaluations needed to sample the entire Pareto front is significant larger than the largest outlier for the two algorithms using local mutations. We will examine this case further in the conclusions when discussing questions for future work.

6 Conclusions

We have presented a first theoretical analysis of lexicase selection in the context of discrete multi-objective optimisation. Considering a simple hillclimbing algorithm we show that lexicase selection can be used to efficiently sample the entire Pareto front of the well-known bi-objective benchmark function leading ones trailing zeros (LOTZ). We compare lexicase selection with three multi-objective selection mechanisms from the literature. We obtain asymptotically the same optimisation times when using the mechanism with the weakest selection pressure while classical selection mechanisms based on the (weak) dominance relationship get stuck on the first search point sampled on the Pareto front.

Our study hints towards several interesting questions for future research. We give some insights into these questions in the following, but leave the formal analyses for future research. A comparison with indicator-based selection [24] or considering problems with more than two objectives would also be interesting.

As discussed in the previous section, experiments indicate that moving from local mutations to global mutations significantly increases the runtime of the algorithm. We conjecture that the $(1+1)$ EA with lexicase selection is not able to find the entire Pareto front efficiently. One reason for this is that standard bit mutations may hinder the algorithm to perform a random walk on the Pareto front by preferring a non Pareto optimal search point over a Pareto optimal one.

Consider for example, the search points $x = 1^{50}\,0^{50}$ (50 1-bits followed by 50 0-bits) with fitness $(50, 50)$ and $y = 1^{30}\,0\,1^{17}\,1\,0^{51}$ with fitness $(30, 51)$ for $n = 100$, where y is the result of a 2-bit mutation of x (flipping the bits at positions 31 and 49). With probability $1/2$, lexicase selection chooses the second objective over the first objective and thus, will accept y over x. After that any mutation only effecting the 1-bits at positions 32–48 will be accepted. Note that the $(1+1)$ EA with weakest selection would also accept the new search point as x and y are incomparable.

Since lexicase selection is a parent selection mechanism it would make sense to consider populations, e.g., a $(\mu+1)$ RLS or $(\mu+1)$ EA. We conjecture that simply using lexicase selection for parent selection and negative lexicase selection for replacement will not be efficient. Again, the main reason is that search points that have made some progress on the Pareto front can be lost and replaced by search points that are not Pareto optimal. Consider for example a population that contains $x = 1^{n-1}\,0$ and $y = 1\,0^{n-1}$ and assume that we have not found the two extreme points on the Pareto front, yet. Moreover, we assume that x and y are the two points with fewest trailing zeros and leading ones, respectively. That means that the remaining search points points can be arbitrary points with less than $n-1$ but more than 1 leading ones and trailing zeros. In this case, lexicase selection will select either x or y as parent (depending on the random order of objectives). Assume w. l. o. g. that x was selected and $z = 1^{n-3}\,0\,1\,0$ was created (flipping only the bit at position $n-2$). If negative lexicase selection uses the same objective for replacement, the offspring (that is not on the Pareto front) will replace y. The latter happens with probably $1/2$ and thus, we expect to frequently lose points on the (extreme ends of the) Pareto front.

References

1. De Jong, K.A.: Evolutionary Computation. A Unified Approach. MIT Press, Cambridge (2016)
2. Doerr, B., Jansen, T., Klein, C.: Comparing global and local mutations on bit strings. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008), pp. 929–936. ACM Press (2008)
3. Giel, O.: Expected runtimes of a simple multi-objective evolutionary algorithm. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), pp. 1918–1925 (2003)
4. Giel, O., Lehre, P.K.: On the effect of populations in evolutionary multi-objective optimisation. Evol. Comput. **18**(3), 335–356 (2010)
5. Helmuth, T.: General program synthesis from examples using genetic programming with parent selection based on random lexicographic orderings of test cases. Ph.D. thesis, University of Massachusetts Amherst (2015)
6. Helmuth, T., McPhee, N.F., Spector, L.: Effects of lexicase and tournament selection on diversity recovery and maintenance. In: Genetic and Evolutionary Computation Conference (GECCO 2016), Companion, pp. 983–990 (2016)
7. Helmuth, T., McPhee, N.F., Spector, L.: The impact of hyperselection on lexicase selection. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016), pp. 717–724 (2016)

8. Helmuth, T., McPhee, N.F., Spector, L.: Lexicase selection for program synthesis: a diversity analysis. In: Riolo, R., Worzel, B., Kotanchek, M., Kordon, A. (eds.) Genetic Programming Theory and Practice XIII. GEC, pp. 151–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34223-8_9
9. Helmuth, T., Spector, L.: Evolving a digital multiplier with the PushGP genetic programming system. In: Genetic and Evolutionary Computation Conference (GECCO 2013), Companion, pp. 1627–1634 (2013)
10. Helmuth, T., Spector, L.: Word count as a traditional programming benchmark problem for genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014), pp. 919–926 (2014)
11. Helmuth, T., Spector, L.: General program synthesis benchmark suite. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015), pp. 1039–1046 (2015)
12. Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. IEEE Trans. Evol. Comput. **19**(5), 630–643 (2015)
13. Jansen, T.: Analyzing Evolutionary Algorithms. The Computer Science Perspective. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-17339-4
14. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A probabilistic and multi-objective analysis of lexicase selection and epsilon-lexicase selection. Evol. Comput. (2018, to appear). http://doi.org/10.1162/evco_a_00224
15. La Cava, W., Spector, L., Danai, K.: Epsilon-lexicase selection for regression. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016), pp. 741–748 (2016)
16. Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., Deb, K.: Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In: Guervós, J.J.M., Adamidis, P., Beyer, H.-G., Schwefel, H.-P., Fernández-Villacañas, J.-L. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 44–53. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45712-7_5
17. Laumanns, M., Thiele, L., Zitzler, E.: Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. IEEE Trans. Evol. Comput. **8**(2), 170–182 (2004)
18. Liskowski, P., Krawiec, K., Helmuth, T., Spector, L.: Comparison of semantic-aware selection methods in genetic programming. In: Genetic and Evolutionary Computation Conference (GECCO 2015), Companion, pp. 1301–1307 (2015)
19. McPhee, N.F., Donatucci, D., Helmuth, T.: Using graph databases to explore the dynamics of genetic programming runs. In: Riolo, R., Worzel, B., Kotanchek, M., Kordon, A. (eds.) Genetic Programming Theory and Practice XIII. GEC, pp. 185–201. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34223-8_11
20. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
21. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming (2008). http://lulu.com, http://www.gp-field-guide.org.uk
22. Spector, L.: Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In: Genetic and Evolutionary Computation Conference (GECCO 2012), Companion, pp. 401–408 (2012)
23. Timmis, J.: Artificial immune systems. In: Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning and Data Mining, pp. 61–65. Springer, New York (2017)
24. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: Yao, X., et al. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 832–842. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30217-9_84

Towards a Running Time Analysis of the (1+1)-EA for OneMax and LeadingOnes Under General Bit-Wise Noise

Chao Bian[1], Chao Qian[1(✉)], and Ke Tang[2]

[1] Anhui Province Key Lab of Big Data Analysis and Application, University of Science and Technology of China, Hefei 230027, China
biancht@mail.ustc.edu.cn, chaoqian@ustc.edu.cn
[2] Shenzhen Key Lab of Computational Intelligence, Southern University of Science and Technology, Shenzhen 518055, China
tangk3@sustc.edu.cn

Abstract. Running time analysis of evolutionary algorithms (EAs) under noisy environments has recently received much attention, which can help us understand the behavior of EAs in practice where the fitness evaluation is often subject to noise. One of the mainly investigated noise models is bit-wise noise, which is characterized by a pair (p, q) of parameters. However, previous analyses usually fix p or q, which makes our understanding on bit-wise noise incomplete. In this paper, we analyze the running time of the (1+1)-EA solving OneMax and LeadingOnes under general bit-wise noise. Our results largely extend the known ones in specific cases of bit-wise noise, and disclose that p and pq together decide the running time to be polynomial or super-polynomial.

1 Introduction

Evolutionary algorithms (EAs) have been widely applied to solve real-world optimization problems, where the fitness evaluation of a solution is often disturbed by noise. Although they have shown good empirical performances in noisy optimization [8], it is also important to understand the impact of noise from a theoretical viewpoint. As an essential theoretical aspect of EAs, the running time analysis has received much attention in the last two decades, and has achieved many promising results [1,9]. However, most of them focus on noise-free optimization, where the fitness evaluation is exact. The theoretical understanding of EAs is still largely incomplete for the noisy case.

Previous running time analyses in noisy evolutionary optimization mainly considered two kinds of noise models, posterior and prior. The posterior noise

This work was supported by the Ministry of Science and Technology of China (2017YFC0804003), the NSFC (61603367, 61672478), the YESS (2016QNRC001), the Science and Technology Innovation Committee Foundation of Shenzhen (ZDSYS201703031748284), and the Royal Society Newton Advanced Fellowship (NA150123).

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 165–177, 2018.
https://doi.org/10.1007/978-3-319-99259-4_14

comes from the variation on the fitness of a solution, e.g., additive noise adds a value randomly drawn from some distribution. There was a sequence of papers [2,4,5,10] mainly showing that some specific EAs (e.g., compact genetic algorithm and ant colony optimization) can well handle additive noise, and using populations can bring robustness to additive noise.

For the prior noise coming from the variation on a solution, Droste [3] first considered a specific model, one-bit noise, which flips a randomly chosen bit of a binary solution before evaluation with probability p. He analyzed the (1+1)-EA solving the OneMax problem under one-bit noise, and proved that the maximal value of p allowing a polynomial running time is $O(\log n/n)$. Gießen and Kötzing [6] considered another problem LeadingOnes, and proved that the expected running time is polynomial if $p \leq 1/(6en^2)$ and exponential if $p = 1/2$. For these two problems, Qian et al. [11,12,14] theoretically studied the effectiveness of two noise-handling strategies, threshold selection and resampling, and proved that under some large values of p, both of them can reduce the running time of the (1+1)-EA from exponential to polynomial.

Gießen and Kötzing [6] also studied another prior noise model, which flips each bit of a solution independently with probability q before evaluation. They proved that for the (1+1)-EA on OneMax, the maximal q allowing a polynomial running time is $O(\log n/n^2)$. By combining this model with one-bit noise, Qian et al. [13] studied a general prior noise model, bit-wise noise, which is characterized by a pair (p, q) of parameters. It happens with probability p, and independently flips each bit of a solution with probability q before evaluation. Their derived results for the (1+1)-EA on OneMax and LeadingOnes are shown in the middle two columns of Table 1, which give the ranges of p and q for a polynomial running time upper bound as well as a super-polynomial lower bound.

Table 1. For the expected running time of the (1+1)-EA on OneMax and LeadingOnes under bit-wise noise (p, q), the ranges of p, q for a polynomial upper bound (the first line in each cell) and a super-polynomial lower bound (the second line) are shown below.

(1+1)-EA	$(p, 1/n)$	$(1, q)$	(p, q)
OneMax	$O(\log n/n)$ [13] $\omega(\log n/n)$	$O(\log n/n^2)$ [6] $\omega(\log n/n^2)$	$p = O(\log n/n) \vee pq = O(\log n/n^2)$ $p = \omega(\log n/n) \wedge pq = \omega(\log n/n^2)$
LeadingOnes	$O(\log n/n^2)$ [13] $\omega(\log n/n)$	$O(\log n/n^3)$ [13] $\omega(\log n/n^2)$	$p = O(\log n/n^2) \vee pq = O(\log n/n^3)$ $p = \omega(\log n/n) \wedge pq = \omega(\log n/n^2)$

However, the analysis for bit-wise noise [13] only considered two specific cases: $(p, 1/n)$ (i.e., q is fixed to $1/n$) and $(1, q)$ (i.e., p is fixed to 1). Some fundamental theoretical issues are thus not addressed. For example, what is the deciding factor for the running time of the (1+1)-EA under bit-wise noise? Will the (1+1)-EA perform similarly for two scenarios (p, q) and (p', q') with $pq = p'q'$? In this paper, we analyze the running time of the (1+1)-EA solving OneMax and LeadingOnes

under general bit-wise noise, i.e., without fixing p or q. We derive the ranges of p and q for a polynomial upper bound and a super-polynomial lower bound, which are summarized in the last column of Table 1. It is easy to verify that our results cover previously known ones [6,13], as shown in the middle two columns of Table 1. The results show that p and pq together decide the running time to be polynomial or super-polynomial. We can also observe that the expected running time for two scenarios (p, q) and (p', q') with $pq = p'q'$ may be significantly different, e.g., for the (1+1)-EA on OneMax, the running time is polynomial if $p = q = \log n/n$, but super-polynomial if $p' = 1, q' = (\log n/n)^2$.

The rest of this paper is organized as follows. Section 2 introduces some preliminaries. Sections 3 and 4 present the running time analysis on OneMax and LeadingOnes, respectively. Section 5 concludes the paper.

2 Preliminaries

In this section, we first introduce the optimization problems, noise models and EAs studied in this paper, respectively, then present the analysis tools that we use throughout this paper.

2.1 OneMax and LeadingOnes

In this paper, we consider two pseudo-Boolean problems OneMax and LeadingOnes, which are widely used in EAs' theoretical analyses [1,9]. The former aims to maximize the number of 1-bits of a solution, and the latter aims to maximize the number of consecutive 1-bits counting from the left of a solution. Their optimal solutions are both $11\ldots1$ (briefly denoted as 1^n).

Definition 1 (OneMax). *The OneMax Problem of size n is to find an n bits binary string x^* that maximizes $f(x) = \sum_{i=1}^{n} x_i$.*

Definition 2 (LeadingOnes). *The LeadingOnes Problem of size n is to find an n bits binary string x^* that maximizes $f(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$.*

2.2 Bit-Wise Noise

There are mainly two kinds of noise models: prior and posterior [6,8]. The prior noise comes from the variation on a solution, while the posterior noise comes from the variation on the fitness of a solution. Previous theoretical analyses involving prior noise [2,3,6,11,12] often focused on one-bit noise, which flips a random bit of a solution before evaluation with probability p. Qian et al. [13] recently considered a natural extension of one-bit noise, i.e., the bit-wise noise model. As presented in Definition 3, it happens with probability p, and independently flips each bit of a solution with probability q before evaluation. However, the analyses in [13] considered bit-wise noise with one parameter fixed, i.e., $p = 1$ or $q = 1/n$. In this paper, we will analyze the general bit-wise noise model.

Definition 3 (Bit-wise Noise [13]). *Given $p, q \in [0,1]$, let $f^{\mathrm{n}}(x)$ and $f(x)$ denote the noisy and true fitness of a solution $x \in \{0,1\}^n$, respectively, then*

$$f^{\mathrm{n}}(x) = \begin{cases} f(x) & \text{with probability } 1-p, \\ f(x') & \text{with probability } p, \end{cases}$$

where x' is generated by independently flipping each bit of x with probability q.

Algorithm 1. (1+1)-EA

Given a function f over $\{0,1\}^n$ to be maximized, it consists of the following steps:

1: $x :=$ uniformly randomly selected from $\{0,1\}^n$.
2: Repeat until some termination condition is met
3: $\quad x' :=$ flip each bit of x with probability $1/n$.
4: $\quad$ if $f^{\mathrm{n}}(x') \geq f^{\mathrm{n}}(x)$ $\quad$ then $x := x'$.

2.3 (1+1)-EA

This paper considers the (1+1)-EA, which is a benchmark EA widely used in theoretical analyses. For noisy optimization, only a noisy fitness value $f^{\mathrm{n}}(x)$ instead of the exact one $f(x)$ can be accessed, and thus the condition in line 4 of Algorithm 1 is "if $f^{\mathrm{n}}(x') \geq f^{\mathrm{n}}(x)$" instead of "if $f(x') \geq f(x)$". Note that the reevaluation strategy is used as in [3,6,11,12]. That is, besides evaluating $f^{\mathrm{n}}(x')$, $f^{\mathrm{n}}(x)$ will be reevaluated in each iteration of the (1+1)-EA. The running time is usually defined as the number of fitness evaluations needed to find an optimal solution w.r.t. the true fitness function f for the first time [3,6,11,12].

2.4 Analysis Tools

The process of the (1+1)-EA solving OneMax or LeadingOnes under noise can be modeled as a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$. We only need to take the solution space $\{0,1\}^n$ as the chain's state space (i.e., $\xi_t \in \mathcal{X} = \{0,1\}^n$), and take the optimal solution 1^n as the chain's target state (i.e., $\mathcal{X}^* = \{1^n\}$). Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ and $\xi_{\hat{t}} = x$, we define its *first hitting time* as $\tau = \min\{t \mid \xi_{\hat{t}+t} \in \mathcal{X}^*, t \geq 0\}$. The mathematical expectation of τ, $\mathbb{E}[\![\tau \mid \xi_{\hat{t}} = x]\!] = \sum_{i=0}^{+\infty} i \cdot \mathrm{P}(\tau = i \mid \xi_{\hat{t}} = x)$, is called the *expected first hitting time* (EFHT) starting from $\xi_{\hat{t}} = x$. If ξ_0 is drawn from a distribution π_0, $\mathbb{E}[\![\tau \mid \xi_0 \sim \pi_0]\!] = \sum_{x \in \mathcal{X}} \pi_0(x) \mathbb{E}[\![\tau \mid \xi_0 = x]\!]$ is called the EFHT of the chain over the initial distribution π_0. Thus, the expected running time of the (1+1)-EA starting from $\xi_0 \sim \pi_0$ is $1 + 2 \cdot \mathbb{E}[\![\tau \mid \xi_0 \sim \pi_0]\!]$, where the term 1 corresponds to evaluating the initial solution, and the factor 2 corresponds to evaluating the offspring solution x' and reevaluating the parent solution x in each iteration. Note that we consider the expected running time of the (1+1)-EA starting from a uniform initial distribution in this paper.

In the following, we give two drift theorems that will be used to derive upper and lower bounds on the EFHT of Markov chains in the paper.

Theorem 1 (Additive Drift [7]). *Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ and a distance function $V(x)$ with $V(x \in \mathcal{X}^*) = 0$ and $V(x \notin \mathcal{X}^*) > 0$, if for any $t \geq 0$ and any ξ_t with $V(\xi_t) > 0$, there exists a real number $c > 0$ such that $\mathbb{E}[\![V(\xi_t) - V(\xi_{t+1}) \mid \xi_t]\!] \geq c$, then the EFHT satisfies that $\mathbb{E}[\![\tau \mid \xi_0]\!] \leq V(\xi_0)/c$.*

Theorem 2 (Negative Drift with Self-loops [15]). *Let X_t, $t \geq 0$, be real-valued random variables describing a stochastic process. Suppose there exists an interval $[a, b] \subseteq \mathbb{R}$, two constants $\delta, \epsilon > 0$ and, possibly depending on $l := b - a$, a function $r(l)$ satisfying $1 \leq r(l) = o(l/\log(l))$ such that for all $t \geq 0$:*

(1) $\forall a < i < b : \mathbb{E}[\![X_t - X_{t+1} \mid X_t = i]\!] \leq -\epsilon \cdot \mathrm{P}(X_{t+1} \neq i \mid X_t = i)$,

(2) $\forall i > a, j \in \mathbb{N}^+ : \mathrm{P}(|X_{t+1} - X_t| \geq j \mid X_t = i) \leq \dfrac{r(l)}{(1+\delta)^j} \cdot \mathrm{P}(X_{t+1} \neq i \mid X_t = i)$.

Then there is a constant $c > 0$ such that for $T := \min\{t \geq 0 : X_t \leq a \mid X_0 \geq b\}$ it holds $\mathrm{P}(T \leq 2^{cl/r(l)}) = 2^{-\Omega(l/r(l))}$.

3 The OneMax Problem

In this section, we analyze the expected running time of the (1+1)-EA on OneMax under bit-wise noise (p, q). We prove in Theorems 3 and 4 that the expected running time is polynomial when $p = O(\log n/n) \vee pq = O(\log n/n^2)$; otherwise, it is super-polynomial. The results generalize that in [6,13], which only considered the case where $p = 1$ or $q = 1/n$.

Theorem 3 is proved by applying Lemma 1, which gives an upper bound on the running time of the (1+1)-EA solving noisy OneMax. Let x^k denote any solution with k 1-bits, and $f^{\mathrm{n}}(x^k)$ denote its noisy objective value, which is a random variable. Lemma 1 intuitively means that if the probability of recognizing the true better solution by noisy evaluation is large enough (i.e., Eq. (1)), the running time can be upper bounded. Note that in their original theorem (i.e., Theorem 5 in [6]), it requires that Eq. (1) holds with only $j = k$, but their proof actually also requires the property, i.e., $\forall j < k < n$, $\mathrm{P}(f^{\mathrm{n}}(x^k) < f^{\mathrm{n}}(x^{k+1})) \leq \mathrm{P}(f^{\mathrm{n}}(x^j) < f^{\mathrm{n}}(x^{k+1}))$. We have combined these two conditions in Lemma 1 by requiring Eq. (1) to hold with any $j \leq k$ instead of only $j = k$.

Lemma 1 ([6]). *If there is a constant $0 < c \leq \frac{1}{15}$ and some $2 < l \leq \frac{n}{2}$ such that*

$$\begin{aligned} &\forall j \leq k < n : \mathrm{P}(f^{\mathrm{n}}(x^j) < f^{\mathrm{n}}(x^{k+1})) \geq 1 - l/n; \\ &\forall j \leq k < n - l : \mathrm{P}(f^{\mathrm{n}}(x^j) < f^{\mathrm{n}}(x^{k+1})) \geq 1 - c(n-k)/n, \end{aligned} \tag{1}$$

then the (1+1)-EA optimizes f in expectation in $O(n \log n) + n2^{O(l)}$ iterations.

Theorem 3. *For the (1+1)-EA on OneMax under bit-wise noise (p, q), the expected running time is polynomial if $p = O(\log n/n) \vee pq = O(\log n/n^2)$.*

Proof. We use Lemma 1 to prove it. We analyze the probability $\mathrm{P}(f^{\mathrm{n}}(x^j) \geq f^{\mathrm{n}}(x^{k+1}))$ for any $j \leq k < n$. We consider two cases.

(1) $p = O(\log n/n)$. For some positive constant c_1, assume that $p \leq c_1 \log n/n$. It is easy to see that $f^{\mathrm{n}}(x^j) \geq f^{\mathrm{n}}(x^{k+1})$ implies that the fitness evaluation of x^j or x^{k+1} is affected by noise, whose probability is at most $2p$. Thus, we have $\mathrm{P}(f^{\mathrm{n}}(x^j) \geq f^{\mathrm{n}}(x^{k+1})) \leq 2p \leq 2c_1 \log n/n$.
(2) $pq = O(\log n/n^2)$. For some positive constant c_2, assume that $pq \leq c_2 \log n/n^2$. Note that $f^{\mathrm{n}}(x^j) \geq f^{\mathrm{n}}(x^{k+1})$ implies that at least one bit of x^j or x^{k+1} is flipped by noise, whose probability is at most $2p(1-(1-q)^n) \leq 2pqn \leq 2c_2 \log n/n$.

Combining the above two cases leads to $\mathrm{P}(f^{\mathrm{n}}(x^j) \geq f^{\mathrm{n}}(x^{k+1})) \leq \frac{2\max\{c_1,c_2\}\log n}{n}$. Let $l = 30\max\{c_1, c_2\}\log n$ and $c = 1/15$. Then, $\mathrm{P}(f^{\mathrm{n}}(x^j) \geq f^{\mathrm{n}}(x^{k+1})) \leq c \cdot \frac{l}{n}$, and it is easy to verify that the condition of Lemma 1 holds. Thus, by Lemma 1, the expected number of iterations is $O(n \log n) + n2^{O(\log n)}$, which implies that the expected running time is polynomial. □

Theorem 4 is proved by applying Lemma 2, which gives a lower bound on the running time of the (1+1)-EA solving noisy OneMax when the probability of making a right comparison due to noise is not large enough (i.e., Eq. (2)).

Lemma 2 ([6]). *If there is a constant $c \geq 16$ and some $l \leq n/4$ such that*

$$\forall n - l \leq k < n : \mathrm{P}(f^{\mathrm{n}}(x^k) < f^{\mathrm{n}}(x^{k+1})) \leq 1 - c(n-k)/n, \tag{2}$$

then the (1+1)-EA optimizes f in $2^{\Omega(l)}$ iterations with a high probability.

Theorem 4. *For the (1+1)-EA on OneMax under bit-wise noise (p, q), the expected running time is super-polynomial if $p = \omega(\log n/n) \wedge pq = \omega(\log n/n^2)$.*

Proof. We use Lemma 2 to prove it. We are to show that $\forall 3n/4 \leq k < n$, $\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) = \omega(\log n/n)$ by considering two cases of p.

(1) $p = \omega(\log n/n) \cap o(1)$. To make $f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})$, it is sufficient that $f^{\mathrm{n}}(x^k) = k$ and $f^{\mathrm{n}}(x^{k+1}) \leq k$. The former event happens with probability at least $1-p$, since it is sufficient that the noise doesn't happen. Thus, we have

$$\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) \geq (1-p) \cdot \mathrm{P}(f^{\mathrm{n}}(x^{k+1}) \leq k). \tag{3}$$

Then, we analyze $\mathrm{P}(f^{\mathrm{n}}(x^{k+1}) \leq k)$ by further considering two subcases.

(1a) $q \leq 1/n$. To make $f^{\mathrm{n}}(x^{k+1}) \leq k$, it is sufficient that exactly one 1-bit is flipped by noise. Thus,

$$\mathrm{P}(f^{\mathrm{n}}(x^{k+1}) \leq k) \geq p \cdot (k+1)q(1-q)^{n-1} \geq \omega(\log n/n) \cdot (1/e) = \omega(\log n/n),$$

where the second inequality is by $pq = \omega(\log n/n^2)$, $k \geq 3n/4$ and $q \leq 1/n$.

(1b) $q > 1/n$. Let Y denote a random variable such that $\mathrm{P}(Y = 0) = 1-q$ and $\mathrm{P}(Y = 1) = q$. Let $Y_1, Y_2, ..., Y_n$ denote random variables which are independent and have the same distribution as Y. Then, under the condition that the noise happens in evaluating x^{k+1}, we have

$$f^{\mathrm{n}}(x^{k+1}) = \sum_{j=1}^{k+1}(1-Y_j) + \sum_{j=k+2}^{n} Y_j = k+1 - \sum_{j=1}^{2k-n+2} Y_j - \sum_{j=2k-n+3}^{k+1} Y_j + \sum_{j=k+2}^{n} Y_j.$$

Note that $\sum_{j=k+2}^{n} Y_j - \sum_{j=2k-n+3}^{k+1} Y_j$ is the difference between the sum of the same number of Y_j, thus $\mathrm{P}(\sum_{j=k+2}^{n} Y_j - \sum_{j=2k-n+3}^{k+1} Y_j \leq 0) \geq 1/2$ due to symmetry. Then, we have

$$\mathrm{P}(f^{\mathrm{n}}(x^{k+1}) \leq k) \geq \mathrm{P}(\text{the noise happens}) \cdot \mathrm{P}\left(k+1-\textstyle\sum_{j=1}^{2k-n+2} Y_j \leq k\right) \cdot \frac{1}{2}$$
$$= \frac{p}{2} \cdot (1-(1-q)^{2k+2-n}) \geq \frac{p}{2} \cdot (1-(1-q)^{n/2}) \geq \frac{p}{2} \cdot (1-(1/e)^{1/2}) = \omega(\log n/n),$$

where the second inequality is by $k \geq 3n/4$ and the last is by $q > 1/n$.

Combining subcases (1a) and (1b) leads to $\mathrm{P}(f^{\mathrm{n}}(x^{k+1}) \leq k) = \omega(\log n/n)$. Thus, according to Eq. (3) and $p = o(1)$, it holds that for $3n/4 \leq k < n$, $\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) = \omega(\log n/n)$.

(2) $p = \Omega(1)$. Since $pq = \omega(\log n/n^2)$, it must hold that $q = \omega(\log n/n^2)$. We consider three subcases.

(2a) $q = \omega(\log n/n^2) \cap O(1/n)$. We have $\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) \geq \mathrm{P}(f^{\mathrm{n}}(x^k) = k) \cdot \mathrm{P}(f^{\mathrm{n}}(x^{k+1}) = k)$. To make $f^{\mathrm{n}}(x^k) = k$, it is sufficient that the noise happens but no bit is flipped by noise. To make $f^{\mathrm{n}}(x^{k+1}) = k$, it is sufficient that exactly one 1-bit is flipped by noise. Thus,

$$\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) \geq p(1-q)^n \cdot p(k+1)q(1-q)^{n-1} = \omega(\log n/n),$$

where the equality holds since $p = \Omega(1)$, $q = \omega(\log n/n^2) \cap O(1/n)$ and $k \geq 3n/4$.

(2b) $q = \omega(1/n) \cap O(\log n/n)$. We conduct the following analysis under the condition that both noise happens in evaluating x^k and x^{k+1}, whose probability is $p^2 = \Omega(1)$. We divide x^k into two parts: y^k and z^k, where y^k is a string with $(\log n - 1)$ 1-bits and one 0-bit, and z^k is a string with $(k - \log n + 1)$ 1-bits and $(n - k - 1)$ 0-bits. We also divide x^{k+1} into two parts: y^{k+1} and z^{k+1}, where y^{k+1} is a string with $(\log n)$ 1-bits, and z^{k+1} is a string with $(k + 1 - \log n)$ 1-bits and $(n-k-1)$ 0-bits. Let $mut(x)$ denote the string generated by flipping each bit of x with probability q, and let $|x|_1$ denote the number of 1-bits of a string x. Thus, we have $f^{\mathrm{n}}(x^k) = |mut(y^k)|_1 + |mut(z^k)|_1$ and $f^{\mathrm{n}}(x^{k+1}) = |mut(y^{k+1})|_1 + |mut(z^{k+1})|_1$. To make $f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})$, it is sufficient that $|mut(y^k)|_1 \geq \log n - 1$, $|mut(y^{k+1})|_1 = \log n - 1$ and $|mut(z^k)|_1 \geq |mut(z^{k+1})|_1$. Note that $\mathrm{P}(|mut(y^k)|_1 \geq \log n - 1) \geq (1-q)^{\log n - 1}$ since it is sufficient that all the 1-bits of y^k are not flipped; $\mathrm{P}(|mut(y^{k+1})|_1 = \log n - 1) = \log n \cdot q(1-q)^{\log n - 1}$ since exactly one 1-bit needs to be flipped. For z^k and z^{k+1}, they are two strings with the same number of 1-bits and 0-bits, and thus $\mathrm{P}(|mut(z^k)|_1 \geq |mut(z^{k+1})|_1) \geq 1/2$ due to symmetry. Then, we get

$$\begin{aligned}\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) &\geq p^2 \cdot (1-q)^{\log n - 1} \cdot \log n \cdot q(1-q)^{\log n - 1} \cdot (1/2) \\ &\geq \omega(\log n/n) \cdot (1 - 2\log n \cdot q) \geq \omega(\log n/n),\end{aligned}$$

where the second inequality is by $q = \omega(1/n)$ and Bernoulli's inequality, and the last is by $q = O(\log n/n)$.

(2c) $q = \omega(\log n/n)$. The analysis is similar to that of case (2b), except the division of x^k and x^{k+1}. Here, y^k is just a 0-bit, y^{k+1} is just a 1-bit, and z^k, z^{k+1} are two strings with k 1-bits and $(n-k-1)$ 0-bits. We similarly get

$$\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) \geq p^2 \cdot q \cdot (1/2) = \omega(\log n/n).$$

Combining cases (1) and (2) shows that $\forall\, 3n/4 \leq k < n$, $\mathrm{P}(f^{\mathrm{n}}(x^k) \geq f^{\mathrm{n}}(x^{k+1})) = \omega(\log n/n)$. We set the parameters in Lemma 2 as $c = 16$ and $l = b \log n$, where b is any positive constant. Thus, for any $n - l \leq k < n$, $\mathrm{P}(f^{\mathrm{n}}(x^k) < f^{\mathrm{n}}(x^{k+1})) = 1 - \omega(\log n/n) \leq 1 - c(n-k)/n$. By Lemma 2, the expected number of iterations is $2^{\Omega(l)} = n^{\Omega(b)}$. Since b can be any positive constant, the expected running time is super-polynomial. □

4 The LeadingOnes Problem

In this section, we consider the (1+1)-EA solving LeadingOnes under bit-wise noise (p, q). We prove in Theorems 5 and 6 that the expected running time is polynomial if $p = O(\log n/n^2) \vee pq = O(\log n/n^3)$, and super-polynomial if $p = \omega(\log n/n) \wedge pq = \omega(\log n/n^2)$. The results generalize that in [13], where p is fixed to 1 or q is fixed to $1/n$.

Theorem 5 is proved by applying the additive drift theorem (i.e., Theorem 1). We will use $\mathrm{LO}(x)$ to denote the number of leading 1-bits of a solution x.

Theorem 5. *For the (1+1)-EA on LeadingOnes under bit-wise noise (p, q), the expected running time is polynomial if $p = O(\log n/n^2) \vee pq = O(\log n/n^3)$.*

Proof. We use Theorem 1 to prove it. For some positive constant b, suppose that $p \leq b \log n/n^2$ or $pq \leq b \log n/n^3$. We construct a distance function as follows:

$$V(x) = (1 + c/n)^n - (1 + c/n)^{\mathrm{LO}(x)},$$

where $c = 12b \log n + 1$. It is easy to verify that $V(x) = 0$ iff $x \in \mathcal{X}^* = \{1^n\}$.

Then, we investigate $\mathbb{E}[\![V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x]\!]$ for any x with $\mathrm{LO}(x) = i < n$. Let $\mathrm{P}_{\mathrm{mut}}(x, x')$ denote the probability that x' is generated from x by mutation, and let $\mathrm{P}_{\mathrm{acc}}(x, x')$ denote the probability that the offspring solution x' is accepted by comparing with x, i.e., $\mathrm{P}_{\mathrm{acc}}(x, x') = \mathrm{P}(f^{\mathrm{n}}(x') \geq f^{\mathrm{n}}(x))$. We divide the drift into two parts: positive E^+ and negative E^-. That is,

$$\mathbb{E}[\![V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x]\!] = \mathrm{E}^+ - \mathrm{E}^-,$$

where

$$\mathrm{E}^+ = \textstyle\sum_{x':\mathrm{LO}(x')>i} \mathrm{P}_{\mathrm{mut}}(x, x') \cdot \mathrm{P}_{\mathrm{acc}}(x, x') \cdot (V(x) - V(x')),$$
$$\mathrm{E}^- = \textstyle\sum_{x':\mathrm{LO}(x')<i} \mathrm{P}_{\mathrm{mut}}(x, x') \cdot \mathrm{P}_{\mathrm{acc}}(x, x') \cdot (V(x') - V(x)).$$

Note that $V(x) > V(x')$ iff $\mathrm{LO}(x') > \mathrm{LO}(x) = i$, since the distance function V decreases with the number of leading 1-bits.

We first analyze the positive drift E^+. For any x' with $\mathrm{LO}(x') > i$,

$$V(x) - V(x') = (1 + c/n)^{\mathrm{LO}(x')} - (1 + c/n)^i \geq (1 + c/n)^i \cdot c/n. \tag{4}$$

To generate x' with $\mathrm{LO}(x') > i$ by mutating x, it needs to flip the $(i+1)$-th bit (which must be 0) of x and keep the i leading 1-bits unchanged. Thus, we have

$$\sum_{x':\mathrm{LO}(x')>i} \mathrm{P}_{\mathrm{mut}}(x,x') = \mathrm{P}(\mathrm{LO}(x') > i) = (1-1/n)^i\,(1/n) \geq 1/en. \tag{5}$$

To analyze $\mathrm{P}_{\mathrm{acc}}(x,x')$ for any x' with $\mathrm{LO}(x') > i$, we consider the opposite event that x' is rejected, i.e., $f^{\mathrm{n}}(x) > f^{\mathrm{n}}(x')$, which implies that $f^{\mathrm{n}}(x) \geq i+1$ or $f^{\mathrm{n}}(x') \leq i-1$. By the union bound, $\mathrm{P}(f^{\mathrm{n}}(x) > f^{\mathrm{n}}(x')) \leq \mathrm{P}(f^{\mathrm{n}}(x) \geq i+1) + \mathrm{P}(f^{\mathrm{n}}(x') \leq i-1) = pq(1-q)^i + p(1-(1-q)^i) = p - p(1-q)^{i+1}$, where the first equality is because $f^{\mathrm{n}}(x) \geq i+1$ iff the $(i+1)$-th bit of x is flipped by noise while the i leading 1-bits are not flipped; $f^{\mathrm{n}}(x') \leq i-1$ iff at least one of the i leading 1-bits of x' is flipped by noise. Then, we get

$$\begin{aligned} \mathrm{P}_{\mathrm{acc}}(x,x') &= 1 - \mathrm{P}(f^{\mathrm{n}}(x) > f^{\mathrm{n}}(x')) \geq 1 - p + p(1-q)^{i+1} \\ &\geq \max\{1-p, 1-pq(i+1)\} \geq 1 - b\log n/n^2 \geq 1/2, \end{aligned} \tag{6}$$

where the second inequality is by Bernoulli's inequality, the third inequality is by $p \leq b\log n/n^2$ or $pq \leq b\log n/n^3$, and the last holds with sufficiently large n. By applying Eqs. (4), (5) and (6) to E^+, we get

$$\mathrm{E}^+ \geq (1/en)\cdot(1/2)\cdot(1+c/n)^i \cdot (c/n) \geq (1+c/n)^i \cdot (c/6n^2).$$

We then analyze the negative drift E^-. For any x' with $\mathrm{LO}(x') < i$, we have

$$V(x') - V(x) = (1+c/n)^i - (1+c/n)^{\mathrm{LO}(x')} \leq (1+c/n)^i - 1. \tag{7}$$

To analyze $\mathrm{P}_{\mathrm{acc}}(x,x')$ for any x' with $\mathrm{LO}(x') < i$, we consider $f^{\mathrm{n}}(x) \leq f^{\mathrm{n}}(x')$, which implies that at least one bit of x or x' is flipped by noise. By the union bound, we have $\mathrm{P}_{\mathrm{acc}}(x,x') \leq 2p\,(1-(1-q)^n)$. Note that $1-(1-q)^n \leq \min\{qn, 1\}$ and $p \leq b\log n/n^2$ or $pq \leq b\log n/n^3$, we have

$$\mathrm{P}_{\mathrm{acc}}(x,x') \leq 2p\cdot\min\{nq,1\} = \min\{2npq, 2p\} \leq 2b\log n/n^2. \tag{8}$$

By applying Eqs. (7) and (8) to E^-, we get

$$\mathrm{E}^- \leq \sum_{x':\mathrm{LO}(x')<i} \mathrm{P}_{\mathrm{mut}}(x,x')\cdot(2b\log n/n^2)\cdot(1+c/n)^i \leq (1+c/n)^i\,(2b\log n/n^2).$$

By subtracting E^- from E^+, we get

$$\mathbb{E}[\![V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x]\!] \geq (1+c/n)^i\left(c/6n^2 - 2b\log n/n^2\right) \geq 1/6n^2,$$

where the last inequality is by $c = 12b\log n + 1$. Note that $V(x) \leq (1+\frac{c}{n})^n \leq e^c = en^{12b}$. By Theorem 1, we have $\mathbb{E}[\![\tau \mid \xi_0]\!] \leq 6n^2 \cdot en^{12b} = O\left(n^{12b+2}\right)$, thus the expected running time is polynomial. □

Next, we use the negative drift with self-loops theorem (i.e., Theorem 2) to prove a super-polynomial lower bound for $p = \omega(\log n/n) \wedge pq = \omega(\log n/n^2)$.

Theorem 6. *For the (1+1)-EA on LeadingOnes under bit-wise noise (p, q), the expected running time is super-polynomial if $p = \omega(\log n/n) \wedge pq = \omega(\log n/n^2)$.*

Proof. We use Theorem 2 to prove it. Let $X_t = |x|_0$ be the number of 0-bits of the solution x after t iterations of the (1+1)-EA. Let c by any positive constant. We consider the interval $[0, c\log n]$, i.e., $a = 0$ and $b = c\log n$ in Theorem 2.

Then, we analyze $\mathbb{E}[\![X_t - X_{t+1} \mid X_t = i]\!]$ for $1 \leq i < c\log n$. As in the proof of Theorem 5, we also divide the drift into positive E^+ and negative E^-:

$$\mathbb{E}[\![X_t - X_{t+1} \mid X_t = i]\!] = \mathrm{E}^+ - \mathrm{E}^-,$$

where

$$\mathrm{E}^+ = \textstyle\sum_{x':|x'|_0<i} \mathrm{P}_{\mathrm{mut}}(x, x') \cdot \mathrm{P}_{\mathrm{acc}}(x, x') \cdot (i - |x'|_0),$$
$$\mathrm{E}^- = \textstyle\sum_{x':|x'|_0>i} \mathrm{P}_{\mathrm{mut}}(x, x') \cdot \mathrm{P}_{\mathrm{acc}}(x, x') \cdot (|x'|_0 - i).$$

Note that we still use $\mathrm{P}_{\mathrm{mut}}(x, x')$ and $\mathrm{P}_{\mathrm{acc}}(x, x')$ to denote the probability that the offspring solution x' is generated and accepted, respectively.

To analyze E^+, we use a trivial upper bound 1 for $\mathrm{P}_{\mathrm{acc}}(x, x')$. Then, we have

$$\mathrm{E}^+ \leq \textstyle\sum_{x':|x'|_0<i} \mathrm{P}_{\mathrm{mut}}(x, x')(i - |x'|_0) \leq i/n,$$

where the second inequality is directly from the proof of Theorem 4.2 in [13].

For the negative drift E^-, we need to consider the increase of the number of 0-bits. We analyze the $n-i$ cases where only one 1-bit is flipped (i.e., $|x'|_0 = i+1$), which happens with probability $\frac{1}{n}(1-\frac{1}{n})^{n-1}$. Assume that $\mathrm{LO}(x) = k \leq n - i$. To analyze $\mathrm{P}_{\mathrm{acc}}(x, x') = \mathrm{P}(f^{\mathrm{n}}(x') \geq f^{\mathrm{n}}(x))$, we consider two cases.
(1) The j-th (where $1 \leq j \leq k$) leading 1-bit is flipped. To make $f^{\mathrm{n}}(x') \geq f^{\mathrm{n}}(x)$, we consider the j cases where $f^{\mathrm{n}}(x) = l$ and $f^{\mathrm{n}}(x') \geq l$ for $0 \leq l \leq j-1$. Note that $\mathrm{P}(f^{\mathrm{n}}(x) = l) = p(1-q)^l q$ and $\mathrm{P}(f^{\mathrm{n}}(x') \geq l) = 1 - p + p(1-q)^l$. Thus,

$$\mathrm{P}_{\mathrm{acc}}(x, x') \geq \textstyle\sum_{l=0}^{j-1} p(1-q)^l q \cdot \left(1 - p + p(1-q)^l\right).$$

If $p = o(1)$, $1-p+p(1-q)^l \geq \Omega(1)$; otherwise, $1-p+p(1-q)^l \geq \Omega(1)\cdot(1-q)^l$. Thus,

$$\mathrm{P}_{\mathrm{acc}}(x, x') \geq \Omega(1) \cdot pq \textstyle\sum_{l=0}^{j-1}(1-q)^{2l} \geq \Omega(1) \cdot p \cdot \left(1 - (1-q)^{2j}\right).$$

(2) One of the $(n-i-k)$ non-leading 1-bits is flipped, i.e., $\mathrm{LO}(x') = \mathrm{LO}(x) = k$. To make $f^{\mathrm{n}}(x') \geq f^{\mathrm{n}}(x)$, we consider the $k+1$ cases where $f^{\mathrm{n}}(x) = l$ and $f^{\mathrm{n}}(x') \geq l$ for $0 \leq l \leq k$. Thus, we have

$$\begin{aligned}\mathrm{P}_{\mathrm{acc}}(x, x') \geq & \textstyle\sum_{l=0}^{k-1} p(1-q)^l q \cdot \left(1 - p + p(1-q)^l\right) \\ & + \left(1 - p + p(1-q)^{k+1}\right) \cdot \left(1 - p + p(1-q)^k\right).\end{aligned}$$

If $p = o(1)$, obviously $\mathrm{P}_{\mathrm{acc}}(x, x') = \Omega(1)$. If $p = \Omega(1)$, we can derive that $\mathrm{P}_{\mathrm{acc}}(x, x') \geq \Omega(1) \cdot \left(1-(1-q)^{2k}\right) + \left(\Omega(1)(1-q)^{k+1}\right)^2$, and then further consider

two cases for q. If $q=\Omega(1)$, $\mathrm{P}_{\mathrm{acc}}(x,x')\geq\Omega(1)\cdot(1-(1-q)^{2k})\geq\Omega(1)\cdot(1-(1-q))=\Omega(1)$. If $q=o(1)$, $\mathrm{P}_{\mathrm{acc}}(x,x')\geq\Omega(1)(1-(1-q)^{2k})+\Omega(1)(1-q)^{2k}=\Omega(1)$. Thus,

$$\mathrm{P}_{\mathrm{acc}}(x,x')=\Omega(1).$$

Combining cases (1) and (2), we get

$$\mathrm{E}^{-}\geq(1/n)\,(1-1/n)^{n-1}\cdot\left(\Omega(1)\cdot p\textstyle\sum_{j=1}^{k}(1-(1-q)^{2j})+(n-i-k)\cdot\Omega(1)\right).$$

If $(1-q)^{2j}<1/2, 1-(1-q)^{2j}>1/2$; otherwise, $1-(1-q)^{2j}=(1-q)^{2j}((1+\frac{q}{1-q})^{2j}-1)\geq(1-q)^{2j}\frac{2qj}{1-q}\geq(1-q)^{2j}\cdot 2qj\geq qj$. Thus,

$$\mathrm{E}^{-}\geq\Omega(1/n)\cdot\left(p\textstyle\sum_{j=1}^{k}\min\{1/2,qj\}+n-i-k\right).$$

By subtracting E^{-} from E^{+}, we get

$$\mathbb{E}[\![X_t-X_{t+1}\mid X_t=i]\!]\leq i/n-\Omega(1/n)\cdot\left(p\textstyle\sum_{j=1}^{k}\min\{1/2,qj\}+n-i-k\right).$$

To investigate condition (1) of Theorem 2, we need to derive an upper bound on $\mathrm{P}(X_{t+1}\neq i\mid X_t=i)$. To make $X_{t+1}\neq i$, it is necessary that at least one bit of x is flipped and x' is accepted. We consider two cases: (1) at least one of the k leading 1-bits of x is flipped; (2) the k leading 1-bits of x are not flipped and at least one of the last $n-k$ bits is flipped. For case (1), $\mathrm{P}_{\mathrm{acc}}(x,x')\leq\min\{2npq,2p\}$ by Eq. (8). For case (2), $\sum_{x'}\mathrm{P}_{\mathrm{mut}}(x,x')\leq\frac{n-k}{n}$. Thus, we get

$$\mathrm{P}(X_{t+1}\neq i\mid X_t=i)\leq\min\{2npq,2p\}+(n-k)/n.$$

Now we compare $\mathbb{E}[\![X_t-X_{t+1}\mid X_t=i]\!]$ with $\mathrm{P}(X_{t+1}\neq i\mid X_t=i)$.
(1) $k<n/2$. We have

$$\begin{aligned}\mathbb{E}[\![X_t-X_{t+1}\mid X_t=i]\!]&\leq i/n-\Omega(1/n)\cdot(n-i-k)\\&=-\Omega(1)\leq-\Omega(1)\cdot\mathrm{P}(X_{t+1}\neq i\mid X_t=i),\end{aligned}$$

where the equality is by $i<c\log n$ and $k<n/2$.
(2) $k\geq n/2$. We first investigate $\sum_{j=1}^{k}\min\{1/2,qj\}$. If $q=o(1/n)$, we have $\sum_{j=1}^{k}\min\{1/2,qj\}\geq qk^2/2=\Omega(qn^2)$. If $q=\Omega(1/n)$, $\sum_{j=1}^{k}\min\{1/2,qj\}=\Omega(n)$. Thus, we have $\sum_{j=1}^{k}\min\{1/2,qj\}\geq\Omega(1)\cdot\min\{qn^2,n\}$. Then we get

$$\begin{aligned}\mathbb{E}[\![X_t-X_{t+1}\mid X_t=i]\!]&\leq i/n-\Omega(1/n)\cdot\left(p\cdot\min\{qn^2,n\}+n-i-k\right)\\&=i/n-\Omega(1)\cdot(\min\{pqn,p\}-i/n+(n-k)/n)\,.\end{aligned}$$

Note that $pq=\omega(\log n/n^2)$ and $p=\omega(\log n/n)$, thus $\min\{pqn,p\}=\omega(\log n/n)$. Furthermore, $i<c\log n$. Thus, we get

$$\begin{aligned}\mathbb{E}[\![X_t-X_{t+1}\mid X_t=i]\!]&\leq-\Omega(1)\cdot(\min\{pqn,p\}+(n-k)/n)\\&\leq-\Omega(1)\cdot\mathrm{P}(X_{t+1}\neq i\mid X_t=i).\end{aligned}$$

Combining the above two cases implies that condition (1) of Theorem 2 holds.

To investigate condition (2) of Theorem 2, we need to derive a lower bound on $\mathrm{P}(X_{t+1} \neq i \mid X_t = i)$. We consider the n cases where only one bit is flipped. We can directly follow the analysis for E^- to derive that

$$\mathrm{P}(X_{t+1} \neq i \mid X_t = i) \geq \Omega(1/n) \cdot \big(p\textstyle\sum_{j=1}^{k} \min\{1/2, qj\} + n - k\big).$$

The only difference is that we also consider flipping only one 0-bit, whose analysis is the same as that for flipping only one non-leading 1-bit. To make $|X_{t+1}–X_t| \geq j$, it is necessary that at least j bits of x are flipped and x' is accepted. We consider two cases: (1) at least one of the k leading 1-bits of x is flipped; (2) the k leading 1-bits are not flipped. For case (1), $\sum_{x'} \mathrm{P}_{\mathrm{mut}}(x, x') \leq \frac{k}{n}\binom{n-1}{j-1}\frac{1}{n^{j-1}}$ and $\mathrm{P}_{\mathrm{acc}}(x, x') \leq \min\{2npq, 2p\}$ by Eq. (8). For case (2), $\sum_{x'} \mathrm{P}_{\mathrm{mut}}(x, x') \leq (1-\frac{1}{n})^k\binom{n-k}{j}\frac{1}{n^j}$. Thus,

$$\begin{aligned}\mathrm{P}(|X_{t+1} - X_t| \geq j \mid X_t = i) &\leq \frac{k}{n}\binom{n-1}{j-1}\frac{\min\{2npq, 2p\}}{n^{j-1}} + \left(1-\frac{1}{n}\right)^k\binom{n-k}{j}\frac{1}{n^j} \\ &\leq (k/n) \cdot \min\{2npq, 2p\} \cdot (4/2^j) + ((n-k)/n) \cdot (2/2^j).\end{aligned}$$

By following the way of comparing $\mathbb{E}[\![X_t - X_{t+1} \mid X_t = i]\!]$ with $\mathrm{P}(X_{t+1} \neq i \mid X_t = i)$ in the above analysis, we can derive that

$$\mathrm{P}(|X_{t+1} - X_t| \geq j \mid X_t = i) \leq (O(1)/2^j) \cdot \mathrm{P}(X_{t+1} \neq i \mid X_t = i),$$

i.e., condition (2) of Theorem 2 holds with $\delta = 1$ and $r(l) = O(1)$.

The parameter l in Theorem 2 is $b-a = c\log n$. Thus, the expected running time is $2^{\Omega(c\log n)}$ (where $c > 0$ can be any constant), i.e., super-polynomial. □

5 Conclusion

In this paper, we analyze the running time of the (1+1)-EA solving OneMax and LeadingOnes under bit-wise noise (p, q). We derive the ranges of p, q for the running time being polynomial and super-polynomial, respectively. Our results complement previous analyses, which fix $p = 1$ or $q = 1/n$. Note that our analysis on LeadingOnes does not cover all the ranges of p, q. That is, the running time is not known for $p = \omega(\log n/n^2) \cap O(\log n/n) \wedge pq = \omega(\log n/n^3)$ and $p = \omega(\log n/n) \wedge pq = \omega(\log n/n^3) \cap O(\log n/n^2)$. This question has been partially addressed in the recent work [16]. We leave the full analysis as our future work.

References

1. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific, Singapore (2011)
2. Dang, D.C., Lehre, P.K.: Efficient optimisation of noisy fitness functions with population-based evolutionary algorithms. In: FOGA 2015, Aberystwyth, UK, pp. 62–68 (2015)

3. Droste, S.: Analysis of the (1+1) EA for a noisy OneMax. In: Deb, K. (ed.) GECCO 2004. LNCS, vol. 3102, pp. 1088–1099. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24854-5_107
4. Friedrich, T., Kötzing, T., Krejca, M., Sutton, A.: Robustness of ant colony optimization to noise. Evol. Comput. **24**(2), 237–254 (2016)
5. Friedrich, T., Kötzing, T., Krejca, M., Sutton, A.: The compact genetic algorithm is efficient under extreme gaussian noise. IEEE Trans. Evol. Comput. **21**(3), 477–490 (2017)
6. Gießen, C., Kötzing, T.: Robustness of populations in stochastic environments. Algorithmica **75**(3), 462–489 (2016)
7. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **127**(1), 57–85 (2001)
8. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments-a survey. IEEE Trans. Evol. Comput. **9**(3), 303–317 (2005)
9. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16544-3
10. Prügel-Bennett, A., Rowe, J., Shapiro, J.: Run-time analysis of population-based evolutionary algorithm in noisy environments. In: FOGA 2015, Aberystwyth, UK, pp. 69–75 (2015)
11. Qian, C., Yu, Y., Tang, K., Jin, Y., Yao, X., Zhou, Z.H.: On the effectiveness of sampling for evolutionary optimization in noisy environments. Evol. Comput. **26**(2), 237–267 (2018)
12. Qian, C., Yu, Y., Zhou, Z.H.: Analyzing evolutionary optimization in noisy environments. Evol. Comput. **26**(1), 1–41 (2018)
13. Qian, C., Bian, C., Jiang, W., Tang, K.: Running time analysis of the (1+1)-EA for OneMax and LeadingOnes under bit-wise noise. In: GECCO 2017, Berlin, Germany, pp. 1399–1406 (2017)
14. Qian, C., Bian, C., Yu, Y., Tang, K., Yao, X.: Analysis of noisy evolutionary optimization when sampling fails. In: GECCO 2018, Kyoto, Japan, pp. 1507–1514 (2018)
15. Rowe, J.E., Sudholt, D.: The choice of the offspring population size in the (1,λ) evolutionary algorithm. Theor. Comput. Sci. **545**, 20–38 (2014)
16. Sudholt, D.: On the robustness of evolutionary algorithms to noise: Refined results and an example where noise helps. In: GECCO 2018, Kyoto, Japan, pp. 1523–1530 (2018)

Fitness Landscape Modeling and Analysis

A Surrogate Model Based on Walsh Decomposition for Pseudo-Boolean Functions

Sébastien Verel[1(✉)], Bilel Derbel[2,3], Arnaud Liefooghe[2,3], Hernán Aguirre[4], and Kiyoshi Tanaka[4]

[1] Univ. Littoral Côte d'Opale, LISIC, 62100 Calais, France
verel@univ-littoral.fr
[2] Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL, 59000 Lille, France
[3] Inria Lille – Nord Europe, 59650 Villeneuve d'Ascq, France
[4] Faculty of Engineering, Shinshu University, Nagano, Japan

Abstract. Extensive efforts so far have been devoted to the design of effective surrogate models aiming at reducing the computational cost for solving expensive black-box continuous optimization problems. There are, however, relatively few investigations on the development of methodologies for combinatorial domains. In this work, we rely on the mathematical foundations of discrete Walsh functions in order to derive a surrogate model for pseudo-boolean optimization functions. Specifically, we model such functions by means of Walsh expansion. By conducting a comprehensive set of experiments on nk-landscapes, we provide empirical evidence on the accuracy of the proposed model. In particular, we show that a Walsh-based surrogate model can outperform the recently-proposed discrete model based on Kriging.

1 Introduction

Context. Black-box optimization refers to the situation where no specific properties nor hypothesis are known about the problem to be solved. Nothing but the objective value associated to a given (candidate) solution can be used by the optimization process. For example, black-box optimization problems can be found in engineering and multi-disciplinary design fields, and more broadly when the problem formulation involves some numerical simulations [1]. Hence, solving a black-box optimization problem consists in exploring a number of candidate solutions, based solely on the evaluation of their fitness value. When the cost of computing fitness values is time consuming, traditional black-box optimization techniques, such as evolutionary algorithms and metaheuristics can have a prohibitive computational cost. In this context, surrogate-assisted approaches, such as Kriging and the Efficient Global Optimization (EGO) approach [11], are methods of choice to 'predict' the quality of solutions without systematically computing their objective values.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 181–193, 2018.
https://doi.org/10.1007/978-3-319-99259-4_15

Motivation. Surrogates models, also called meta-models, have well-established foundations at the crossroad of optimization and machine learning [10]. Roughly speaking, a surrogate model can be viewed as an estimate of the function being optimized based solely of the points (learning data) sampled by the optimization process so far. This (cheap) estimate is then used to sample and evaluate new points that are hopefully beneficial for the optimization process, while significantly reducing the overall computational cost. Except in few recent works [1,13], most existing investigations from the literature on surrogates are with respect to the continuous domain. When turning to the combinatorial setting, that is when the decision variables are discrete, we can safely claim that the adaptation of existing techniques is relatively scarce [1], and the development of dedicated surrogate models is in its very infancy beginning. It is worth noticing that expensive combinatorial optimization problems are a natural outcome for real-world applications from complex scheduling or neural networks, among others [10].

Contribution. In this paper, we are interested in pushing a step toward the establishment of novel surrogate models for combinatorial optimization. We focus on the class of pseudo-boolean functions for which the solution space is the set of binary strings. Our work is based on the application of Walsh functions [16], which form a complete orthogonal set of functions, and share common mathematical properties with the trigonometric functions used in Fourier analysis. As such, we propose to represent a pseudo-boolean function as a finite decomposition of Walsh functions, which enables us to derive a new surrogate model for this class of optimization problems. Having the model established, we approximate its coefficients (hyper-parameters) using different optimization and machine learning techniques for linear regression, namely conjugate gradient (CG) and Least-Angle Regression (LARS). Using a comprehensive set of nk-landscapes [12], we first evaluate the accuracy of the model. We then conduct a comparative study with the recently-proposed Kriging surrogate model for combinatorial problems. Our experimental results allows us to show that the designed Walsh-based surrogate model is able to provide a highly accurate approximation of the considered instances, outperforming Kriging in a number of scenarios.

Outline. In Sect. 2, we provide an overview of the mathematical foundation of Walsh functions and related works. In Sect. 3, we describe the proposed Walsh-based model. In Sect. 4, we evaluate the accuracy of the model using nk-landscapes. In Sect. 5, we conclude the paper and discuss further research.

2 Walsh Functions: Background and Related Work

2.1 Walsh Functions Basics and Evolutionary Computation

Continuous Walsh Decomposition. Walsh functions [16] constitute an enumerable set of functions $\varphi_k : [0,1] \to \{-1,1\}$ which composes a normal and orthogonal basis of the Hilbert space $L^2([0,1])$. Like other basis of functions such

as trigonometric functions of the Fourrier basis, and although the Walsh functions are not continuous since their values is either -1 or 1, they can be used to decompose any function of the Hilbert space; see [16] for the mathematical conditions. More formally, for any integer $k \in \mathbb{N}$ with the binary representation $k = \sum_{j=0}^{\infty} k_j 2^j$ and $k_j \in \{0,1\}$, the Walsh function φ_k is defined for any (real-valued) $x \in [0,1]$ with a natural binary representation $x = \sum_{j=1}^{\infty} x_j 2^{-j}$ and $x_j \in \{0,1\}$, by $\varphi_k(x) = (-1)^{\sum_{j=0}^{\infty} k_j x_{j+1}}$. Over the interval $[0,1]$, the graphical representation of the Walsh functions can be viewed in a similar way than the cosine functions. Indeed, for all $x \in [0,1]$, $\varphi_0(x) = 1$ is the constant function, the values taken by φ_1 change from 1 to -1 at $x = 0.5$, and so on. The *orthogonality* of Walsh functions means that for any positive integers j and $k \in \mathbb{N}$, $\int_0^1 \varphi_j(x)\varphi_k(x)dx = \delta_{jk}$ where δ_{jk} is the Kronecker delta. Thus, for any function f from $L^2([0,1])$, and for any $x \in [0,1]$, we have that $f(x) = \sum_{k=0}^{\infty} w_k \varphi_k(x)$, where $w_k \in \mathbb{R}$ are the *coefficients* given by the projection of f on φ_k: $w_k = \int_0^1 f(t)\varphi_k(t)dt$. The *order* of a Walsh function φ_k, denoted by $o(\varphi_k)$, is defined by the number of binary digit equals to 1 in the binary representation of k. For example, the function of order 0 is φ_0, the functions of order 1 are φ_{2^p} for all integers $p \geqslant 0$, the functions of order 2 are $\varphi_{2^p+2^{p'}}$ for all pairs of integers $p \neq p' \geqslant 0$, and so on. While the previous discussion is with respect to a continuous function, similar considerations can be discussed for the discrete case of pseudo-boolean functions.

Discrete Walsh Decomposition and EAs. Tightly related to evolutionary algorithms (EA), discrete Walsh functions were considered by Bethke [2], a PhD student of J. Holland in the late seventies. This was further extended by Goldberg [8], Forrest and Mitchell [6] to offer a relevant theoretical framework on the properties of fitness functions related to the schemata theorem, and on deceptive functions in EAs. In this context, the Walsh functions are defined for any pseudo-boolean function as follows. For any integer $k \in [0, 2^n - 1]$ with the binary representation $k = \sum_{j=0}^{\infty} k_j 2^j$ and $k_j \in \{0,1\}$, the Walsh function $\varphi_k : \{0,1\}^n \to \{-1,1\}$ is defined for any binary string $x = (x_1, \ldots, x_j, \ldots, x_n) \in \{0,1\}^n$ as: $\varphi_k(x) = (-1)^{\sum_{j=0}^{n-1} k_j x_j}$. The so-defined (finite) set of discrete functions is a normal orthogonal basis for the space of pseudo-boolean functions. For any integer j, and $k \in [0, 2^n - 1]$, $\frac{1}{2^n} \sum_{x \in \{0,1\}^n} \varphi_j(x)\varphi_k(x) = \delta_{jk}$[1]. Therefore, any pseudo-boolean function $f : \{0,1\}^n \to \mathbb{R}$ can be written as a unique finite weighted sum of Walsh functions.

$$\forall x \in \{0,1\}^n,\ f(x) = \sum_{k=0}^{2^n-1} w_k.\varphi_k(x) \ \text{ s.t. } \ w_k = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x).\varphi_k(x) \qquad (1)$$

Schemata Theory. The average of fitness values over a schemata of order p can be computed with a subset of Walsh functions of lower orders [8]. In fact, let us

[1] Indeed, the matrix $(\varphi_k(x_j))_{jk}$ of dimension $2^n \times 2^n$ is a Hadamard matrix.

recall that a schemata is a hypercube of the binary space. Usually, a schemata is written with the alphabet $\{*, 0, 1\}$ where 0, and 1 give the fixed position bits of the hyperplane. The order of the schemata is the number of 0/1 in the string. For instance, the schemata $h = *01***0$ is a schemata of length 7, and of order 3. The average fitness of a schemata h is then $f(h) = \sum_{k \subset h} w_k.\varphi_k(x)$ where $k \subset h$ means that the 1 in the binary representation of k corresponds to 0 or 1 of the schemata. Hence, it is possible to design deceptive functions to challenge EAs [8].

Walsh Decomposition in Combinatorial Optimization. Besides their initial theoretical interest, there was recently a renewed interest to Walsh decomposition, which remains the subject of active research in the optimization community [9]. In particular, in the so-called *grey-box* optimization setting [3], standard problems such as nk-landscapes, or max-SAT are regarded as a decomposition of Walsh functions. Within such a perspective, the fitness value, the fitness distribution, or the best solution at a given Hamming distance is computationally fast to compute, hence enabling the design of effective and efficient optimization techniques. Additionally, the Walsh decomposition can be used to detect accurate crossover points, and to identify independent sub-space problems that lead to the solving of very large combinatorial optimization problems with an impressively reduced cost [4].

2.2 Surrogate Models for Combinatorial Optimization

Surrogate Models. A standard surrogate-assisted optimization framework consists in an iterative process, where at each iteration: (i) build a model on the basis of the solutions (learning data) evaluated so far at previous iterations, (ii) compute best (believed, predicted) solution(s) on the basis of the so-constructed (cheap) model, (iii) evaluate the so-chosen solution(s) using the real (expensive) black-box function f. Each of these three steps comes with different challenges and different techniques and tools to address and implement. In our work, we are interested in designing a surrogate model dedicated to pseudo-boolean functions. We thereby focus very specifically on the very first step of the aforementioned framework, that is, the definition and the building of a highly accurate model that can eventually be used as a substitute of the real function. It is worth noticing that this is of crucial importance towards the design of effective and efficient surrogate-assisted optimization techniques.

Discrete Surrogates. As summarized in [17], when looking at the previous works on surrogate models for discrete problems, a number of approaches can be found from basic to more specialized ones. In straightforward approaches, the discrete nature of variables is simply ignored, and standard machine learning techniques is applied on the vector data. In most sophisticated approaches, either the model is inherently discrete or a distance 'measure' between discrete solutions is used to leverage existing continuous models. The work presented in this paper falls in this last category, encompassing a number of noticeable techniques [1]. To cite

a few, in [13], it is shown how to leverage existing distance-based surrogates, by considering more general (not necessarily continuous) metric spaces. This idea is then illustrated using Radial Basis Function Networks (RBFN). Later in [18,19], a seemingly similar principle is adopted in order to derive a Kriging (Gaussian Process) like surrogate model. Kriging has the interesting feature of providing a measure of uncertainty when determining predictions. This can be used to calculate the Expected Improvement (EI) of a solution, which is then used as the main criteria to balance exploitation and exploration when sampling candidate solutions in the so-called Efficient Global Optimization (EGO) approach [11]. Such an EGO approach [19] is shown to outperform Kriging and RBFN [13] on a number of nk-landscapes [12] considered as difficult adversarial pseudo-boolean benchmark functions. In our work, we also validate empirically our model using nk-landscapes as a case study, while comparing to the Kriging approach considered as a baseline competitor. Notice that using the proposed Walsh model to sample promising points (as performed in EGO) is left for future work since our main goal is to investigate the accuracy of the Walsh model in correctly rendering the original expensive function.

3 Surrogate Model Based on Walsh Functions

The Walsh-Based Surrogate Model. Given a pseudo-boolean function $f : \{0,1\}^n \rightarrow \mathbb{R}$, we have a closed form decomposition of f using the set of Walsh functions φ_k, as given in Eq. (1). It is worth noticing that the functions φ_k are problem independent, and hence uniquely defined irrespective to f. The values of the coefficients w_k depend however on function f, as given in Eq. (1). They are here assumed to be unknown and black-box. Moreover, although the number of coefficients is in general exponential in n, there might exist a significantly large number of zero coefficients, namely, 2^n. For instance, for nk-landscapes, the number of non-zero coefficients is bounded by $n \times 2^{k+1}$ [9] and the maximum order is $k+1$. Hence, our idea is to consider an approximation of f using solely the Walsh functions of a constant order $d \ll n$ and using an estimate $\widehat{w}_k$ of the (unknown) coefficient w_k. More formally, we shall assume that the pseudo-boolean function f can be approximated by the following model constituting the core of our proposed surrogate model:

$$\forall x \in \{0,1\}^n,\ \hat{f}(x) = \sum_{k\ :\ o(\varphi_k) \leqslant d} \widehat{w}_k . \varphi_k(x) \tag{2}$$

Obviously, the previous equation is similar to standard (finite) Taylor series for continuous function expansion. The larger the order d, the better the approximation; and the better the quality of the estimate coefficients $\widehat{w}_k$, the more accurate the expansion. In the following, we shall focus on how to provide a good estimate of the Walsh coefficients, assuming that d is fixed to some constant. For clarity, the choice of the setting of the order d is discussed later on.

Model Approximation. Given the black-box nature of the pseudo-boolean function f, one idea would be to consider a sample of solutions for which we know the true f values. Let us assume given such a set, denoted $\mathcal{S}$. For now, we do not make any further assumption on $\mathcal{S}$. Then, the question is: find an estimate $\widehat{w}_k$ of the coefficients w_k using the data set $\{(x, f(x)) \mid x \in \mathcal{S}\}$. One answer to this question could be to simply use the mean as estimator by setting $\widehat{w}_k = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} f(x)\varphi_k(x)$. By a routine verification, we can show that the bias of the estimate is $\widehat{w}_k - w_k = \sum_{j \neq k} w_j \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \varphi_j(x)\varphi_k(x)$, which is to be interpreted as the degree of 'non-orthogonality' of the Walsh functions on $\mathcal{S}$. While being informative, such an estimate might be misleading, since it might be challenging to design a sample data set $\mathcal{S}$ verifying such properties w.r.t. Walsh functions. In the following, we discuss two techniques to estimate the Walsh coefficients required by the proposed surrogate.

Mean Squared Error Estimation Using Conjugate Gradient (CG). The Walsh decomposition of a pseudo-boolean function is a *linear model* where the predictors are the Walsh functions' values. As a consequence, classical methods for non-sparse and sparse approximation can be used to estimate the coefficients of the regression. Our first technique is based on a standard approach which consists in minimizing the mean squared error of the surrogate (linear) model with respect to data set $\mathcal{S}$. More formally,

$$mse(\widehat{w}) = \sum_{x \in \mathcal{S}} \left(\sum_{k \,:\, o(\varphi_k) \leqslant d} \widehat{w}_k . \varphi_k(x) - f(x) \right)^2 \tag{3}$$

We then find the coefficients $\widehat{w}^*$ minimizing Eq. (3), that is: $\widehat{w}^* = \text{argmin}_w mse(w)$. To solve this equation, we propose to use a non-sparse method, namely the conjugate gradient (CG) approach [14].

Least-Angle Regression (LARS) Coefficients Estimate. When the number of predictors in a linear regression is large, sparse techniques can be used to minimize the number of non-zero coefficients. Among others, lasso is one classical technique for sparse approximation [15]. In this work, we propose to use the least-angle regression (LARS) algorithm [5]. The LARS algorithm is in the same family of regularization/sparse methods and follows a forward stepwise selection regression mechanism. It has the major advantage of being computationally fast and effective when fitting high-dimensional data of relatively small size. Hence, it is a method of choice in our context since the number of Walsh functions of order d might be greater than the number of samples in the training data set $\mathcal{S}$.

Order Setting. Finally, we need to specify a value for the maximum order d to be set in the estimate. Intuitively, the larger the order, the larger the number of Walsh coefficients to be estimated, and hence the better the approximation. However, the larger the number of coefficients, the more difficult and time consuming their estimation using the previously-described techniques. Let n_d be the number of Walsh coefficients of order d. Then, we have that: $n_0 = 1$ and $n_d = n_{d-1} + \binom{n}{d}$ for $d > 0$. This makes the choice of large d values problematic. Nonetheless, we argue that the number of non-zero coefficients is typically much less than n_d and a value of d of at most 3, for which $n_d = O(n^3)$, should be sufficient for an accurate approximation of difficult functions, as supported by our empirical results. Table 1 shows the values of n_d for different values of n and d.

Table 1. Number of Walsh coefficients according to problem dimension n (columns), and order d (rows).

	10	15	20	25
0	1	1	1	1
1	11	16	21	26
2	46	121	211	326
3	176	576	1351	2626

4 Experimental Analysis

4.1 Experimental Setup and Methodology

Test Functions. As in previous studies [13,19], we consider nk-landscapes [12] as benchmark pseudo-boolean functions. For every binary string x of size n, $f(x)$ is defined as the average value of the *contributions* associated with each variable x_i. For every $i \in \{1, \ldots, n\}$, a component function $f_i \colon \{0,1\}^{k+1} \mapsto [0,1]$ assigns a real-valued contribution for every combination of x_i and its k *epistatic interactions* $(x_{i_1}, \ldots, x_{i_k})$. In other words, the individual contribution of a variable x_i to $f(x)$ depends on its value and on the values of $k < n$ other variables $(x_{i_1}, \ldots, x_{i_k})$. The function f is hence defined as follows: $f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x_i, x_{i_1}, \ldots, x_{i_k})$. The k epistatic interactions w.r.t. a variable x_i are set uniformly at random among the $(n-1)$ variables other than x_i [12]. The f_i values are uniformly distributed in $[0,1]$. It is important to remark that by increasing the number of epistatic interactions k from 0 to $(n-1)$, problem instances can be gradually tuned from smooth to rugged, which make nk-landscapes an abstract adversarial optimization benchmark that can eventually cover a wide range of (real-world) pseudo-boolean function classes, as commented further in the following.

Experimental Setup. We consider a comprehensive set of nk-landscapes with $n \in \{10, 15, 20, 25\}$, and $k \in \{0, 1, 2\}$. Notice that for $k = 0$, the function is linear which makes it easy to optimize. For $k = 1$, the function is quadratic which, informally speaking, falls in the same class than the widely studied Unconstrained Binary Quadratic Problem. For $k = 2$, every variable is in interaction with two other ones which, informally speaking, recalls the max-3-SAT problem. For every parameter combination $(4 \times 3 = 12)$, we generate 5 instances for which every competing model/algorithm is run for 5 independent runs. The reported results are over the $5 \times 5 = 25$ independent runs. All algorithms and experiments are implemented in R using standard machine learning and optimization packages.

Algorithm 1. Experimental procedure

Input: A test set Q
$S_0 \leftarrow \varnothing$;
for $t = 1, 2, \ldots, Max\_Budget$ **do**
  $x_t \leftarrow$ a solution generated uniformly at random;
  $S_t \leftarrow S_{t-1} \cup \{(x_t, f(x_t))\}$;
  $\widehat{f}_t \leftarrow$ build a surrogate model for f on the basis of (the training set) S_t;
  $\epsilon_t \leftarrow$ a measure of the quality of the accuracy of $\widehat{f}_t$ using the test set Q;
end

Validation Methodology. In our work, we focus on studying the accuracy of the Walsh expansion surrogate in providing a high fidelity approximation. Consequently, we follow the experimental procedure depicted in the template of Algorithm 1. First, we generate a (test) set Q of $N = 1000$ solutions generated uniformly at random (i.e., each bit is set to 0 or 1 with equal probability) which is used as input of our experimental procedure. For each instance, and for every iteration $t > 0$ of Algorithm 1, we generate uniformly at random a new solution x_t and evaluate its true fitness value $f(x_t)$. Next, we build a surrogate model $\widehat{f}_t$ using the (training) data set $S_t = S_{t-1} \cup \{x_t\}$. We then record an error measure (denoted ϵ_t) rendering the quality of the fit $(\widehat{f}_t, S_t)$ with respect to the (test) data set Q. This shall allow us to study the ability of the surrogate model to fit the real function as the size of the available sample data grows, that is, as the available budget in terms of (expensive) function evaluations is given. The maximum allowed budget is actually variable in the size of the considered nk-landscape.

As a baseline, we use Kriging [7] as a state-of-the-art discrete surrogate model, and the implementation provided in the R package CEGO[2]. The hyper-parameters of the Kriging model are set following [19]. The R package lars[6] is used for the implementation of LARS with default hyper-parameters: lasso technique for the cross-validation, and a fraction set to $s = 1$. As an error measure, we compute the mean absolute error (mae) and the mean squared error (mse) of $\widehat{f}$ w.r.t Q. When the proposed Walsh model is experimented, we additionally record the Walsh coefficients estimate $(\widehat{w}_k)$, and compute their R^2 coefficient.

4.2 Training with CG Versus LARS

First, we consider the accuracy of the model when using the two fitting techniques (CG and LARS) for estimating the Walsh coefficients (see Sect. 3). In Fig. 1, we show the evolution of the mean absolute error as a function of the size of the training set S_t, using nk-landscapes with $n = 20$ and $k = 1$. Notice that similar results holds for the mean squared error and are omitted due to lack of space.

[2] Packages CEGO and LARS on CRAN: https://cran.r-project.org.

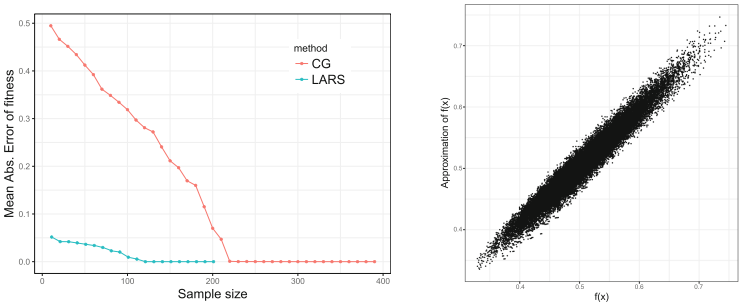


Fig. 1. Mean Absolute Error of fitness values using CG and Gradient techniques as a function of the training set size for $n = 20$, and $k = 1$ (left). Scatter plot (right) of the regression using LARS for $n = 15$, $k = 1$, and a random sample of size 60.

Figure 1 (left) shows that the Walsh model computed with LARS leads to a significantly better fit than the CG based technique, while requiring a sample of significantly lower size. Actually, the error of both methods converges to 0, with LARS being significantly faster. This means that it requires much fewer function evaluations to converge to a high fidelity Walsh approximate. In Fig. 1 (right), we show a scatter plot rendering the relative distribution of $\hat{f}(x)$ and $f(x)$ using the LARS for nk-landscapes with $n = 15$, and $k = 1$ trained on a random sample set of size 60, and tested on the whole search space. The quality of the regression visually approximates the original function for all fitness values. Indeed, the residues can be bounded by a constant independent of the fitness value: for any $x \in \{0,1\}^n$, $|\widehat{f}(x) - f(x)| = |\sum_k (\widehat{w}_k - w_k)\varphi_k(x)|$. Given that $|\varphi_k(x)| = 1$, we obtain the upper bound $|\widehat{f}(x) - f(x)| \leqslant \sum_k |\widehat{w}_k - w_k|$ which interestingly does not depend on x.

4.3 Walsh Versus Kriging

In Fig. 2, we show results comparing the proposed Walsh model to Kriging. Two main tightly related observations can be extracted. On the one hand, for $k = 0$, both surrogates consistently provide similar accuracy. However, as the test function is no more linear ($k = 1$ and $k = 2$), the difference is substantial in favor of the proposed Walsh surrogate. From a fitness landscape analysis perspective, higher values of k lead to more rugged (non-smooth) multi-modal functions. In this case, and although Kriging has a better accuracy with very few samples (very few function evaluations), the Walsh model is able to converge much faster to a high quality fit. On the other hand, the difference becomes even more substantial when scaling the dimension of the pseudo-boolean function. In fact, for the highest values of k and n, it is clear that the performance of Kriging drops very significantly, since it is not able anymore to provide a high accuracy within a reasonable budget. This is to contrast with the Walsh model, which converges to a zero absolute error within a few hundreds of function evaluations. The high quality of the Walsh surrogate can be explained by the fact that it

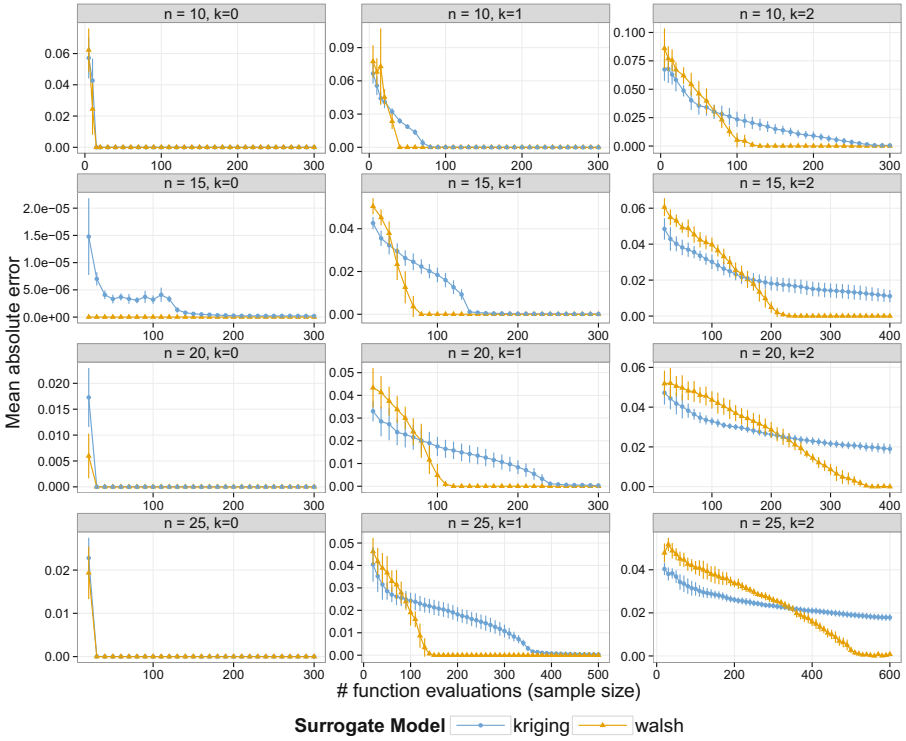


Fig. 2. Mean Absolute Error of fitness values on test set (1000 solutions) as a function of the size of the training set. $n \in \{10, 15, 20, 25\}$ (rows), $k \in \{0, 1, 2\}$ (columns), and the order of the Walsh expansion is $k + 1$.

is a deterministic model (for noise-free function) which provides a quasi-exact modeling of the pseudo-boolean function once the coefficient value estimates are close enough to their true values. This claim can be supported by a more focused analysis on the quality of the coefficients estimates, which is discussed in the next section while commenting on the impact of the Walsh expansion order.

4.4 Impact of the Walsh Expansion Order

In the previous results, the order of the Walsh decomposition was fixed to $d = k + 1$ where k is the number of the epistatic interaction in the considered nk-landscape. However, one might ask what happens if the order is fixed to a different value. This is what is depicted in Fig. 3, showing the coefficient of determination (R^2) to render the relative quality of the approximated coefficients. Notice that the exact values of the Walsh coefficients at any order are computed by the sum of Walsh decomposition of the component functions [9].

First, we can see that the R^2 converges relatively quickly to 1 when the order of the expansion is $k+1$. Hence, one can reasonably suggest that for other highly

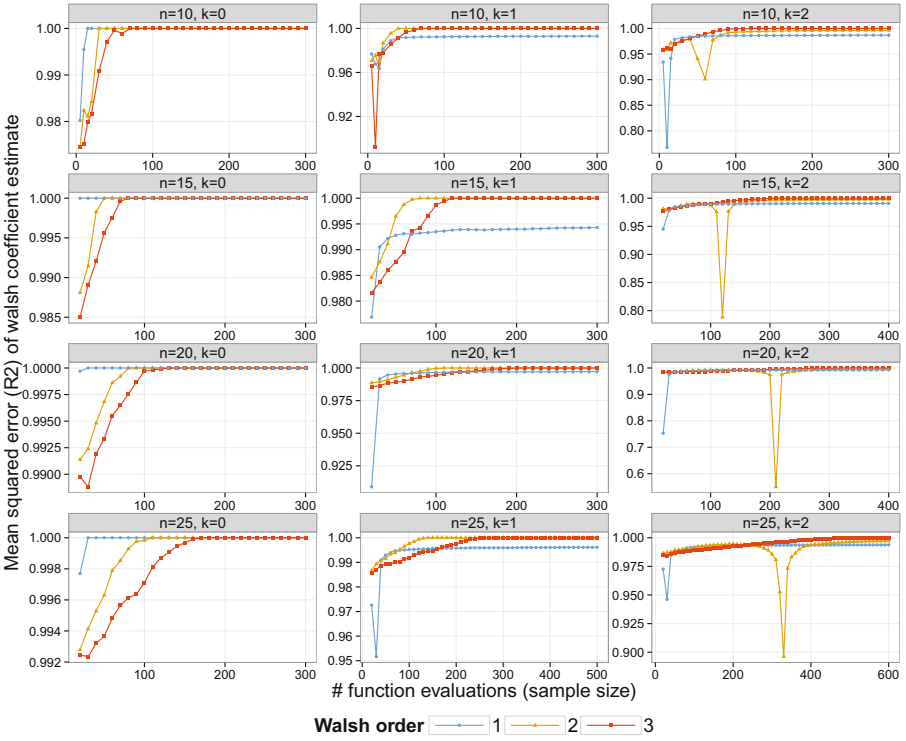


Fig. 3. R^2 of Walsh coefficients as a function of the size of the training set. $n \in \{10, 15, 20, 25\}$ (rows), $k \in \{0, 1, 2\}$ (columns), and the order of the expansion (color).

multi-modal functions, it might hold that only a restricted number of (low) order coefficients have non-zero values. In this case, only a restricted number of coefficient might impact the accuracy of the Walsh surrogate model, which would make it easier to build. Moreover, Fig. 3 shows that the coefficients of lower orders can still be accurately estimated although the value of the order chosen for the fit does not match with the maximum order of non-zero coefficients in the exact Walsh expansion. This suggests that for other highly multi-modal functions, it might hold that a small value of the order considered when fitting the Walsh surrogate is still sufficient to provide a high fidelity rendering of the original function. This property is of special interest since the lower the considered order, the lower the number of coefficient to be estimated, and the lower the cost of building the Walsh model. The cost of computing the surrogate model can in fact constitute a critical issue, especially if it comes to dominate the cost of the (expensive) function evaluation. In this respect, our LARS implementation of the Walsh surrogate was found to have a better CPU runtime compared against the Kriging implementation, by several orders of magnitude.

5 Conclusions

In this paper, we introduced a framework allowing to apply the Walsh functions basis in order to construct a novel discrete surrogate model for expensive pseudo-boolean functions, which is shown to be highly accurate on a set of nk-landscapes. Unlike previous distance/similarity based discrete surrogates, the proposed model is based on a deterministic pseudo-exact approximation. As such, it has some advantages and some shortcomings that, hopefully, will provide new scientific challenges. Besides, embedding the Walsh-based model within a conventional surrogate-assisted optimization framework would provide a highly effective approach to expensive (real-world) pseudo-boolean problems. In fact, not only building the Walsh surrogate is extremely fast compared against Kriging, but its solving to optimality using the recently-proposed grey-box optimization techniques [4] is fully plausible even for large-scale problems. This can for instance be a relevant alternative to the use of EGO-like selection criteria at a reduced cost. Additionally, generalizing the model to other non-necessarily pseudo-boolean functions, like permutation problems, would be a major advance. Finally, we believe that accommodating the deterministic nature of the model, for instance by taking into account the error in the coefficients approximation based on a probabilistic modeling might increase its potential in tackling a wide range of large optimization problems.

Acknowledgments. This research was partially conducted in the scope of the MODŌ International Associated Laboratory, and was partially supported by the French National Research Agency (BigMO project, ANR-16-CE23-0013-01).

References

1. Bartz-Beielstein, T., Zaefferer, M.: Model-based methods for continuous and discrete global optimization. Appl. Soft Comput. **55**, 154–167 (2017)
2. Bethke, A.D.: Genetic algorithms as function optimizers. Ph.D. thesis, University of Michigan (1980)
3. Chicano, F., Whitley, D., Alba, E.: Exact computation of the expectation surfaces for uniform crossover along with bit-flip mutation. Theor. Comput. Sci. **545**, 76–93 (2014)
4. Chicano, F., Whitley, D., Ochoa, G., Tinós, R.: Optimizing one million variable NK landscapes by hybridizing deterministic recombination and local search. In: GECCO, pp. 753–760 (2017)
5. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Ann. Stat. **32**(2), 407–499 (2004)
6. Forrest, S., Mitchell, M.: What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. Mach. Learn. **13**(2–3), 285–319 (1993)
7. Forrester, A., Keane, A.: Engineering Design via Surrogate Modelling: A Practical Guide. Wiley, Hoboken (2008)
8. Goldberg, D.E.: Genetic algorithms and walsh functions: Part I, a gentle introduction. Complex Syst. **3**(2), 129–152 (1989)

9. Heckendorn, R.B.: Embedded landscapes. Evol. Comput. **10**(4), 345–369 (2002)
10. Jin, Y.: Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evol. Comput. **1**(2), 61–70 (2011)
11. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Glob. Optim. **13**(4), 455–492 (1998)
12. Kauffman, S.A.: The Origins of Order. Oxford University Press, Oxford (1993)
13. Moraglio, A., Kattan, A.: Geometric generalisation of surrogate model based optimisation to combinatorial spaces. In: Merz, P., Hao, J.-K. (eds.) EvoCOP 2011. LNCS, vol. 6622, pp. 142–154. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20364-0_13
14. Shewchuk, J.R., et al.: An introduction to the conjugate gradient method without the agonizing pain (1994)
15. Tibshirani, R., Wainwright, M., Hastie, T.: Statistical Learning with Sparsity: The Lasso and Generalizations. Chapman and Hall/CRC, Boca Raton (2015)
16. Walsh, J.L.: A closed set of normal orthogonal functions. Am. J. Math. **45**(1), 5–24 (1923)
17. Zaefferer, M., Bartz-Beielstein, T.: Tabular survey: surrogate models in combinatorial optimization - version 5. Technical report, May 2017
18. Zaefferer, M., Stork, J., Bartz-Beielstein, T.: Distance measures for permutations in combinatorial efficient global optimization. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 373–383. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_37
19. Zaefferer, M., Stork, J., Friese, M., Fischbach, A., Naujoks, B., Bartz-Beielstein, T.: Efficient global optimization for combinatorial problems. In: GECCO (2014)

Bridging Elementary Landscapes and a Geometric Theory of Evolutionary Algorithms: First Steps

Marcos Diez García(✉) and Alberto Moraglio

Department of Computer Science, University of Exeter, Exeter EX4 4QF, UK
{md518,a.moraglio}@exeter.ac.uk

Abstract. Based on a geometric theory of evolutionary algorithms, it was shown that all evolutionary algorithms equipped with a geometric crossover and no mutation operator do the same kind of convex search across representations, and that they are well-matched with generalised forms of concave fitness landscapes for which they provably find the optimum in polynomial time [13]. Analysing the landscape structure is essential to understand the relationship between problems and evolutionary algorithms. This paper continues such investigations by considering the following challenge: develop an analytical method to recognise that the fitness landscape for a given problem provably belongs to a class of concave fitness landscapes. Elementary landscapes theory provides analytic algebraic means to study the landscape structure [15]. This work begins linking both theories to better understand how such method could be devised using elementary landscapes. Examples on the well known One Max, Leading Ones, Not-All-Equal Satisfiability and Weight Partition problems illustrate the fundamental concepts supporting this approach.

Keywords: Discrete nodal domain · Elementary landscape
Geometric crossover · Global concavity · Laplacian · P-structure

1 Introduction

Context and Challenge. Since its early days, evolutionary computing (EC) grew rapidly and diversely, but lacking a fundamental and coherent theory [9]. Despite recent progress, still much work needs to be done regarding unification [2,14]. Fitness landscapes have been helpful in that task since they define the search space structure by the fitness function of the problem and the search operators (i.e. the neighbourhood structure), thus linking three key elements of evolutionary algorithms (EAs): problems, algorithms and performance [14]. Particularly, EAs with geometric crossover but no mutation were proved to do an abstract form of convex search [11]. Moreover, this class of EAs is well-matched with globally concave (or convex, if minimising) landscapes, for which runtime exponentially better than pure random search can provably be guaranteed [13].

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 194–206, 2018.
https://doi.org/10.1007/978-3-319-99259-4_16

This paper focuses on fitness landscapes, and extends [13], aiming at developing an analytical method to recognise if a given fitness landscape of some combinatorial problem provably matches a concave (or convex) class of landscapes in general metric spaces. This method would provide insight to determine for which classes of problems and EAs good performance can be guaranteed. We propose elementary landscapes theory [15] as a fruitful choice to develop such method, because the following reasons suggest that certain classes of elementary landscapes relate to classes of (globally) concave landscapes and that elementary landscapes theory is at our disposal to analyse recombination spaces:

- Certain elementary landscapes have all local optima clustered in a single region of the search space. For instance, the landscape of the Linear Assignment problem. Moreover, for Boolean spaces with Hamming distance, it was shown that these elementary landscapes have a unique local optimum, thus coinciding with unimodal landscapes [7,15,18].
- Many combinatorial problems (such as Graph Coloring, Symmetric Travelling Salesman and Graph Bipartitioning) have elementary landscapes with local optima grouped only in a few clusters in the search space, suggesting an approximately globally concave (or convex) structure [8,18].
- Certain recombination spaces (e.g. induced by uniform recombination) are homomorphic to mutation spaces, meaning that their associated landscapes can be compared with each other and that elementary landscapes theory extends to recombination spaces [5,18].

Scope and Contributions. This paper begins linking elementary landscapes and the geometric framework to establish firm grounds to tackle the previous challenge. The major contributions of this work are:

1. Show that from both theories it is possible to restate recombination in terms of the other, that is enabling dual geometric and algebraic views, to justify that is not futile to propose elementary landscapes as an analytic tool.
2. Examples on four well-known combinatorial problems to clarify the connection between the structure of elementary landscapes and the globally concave classes of the geometric framework.

Although these contributions may appear only loosely connected, they complement each other as we will see in the following sections, and their connection will be further developed in future work.

Organisation. Section 2 introduces basic concepts about fitness landscapes. Sections 3 and 4 present key ideas of the geometric framework and elementary landscapes theory, respectively. Section 5 presents the main result of this study. Section 6 illustrates with examples the utility of elementary landscapes to analyse concave landscapes. Section 7 summarises this work and suggests future research.

2 Fitness Landscapes

In combinatorial problems, it is natural to formalise the search space with configuration set X in terms of a move operator $m : X^n \to X^k$ mapping a vector of n parents into a vector of k offspring [16]. Mutation, acting on a single parent, is naturally connected with a neighbourhood relation that can be described as a connected graph, which can be undirected or directed depending on whether the neighbourhood is symmetric or not. Recombination, acting on pairs of individuals, is non-trivial with different possible formalisations (e.g. using hypergraphs [5]). Besides neighbourhoods, graphs can also be described naturally using graphic distances (e.g. length of shortest paths), which directly relate to metric spaces and more generally fitness landscapes [16].

Definition 1. A *metric space* $\mathscr{M}$ is defined as the pair (X, d), where X is a set of configurations and $d : X \times X \to \mathbb{R}$ is a metric; such that $\forall x, y, z \in X$: (I) non-negativity: $d(x, y) \geq 0$; (II) identity: $d(x, y) = 0 \iff x = y$; (III) symmetry: $d(x, y) = d(y, x)$ and (IV) triangle inequality: $d(x, y) \leq d(x, z) + d(z, y)$.

Definition 2. A *fitness landscape* $\mathscr{F}$ can be defined as the pair $(\mathscr{M}, f)$, where $\mathscr{M}$ is a metric space, and $f : X \to \mathbb{R}$ is a real-valued fitness function that indicates the fitness of each configuration in X, the set of configurations in $\mathscr{M}$.

3 Geometric Framework

Moraglio proposed a general theory of EAs, independent of the problem and representation of solutions, solely based on an axiomatic definition of distance across metric spaces [10]. This allows to formalise mutation and crossover operators in terms of metric balls and geodesic intervals, respectively [20].

Definition 3. Let (X, d) be a metric space with configuration set X and metric d. Then, a *closed ball* centred at point $x \in X$ with radius $r \in \mathbb{R}_{\geq 0}$ is defined as $B_d[x; r] \coloneqq \{y \in X \mid d(x, y) \leq r\}$ and a *geodesic interval* or *metric segment* as $[x; y]_d \coloneqq \{z \in X \mid d(x, z) + d(z, y) = d(x, y)\}$, where $x, y \in X$ are called the *extremes* of the segment and $d(x, y)$ its *length*.

Normally, implementations of heuristic search methods define probabilistic search operators, thus a probability distribution over the configuration set. In other words, move operators that return a subset of the possible neighbours. The geometric framework takes this into account when defining geometric operators [10].

Definition 4. Let (X, d) be a metric space and $\mathrm{Im}[\star(\cdot)]$ denote the set of offspring produced with non-zero probability by an operator $\star$. Then, a unary operator $\mu_\varepsilon : X \to X$ is a *geometric ε-mutation* if $\forall x \in X \big(\mathrm{Im}[\mu_\varepsilon(x)] \subseteq B_d[x; \varepsilon]\big)$, where $\varepsilon \in \mathbb{R}_{\geq 0}$ is the smallest non-negative real number for which this condition holds. When ε is not specified, $\varepsilon = 1$ is assumed. And, a binary operator $\chi : X \times X \to X$ is a *geometric crossover* if $\forall x, y \in X \big(\mathrm{Im}[\chi(x, y)] \subseteq [x; y]_d\big)$.

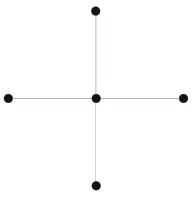
Apart from its benefits regarding unification and formal design of EAs across representations, a geometric definition provides insight on the landscape properties that cause EAs to perform possibly much better than pure random search [10]. Abstract convexity in general metric spaces is one such property [20], which describes the structure induced by geometric crossovers and admits well-behaved generalisations from traditional (Euclidean) concave functions [10,11].

Definition 5. Let a metric space $\mathscr{M} \coloneqq (X, d)$, with configuration set X and metric d, and a fitness function f define a fitness landscape $\mathscr{F} \coloneqq (\mathscr{M}, f)$. Then, $\forall x, y, z \in X$, $\mathscr{F}$ is *quasi-concave* if $z \in [x; y]_d$ and $f(z) \geqq \min\big(f(x), f(y)\big)$; and *average-concave* if $z \sim \mathcal{U}\big([x; y]_d\big)$ and $\mathbb{E}[f(z)] \geqq \big(f(x) + f(y)/2\big)$. Where $\mathcal{U}$ denotes the uniform probability distribution and $\mathbb{E}$ the expectation.

4 Elementary Landscapes Theory

Based on Grover's work [6], Stadler developed an algebraic theory of fitness landscapes known as elementary landscapes (ELs) [15]. Here, regular undirected[1] graphs induced by mutation neighbourhoods are formalised using the graph or mutation Laplacian matrix $\mathbf{L}_\mu$ defined by the graph's diagonal $\mathbf{D}$ and adjacency $\mathbf{A}$ matrices (Fig. 1):

$$-\mathbf{L}_\mu \coloneqq \mathbf{D} - \mathbf{A}. \tag{1}$$



$$-\mathbf{L}_\mu(S_4) = \mathbf{D}(S_4) - \mathbf{A}(S_4) = \begin{pmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 1. Star graph S_4 (left) and its graph Laplacian matrix (right).

To formalise recombination neighbourhoods in terms of hypergraphs [5], a new concept called 'P-structure' given by $(X, \mathcal{R})$ was introduced [18], where X is a finite set with power set $\mathcal{P}(X)$ and $\mathcal{R} : X \times X \to \mathcal{P}(X)$ maps a pair of parents into the set of possible offspring. By a hypergraph we mean a graph with vertex set X and hyperedge set $\mathscr{E} \coloneqq \{\mathcal{R}(x, y) \in \mathcal{P}(X) \mid \mathcal{R}(x, y) \neq \emptyset \wedge x, y \in X\}$. Furthermore, the following subclass of P-structures is defined to capture some desirable properties of recombination operators.

[1] Extensions for non-regular and non-symmetric neighbourhoods are possible, but regular and symmetric ones will be assumed here.

Definition 6. A P-structure $(X, \mathcal{R})$ is a *recombination structure* if $\forall x, y, z \in X$: (I) fix-point: $\mathcal{R}(x,x) = \{x\}$; (II) symmetry: $\mathcal{R}(x,y) = \mathcal{R}(y,x)$; (III) null-recombination: $\{x,y\} \subseteq \mathcal{R}(x,y)$; and (IV) size-monotonicity: if $z \in \mathcal{R}(x,y)$, then $|\mathcal{R}(x,z)| \leqq |\mathcal{R}(x,y)|$.

Laplacians $\mathbf{L}_{\mathcal{R}}$ for recombination P-structures can also be defined based on the identity matrix $\mathbf{I}$ and a generalised adjacency matrix $\mathbf{S}$ for hypergraphs [18]:

$$-\mathbf{L}_{\mathcal{R}} \coloneqq 2\,|X|\,\mathbf{I} - \mathbf{S}\ , \tag{2}$$

where the square matrix $\mathbf{S}$ has entries $\mathbf{S}_{zx} \coloneqq 2\sum_{y\in X} \mathbf{H}_{z,(x,y)}\,|\mathcal{R}(x,y)|^{-1}$, and the non-square binary incidence matrix $\mathbf{H}$ with entries $\mathbf{H}_{z,(x,y)}$ tells whether z is offspring of x and y under $\mathcal{R}$.

To better grasp the generalised adjacency matrix $\mathbf{S}$ in (2), one may consider its associated stochastic matrix $\mathbf{T}_{zx}$ describing the transition probabilities of a recombination-based random walk on a graph, where a father x is mated with a randomly chosen mother y to produce an offspring z that will be the father in the next recombination [18]. It is defined as $\mathbf{T}_{zx} \coloneqq \sum_{y\in X} t^z_{xy}\, p_y$, where p_y is the probability of choosing y from X, and t^z_{xy} is the probability of z given x and y. $\mathbf{T}_{zx}$, $\mathbf{S}$ and $\mathbf{L}_{\mathcal{R}}$ relate as: $\mathbf{T}_{zx} = \sum_{y\in X} t^z_{xy}\, p_y = \sum_{y\in X} \mathbf{H}_{z,(x,y)}\,|\mathcal{R}(x,y)|^{-1} * \frac{1}{|X|} = \frac{1}{2|X|}\mathbf{S}_{zx} = \frac{1}{2|X|}\mathbf{L}_{\mathcal{R}} + \mathbf{I}$. Although this assumes a uniform population and that all offspring occur with equal probability, it is possible to define weights for $\mathbf{S}$ to formalise more realistic population recombination-based search algorithms [17].

In the light of Grover's work and the characterisations above, Stadler defines a non-flat fitness landscape as *elementary* if its zero-averaged fitness function (as a column vector) $\tilde{f} \coloneqq f - \bar{f} = \left[f(x_1) - \bar{f}, f(x_2) - \bar{f}, \ldots, f(x_n) - \bar{f}\right]^{\mathrm{T}}$, with finite discrete domain $X = \{x_1, x_2, \ldots, x_n\}$ representing the vertex set of an underlying connected graph, is an eigenfunction of a (generalised) Laplacian matrix $\mathbf{L}$ of the graph: $\mathbf{L}\tilde{f} = \lambda\tilde{f}$; where $\lambda > 0$ is the eigenvalue and the constant $\bar{f} \coloneqq \frac{1}{|X|}\sum_{x\in X} f(x)$ is the average fitness of a configuration in X [8,15]. If a landscape f is not elementary, it can always be decomposed into a linear combination of ELs f_k called its *Fourier expansion*: $f = a_0 + \sum_{k>0}^{|X|-1} a_k f_k$, where scalars a_k are the Fourier coefficients, a_0 is the average fitness and the eigenfunctions f_k have corresponding eigenvalues λ_k (in increasing order and counting multiplicities, $0 = \lambda_0 < \lambda_1 \leqq \lambda_2 \leqq \cdots \leqq \lambda_{|X|-1}$) such that $\mathbf{L}f_k = \lambda_k f_k$.

ELs have interesting geometric properties. One of them, proved by Grover [6], is that all local maxima (minima) have fitness higher (lower) than or equal to the average fitness: $f(x_{min}) \leqq \bar{f} \leqq f(x_{max})$. This property is closely related to the idea of monotonic sequences of local optima (i.e. sequences of ever non-decreasing or non-increasing fitness) used, for instance, in the local optima networks model [19] to formalise the funnel structure of landscapes.

Another important aspect of ELs is their global structure. There are two major classes depending on the index p (ignoring multiplicities) of the eigenvalue: Fujiyama or single-peaked [7], and non-Fujiyama. Fujiyama are those ELs with the smallest non-zero eigenvalue (λ_1), that is $\mathbf{L}\tilde{f} = \lambda_{p=1}\tilde{f}$; characterised for

having all local optima clustered in a single region of the space [18]. In particular, for Boolean spaces with Hamming distance they have a unique global optimum [15]. Non-Fujiyama are those ELs for $p > 1$ with local optima clustered in more than one region; indeed, many combinatorial problems fall here with $p = 2$ (e.g. Max Cut) [8]. More formally, by clusters we mean *discrete nodal domains* [1] (Fig. 2). These are the maximally connected subgraphs induced by the vertex subsets $V_+ \coloneqq \{x \in V \mid f(x) \geqq 0\}$ and $V_- \coloneqq \{x \in V \mid f(x) \leqq 0\}$, denoting the weak positive and negative *nodal sets* respectively, for a given graph with vertex set V and eigenfunction f (i.e. the fitness function) on V. Similarly, strong nodal sets are defined using a strict inequality instead. Note that, for a zero-averaged function, V_+ and V_- induce subgraphs separating precisely above and below average configurations. Interestingly, the number of discrete nodal domains can be upper-bounded a priori, provided that we know p, using Proposition 1, generally, and the more sharp Proposition 2 specifically for Boolean spaces; which were proved in [1,4].

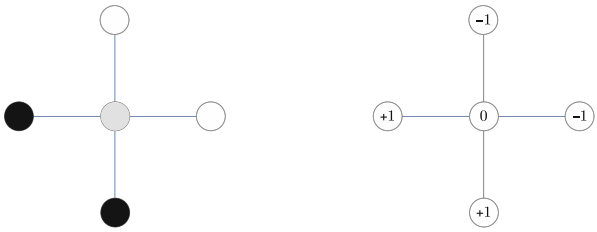


Fig. 2. An eigenfunction defined on the star graph S_4 (right), with eigenvalue $\lambda_1 = 1$ of the graph Laplacian, and its induced discrete nodal domains (left): two weak and four strong. Positive nodes in 'black', negative nodes in 'white' and zero nodes in 'grey'.

Proposition 1. Given a generalised Laplacian of a connected graph, any eigenfunction f_k corresponding to the k-th eigenvalue λ_k with multiplicity r has at most k weak and $k + r - 1$ strong nodal domains.

Proposition 2. For an eigenfunction f with eigenvalue $2p$, where p is its index ignoring multiplicities and N the number of vertices of the Boolean hypercube, the number $W(f)$ of weak nodal domains is upper-bounded as:

$$W(f) \leqq w_{N,p} = \begin{cases} p+1 & \text{if } p = 0 \text{ or } p = 1, \\ 2\left(1 + \sum_{k=0}^{p/2-1} \binom{N}{2k}\right) & \text{if } p \text{ is even}, \\ 2\left(1 + \sum_{k=0}^{\lfloor p/2 \rfloor - 1} \binom{N}{2k+1}\right) & \text{if } p \text{ is odd}. \end{cases} \tag{3}$$

5 Main Results

To analyse the landscapes induced by geometric crossovers (Definition 4) using ELs theory, it is necessary to prove that geometric crossovers belong to the class of recombination P-structures (Definition 6). Connecting both theories (Sects. 3 and 4) can help to develop dual geometric and algebraic views with which tackle the challenge of identifying generalised concave landscape classes using ELs theory, and give insight on explaining the good performance behind convex search (Sect. 1). Next, we prove that all geometric crossovers, regardless of the metric, are recombination P-structures.

Lemma 1. Let (X, d) be any metric space. Then, $\forall x, y \in X$ and any metric d, $(X, [x, y]_d)$ is a recombination P-structure.

Proof. We need to prove that metric segments fulfil the axioms of recombination P-structures.

(I) *Fix-point.* The only possible z in $[x; x]_d = \{z \in X \mid d(x, z) + d(z, x) = d(x, x)\}$ is exactly x. Therefore, $[x; x]_d = \{x\}$.
(II) *Symmetry.* $[x; y]_d = [y; x]_d$ follows immediately from the symmetry axiom of metric segments.
(III) *Null-recombination.* $\{x, y\} \subseteq [x; y]_d$ holds by definition, since the extremes x and y of the segment are always included in the segment.
(IV) *Size-monotonicity.* To prove that if $z \in [x; y]_d$, then $\left|\, [x; z]_d \,\right| \leqq \left|\, [x; y]_d \,\right|$, we can recall that all metric segments fulfil monotonicity [20]: $\forall x, y, z \in X$ if $z \in [x; y]_d$ then $[x; z]_d \subseteq [x; y]_d$. Therefore, it follows $\left|\, [x; z]_d \,\right| \leqq \left|\, [x; y]_d \,\right|$. □

Unfortunately, to prove equivalence in the other direction it is necessary to restrict recombination P-structures to those that precisely produce offspring $z \in \mathcal{R}(x, y)$ lying in the geodesic interval between parents x and y, since in general not all recombination P-structures fulfil this property [3].

Example 1. Consider $\mathcal{R}_\ell$ returning offspring lying on longest paths between parents. This can generate offspring beyond the geodesic interval between parents (i.e. in the extension ray [20]), e.g. $[000; 011]_{d_\mathrm{H}} \not\ni 111 \in \mathcal{R}_\ell(000, 011)$ where d_H is the Hamming distance. Indeed, extended line recombination, which generates offspring in the extension ray, is not a geometric crossover for any metric [12].

Definition 7. A recombination P-structure $(X, \mathcal{R})$, for a connected graph G with vertex set X, that $\forall x, y \in X$ fulfils the axiom $\mathcal{R}(x, y) \subseteq I(x, y)$, where $I(x, y)$ is the set of all shortest paths between x and y in G, is a *geometric recombination P-structure* denoted by $\mathcal{R}_\mathrm{g}$.

Theorem 1. Let (X, d) be any graphic metric space. Then, all geometric recombination P-structures $\mathcal{R}_\mathrm{g}$ are equivalent to all geometric crossovers χ defined on (X, d). That is, $\forall x, y \in X \big(\mathcal{R}_\mathrm{g}(x, y) = \chi(x, y)\big)$.

Proof. The proof follows immediately from Lemma 1, and Definitions 4 and 7 of geometric crossovers and geometric recombination P-structures respectively. □

Interestingly, all recombination P-structures fulfil the inbreeding properties (Theorem 2), that is breeding between close relatives, common to all geometric crossovers and independent of the metric used, which were proposed as a test for non-geometricity of crossovers: if any of them fails, then a crossover is not geometric [12].

Theorem 2. Let $(X, \mathcal{R})$ be any recombination P-structure. Then, $\mathcal{R}$ satisfies all inbreeding properties of geometric operators: purity, convergence and partition.

Proof. Purity: The recombination of one parent with itself can only produce the parent itself. Follows immediately from the fix-point axiom of recombination P-structures: $\forall x \in X$, $\mathcal{R}(x, x) = \{x\}$.

Convergence: The recombination of one parent with one offspring cannot produce the other parent of that offspring, unless the offspring and the second parent coincide. We want to prove that $\forall x, y, z, s \in X$ if $z \in \mathcal{R}(x, y)$ and $s \in \mathcal{R}(x, z)$, then $s = y \implies z = y$. Let $z \in \mathcal{R}(x, y)$ such that $z \neq y$, and $s \in \mathcal{R}(x, z)$. We want to show that actually $s \neq y$ too. From the size-monotonicity axiom of recombination structures we know:

$$|\mathcal{R}(x, z)| < |\mathcal{R}(x, y)|\,, \tag{4}$$

since $y \neq z \in \mathcal{R}(x, z)$. Besides, either $s = z$ or $s \neq z$. If $s = z$, then we know automatically that $s \neq y$, since $s = z \neq y$. If on the other hand $s \neq z$, then:

$$|\mathcal{R}(x, s)| < |\mathcal{R}(x, z)|\,. \tag{5}$$

If s was allowed to be y, then $|\mathcal{R}(x, s)| = |\mathcal{R}(x, y)|$, however by (5):

$$|\mathcal{R}(x, s)| = |\mathcal{R}(x, y)| < |\mathcal{R}(x, z)| \tag{6}$$

thus contradicting (4). Therefore, $s \neq y$ for the $s \neq z$ case. Consequently, the only possibility left for $s = y$ is that $z = y$.

Partition: If z is a child of x and y, then the recombination of x and z, and the recombination of y and z, cannot produce a common grandchild s other than z. Another way to phrase it is that both recombinations must produce two different grandchildren when they are not z. That is, we want to prove that $\forall x, y, z, s_1, s_2 \in X$ if $z \in \mathcal{R}(x, y)$, $s_1 \in \mathcal{R}(x, z)$ and $s_2 \in \mathcal{R}(z, y)$, then $s_1 \neq s_2$. Without loss of generality, let $z \in \mathcal{R}(x, y)$ such that $z \neq y$, and assume the opposite: $s_1 = s_2$. Then, we know by the convergence property that $s_1 \neq y$. Analogously, exchanging the roles of the parents x and y using the symmetry axiom of recombination structures, we know $s_2 \neq x$. Observe, however, that there are no restrictions in having $s_1 = x$ and $s_2 = y$. But, since we assumed $s_1 = s_2$, we find the following contradictions: $x = s_1 = s_2 \neq x$, and $y = s_2 = s_1 \neq y$. Therefore, $s_1 \neq s_2$. □

Since all recombination P-structures fulfil the inbreeding properties but only geometric recombination P-structures are equivalent to geometric crossovers (Theorem 1), from Theorem 2 we conclude the following.

Corollary 1. The inbreeding properties of geometric crossovers are necessary but not sufficient conditions to determine if a crossover is geometric.

This resolves the question of whether the inbreeding properties are sufficient or not for geometricity, which was left open in [12]. In the light of this result, it would be worth identifying additional inbreeding properties that when considered jointly would be sufficient to guarantee that a crossover is geometric.

6 Discrete Nodal Domains for Uniform Recombination in Boolean Spaces with Hamming Distance: Examples

Section 5 justified that proposing ELs theory to analyse the abstract concave classes of the geometric framework is not futile by showing that geometric crossovers are a specific case of recombination P-structures, and that certain recombination P-structures coincide with geometric crossovers; thus motivating a dual interpretation on landscapes induced by recombination. This section clarifies, with illustrative examples, how discrete nodal domains (Sect. 4) help analyse the structure of landscapes induced by uniform recombination in Boolean spaces with Hamming distance in particular, and how they link to global concavity. The examples considered are on two artificial problems, One Max and Leading Ones, and two NP-complete problems [6]:

Not-All-Equal Satisfiability (NAES). An instance is a set of clauses and its fitness is the number of satisfied clauses. A clause consists of three literals (i.e. a binary variable or its complement) not containing simultaneously a binary variable and its complement, and it is satisfied if there are at least two distinct literals such that one is 0 and the other is 1.

Weight Partition (WP). An instance is a vector of n weights w_i corresponding to objects o_i, for $1 \leqq i \leqq n$. The problem consists in finding an assignment $s_i \in \{-1, 1\}$ of weights such that $f(w_1, \ldots, w_n) = \left(\sum_{i=1}^{n} w_i s_i\right)^2$ is minimised.

Figure 3, obtained using `Mathematica`[2], summarises key aspects of the problems' landscape structure. Although the examples considered are for the three-dimensional Boolean hypercube embedded in the hypergraph induced by uniform recombination [18], we have observed equivalent results in higher dimensions.

Next, we see how the global landscape structure could be inferred analytically. For ELs this is possible by knowing the corresponding eigenvalues, since from their indexes we can tell whether they are Fujiyama or non-Fujiyama and also upper-bound the number weak nodal domains using Proposition 2 (Sect. 4). Then, we will explain why Leading Ones poses certain problems for this analytical approach.

[2] The reader is referred to the `Mathematica` notebook publicly available online at: https://github.com/marcosdg/ppsn-2018, for the examples details.

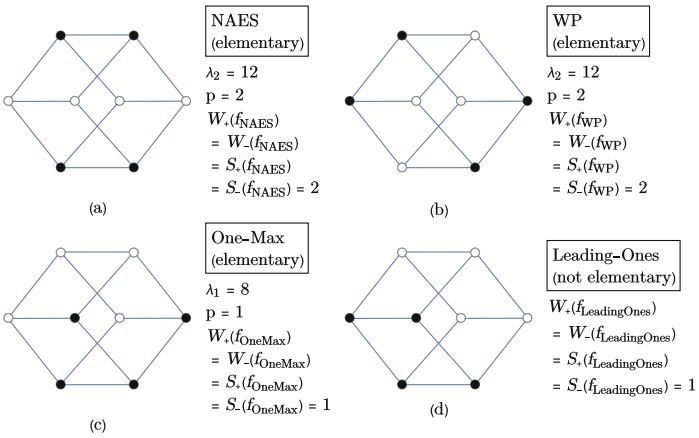


Fig. 3. Strongly positive ('black dots') and negative ('white dots') discrete nodal domains for the (a) Not-All-Equal satisfiability (NAES), (b) Weight Partitioning (WP), (c) One Max and (d) Leading Ones problems on the three-dimensional hypercube. W_+, S_+, W_- and S_- denote the number of weakly positive, strongly positive, weakly negative and strongly negative domains respectively; and p denotes the order of the eigenvalue (λ_p) for each of the eigenfunctions f_{NAES}, f_{WP}, f_{OneMax} and $f_{\mathrm{LeadingOnes}}$.

Eigenvalues of Elementary Landscapes. To find them we need to know the uniform recombination Laplacian $\mathbf{L}_{\mathcal{R}}$ for the three-dimensional hypercube (see Lemma C7 [18]): $(\mathbf{L}_{\mathcal{R}})_{xy} = 2(2^3)\mathbf{I}_{xy} - \left(2\,(3/2)^3\,3^{-d_{\mathrm{H}}(x,y)}\right)$. We observe that this Laplacian has eigenvalues $\{0, 8, 8, 8, 12, 12, 12, 14\}$ (in increasing order and starting at index 0); and that One Max is elementary for $\lambda_1 = 8$, and NAES and WP for $\lambda_2 = 12$: $\mathbf{L}_{\mathcal{R}}\tilde{f}_{\mathrm{OneMax}} = 8 * \tilde{f}_{\mathrm{OneMax}}$, $\mathbf{L}_{\mathcal{R}}\tilde{f}_{\mathrm{NAES}} = 12 * \tilde{f}_{\mathrm{NAES}}$ and $\mathbf{L}_{\mathcal{R}}\tilde{f}_{\mathrm{WP}} = 12 * \tilde{f}_{\mathrm{WP}}$.

One Max, NAES and WP. From the eigenvalues we know that One Max is a Fujiyama EL ($p = 1$), that is with a unique global optimum and single-peaked. This agrees with the fact that $f_{\mathrm{OneMax}}(x_1, \ldots, x_n) \coloneqq \sum_{i=1}^{n} x_i$ is a linear pseudo-Boolean function whose Fourier expansion using Walsh functions has only non-zero terms in the 0-th and 1-st orders thus elementary for $p = 1$. Whereas NAES and WP are non-Fujiyama ELs of order $p = 2$. These results are a consequence of Corollaries 2 and 3 in [18], proving that eigenfunctions of the mutation Laplacian (1) are also eigenfunctions of the uniform recombination Laplacian (2). Besides, one notes that the eigenvalue ($\lambda_2 = 12$) for NAES and WP does not coincide with the eigenvalue ($\lambda_2 = 4$) obtained for the mutation Laplacian [8]. Nevertheless, that is expected: two spaces may have identical eigenfunctions but different eigenvalues, possibly affecting the correlation between fitness values (i.e. ruggedness) of the landscape [18]. Thus, two landscapes may have identical number of nodal domains (i.e. same global structure) and different ruggedness.

Number of Weak Nodal Domains. Proposition 2 provides upper-bounds for the number of weak nodal domains; for One Max, NAES and WP we have: $W_+(\tilde{f}_{\text{OneMax}}) + W_-(\tilde{f}_{\text{OneMax}}) = 1 + 1 \leqq w_{2^3,1} = 2$, $W_+(\tilde{f}_{\text{NAES}}) + W_-(\tilde{f}_{\text{NAES}}) = 2 + 2 \leqq w_{2^3,2} = 4$ and $W_+(\tilde{f}_{\text{WP}}) + W_-(\tilde{f}_{\text{WP}}) = 2 + 2 \leqq w_{2^3,2} = 4$. This means that One Max has two weak nodal domains corresponding to two major clusters of local optima, separated by the hyperplane described by the average fitness. NAES and WP have at most four weak nodal domains and thus we cannot expect more than four clusters. Particularly, for $p = 1$ and $p = 2$, the upper-bounds remain constant regardless of the dimension (Table 4.1 in [1]).

Leading Ones. We observed in `Mathematica` that Leading Ones has the same number of discrete nodal domains as One Max by computing them. From Fig. (3d) one may think that Leading Ones is an EL for $p = 1$; however, it is not elementary neither for recombination nor mutation neighbourhoods, but a sum of n elementary landscapes. The reason is that $f_{\text{LeadingOnes}}(x_1, \ldots, x_n) \coloneqq \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$ is a k-bounded pseudo-Boolean function that is a sum of Walsh functions where the bitwise non-linearity is at most k bits. For Leading Ones $k = n$, thus the highest order is $p = n$ (i.e. the bit string length). This means that we cannot use Propositions 1 and 2 to know a priori the number of discrete nodal domains, since in general they do not hold when the landscape is a sum of ELs. Finding those cases where they hold is an open problem [1].

Concluding, discrete nodal domains are a promising analytic tool with which tackle the challenge of identifying landscape classes, because they capture key information of the landscape structure and can be related to ELs via eigenvalues and eigenfunctions. Then, what the abstract concave classes would correspond to? Previous observations suggest Fujiyama ELs ($p = 1$), since they overlap with single-peaked landscapes. For instance, One Max can be shown to be average-concave [13] and we now know that One Max is Fujiyama. Further research will confirm whether this intuition is correct.

7 Conclusions

The geometric framework has been successful in finding proper subclasses of EAs and fitness landscapes with provably polynomial runtime guarantees, however missing analytical means for landscape structure analysis. Besides, ELs theory provides a fine-grained analysis of the landscape structure in terms of its spectrum (i.e. eigenvalues and eigenfunctions) and discrete nodal domains, but lacking EAs dynamics modelling and runtime results. The ultimate goal of this research is precisely unifying these two frameworks, aiming at developing complementary geometric and algebraic views on fitness landscape analysis, to better understand when and why EAs perform well.

This paper took the first steps towards such goal by putting together for the first time both frameworks and highlighting their key ideas. First, proving that all geometric crossovers can be conceived as recombination P-structures, and

that there exists a specific subclass of these that match geometric crossovers (Sect. 5). Then, illustrating and clarifying with examples how discrete nodal domains could help to identify concave landscape classes for problems that are elementary, by analysing their spectrum (Sect. 6). The next important steps would be to formalise the concave classes in terms of discrete nodal domains and convex search in ELs theory. Also, it would be worth researching whether landscape funnels can be formalised in terms of discrete nodal domains.

References

1. Bıyıkoğlu, T., Leydold, J., Stadler, P.F.: Laplacian Eigenvectors of Graphs: Perron-Frobenius and Faber-Krahn Type Theorems. Lecture Notes in Mathematics, vol. 1915. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73510-6
2. Borenstein, Y., Moraglio, A. (eds.): Theory and Principled Methods for the Design of Metaheuristics. Natural Computing Series. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-33206-7
3. Changat, M., et al.: Topological Representation of the Transit Sets of k-Point Crossover Operators (2017). arXiv:1712.09022
4. Davies, E.B., Gladwell, G.M., Leydold, J., Stadler, P.F.: Discrete nodal domain theorems. Linear Algebra Appl. **336**(1), 51–60 (2001)
5. Gitchoff, P., Wagner, G.P.: Recombination induced hypergraphs: a new approach to mutation-recombination isomorphism. Complexity **2**(1), 37–43 (1996)
6. Grover, L.K.: Local search and the local structure of NP-complete problems. Oper. Res. Lett. **12**(4), 235–243 (1992)
7. Kauffman, S.A.: The Origins of Order: Self-organization and Selection in Evolution. Oxford University Press, New York (1993)
8. Klemm, K., Stadler, P.F.: Rugged and elementary landscapes. In: Borenstein, Y., Moraglio, A. (eds.) Theory and Principled Methods for the Design of Metaheuristics. NCS, pp. 41–61. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-33206-7_3
9. Menon, A. (ed.): Frontiers of Evolutionary Computation, Genetic Algorithms and Evolutionary Computation, vol. 11. Springer, New York (2004). https://doi.org/10.1007/b116128
10. Moraglio, A.: Towards a Geometric Unification of Evolutionary Algorithms. Doctoral thesis, University of Essex, Essex, UK, November 2007
11. Moraglio, A.: Abstract convex evolutionary search. In: Proceedings of the 11th Workshop on Foundations of Genetic Algorithms, FOGA 2011, pp. 151–162. ACM, Schwarzenberg (2011)
12. Moraglio, A., Poli, R.: Inbreeding properties of geometric crossover and non-geometric recombinations. In: Stephens, C.R., Toussaint, M., Whitley, D., Stadler, P.F. (eds.) FOGA 2007. LNCS, vol. 4436, pp. 1–14. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73482-6_1
13. Moraglio, A., Sudholt, D.: Principled design and runtime analysis of abstract convex evolutionary search. Evol. Comput. **25**(2), 205–236 (2017)
14. Richter, H., Engelbrecht, A. (eds.): Recent Advances in the Theory and Application of Fitness Landscapes. ECC, vol. 6. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-41888-4

15. Stadler, P.F.: Towards a theory of landscapes. In: López-Peña, R., Waelbroeck, H., Capovilla, R., García-Pelayo, R., Zertuche, F. (eds.) Complex Systems and Binary Networks. Lecture Notes in Physics, vol. 461, pp. 78–163. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0103571
16. Stadler, P.F.: Fitness landscapes. In: Lässig, M., Valleriani, A. (eds.) Biological Evolution and Statistical Physics. Lecture Notes in Physics, vol. 585, pp. 183–204. Springer, Boston (2002). https://doi.org/10.1007/0-387-28356-0_19
17. Stadler, P.F., Seitz, R., Wagner, G.P.: Population dependent fourier decomposition of fitness landscapes over recombination spaces: evolvability of complex characters. Bull. Math. Biol. **62**(3), 399–428 (2000)
18. Stadler, P.F., Wagner, G.P.: Algebraic theory of recombination spaces. Evol. Comput. **5**(3), 241–275 (1998)
19. Thomson, S.L., Daolio, F., Ochoa, G.: Comparing communities of optima with funnels in combinatorial fitness landscapes. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 377–384. ACM, New York (2017)
20. van de Vel, M.L.J.: Theory of Convex Structures. North-Holland Mathematical Library. North-Holland (1993)

Empirical Analysis of Diversity-Preserving Mechanisms on Example Landscapes for Multimodal Optimisation

Edgar Covantes Osuna(✉) and Dirk Sudholt

University of Sheffield, Sheffield S1 4DP, UK
{ecovantes1,d.sudholt}@sheffield.ac.uk

Abstract. Many diversity-preserving mechanisms have been developed to reduce the risk of premature convergence in evolutionary algorithms and it is not clear which mechanism is best. Most multimodal optimisation problems studied empirically are restricted to real-parameter problems and are not accessible to theoretical analysis, while theoreticians analyse the simple bimodal function TwoMax. This paper looks to narrow the gap between both approaches. We perform an extensive empirical study involving 9 common diversity mechanisms on Jansen-Zarges multimodal function classes (Jansen and Zarges, PPSN 2016) that allow to control important problem features while still being amenable to theoretical analysis. This allows us to study functions with various degrees of multimodality and to explain the results in the light of previous theoretical works. We show which mechanisms are able to find and maintain a large number of distant optima, escape from local optima, and which fail to locate even a single peak.

Keywords: Diversity-preserving mechanisms
Evolutionary algorithms · Multimodal optimisation · Empirical study
Theory

1 Introduction

Many optimisation problems are multimodal, and finding global optima or high-quality local optima can become a challenge for any optimisation algorithm [15,17]. Evolutionary algorithms (EAs) are well suited to dealing with multimodal problems due to their use of a population. A diverse population can explore several hills in the fitness landscape simultaneously and offer several good solutions to the user, a feature desirable for decision making, in multi-objective

The authors would like to thank the Consejo Nacional de Ciencia y Tecnología - CONACYT (the Mexican National Council for Science and Technology) for the financial support under the grant no. 409151 and registration no. 264342.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 207–219, 2018.
https://doi.org/10.1007/978-3-319-99259-4_17

optimisation and in dynamic optimisation. However, a major difficulty when applying EAs is that the population may converge to a sub-optimal individual before the fitness landscape is explored properly.

Many diversity-preserving mechanisms have been developed to reduce the risk of such premature convergence, including fitness sharing, clearing, avoiding duplicates, fitness diversity, crowding methods, restricted tournament selection, and many others [4,7,16,18]. These mechanisms seek to enable EAs to visit many different regions of the search space and generate solutions that differ from those seen before [10]. Given such a variety of mechanisms to choose from, it is often not clear which mechanism is the best choice for a particular problem.

Previous empirical analyses have considered real-parameter multimodal optimisation problems [5] like the 4 one-dimensional, five-peaked, sinusoidal, multimodal functions called M_{1-4} defined in [11, Sect. 5.3]. The single variable x is restricted to the real-value range $[0, 1]$ encoded using binary representation and decoded by interpreting the bit string as unsigned binary integer and dividing it by $2^n - 1$, where n is the length of the bit string. Other studies used Gray codes [15]. The drawback of real-valued encodings is that it is not obvious how phenotypic features such as local optima appear in genotype space; for example what Hamming distance local optima have and how likely it is that mutation jumps from one basin of attraction to another. This makes the analysis of the population dynamics a very challenging task for theoretical analysis.

Previous theoretical studies that considered multimodal problems [1,2,6,13] compared the expected running time of different diversity mechanisms to find both global optima on the bimodal function $\textsc{TwoMax}(x) := \{\sum_{i=1}^{n} x_i, n - \sum_{i=1}^{n} x_i\}$ that has a straightforward mapping between genotypes (bit strings) and phenotypes (number of 1-bits). TwoMax consists of two different symmetric slopes (or branches) ZeroMax and OneMax with 0^n and 1^n as global optima, respectively, and the goal is to evolve a population that contains both optima[1]. This is challenging as the two optima have the maximum possible Hamming distance. Studying TwoMax led to insights into the capabilities and weaknesses of various diversity mechanisms (see Sect. 2 and Sudholt's survey [19]), however a question left open is how diversity mechanisms deal with many local optima.

Jansen and Zarges [9] addressed the need for more general classes of functions for multimodal optimisation for both empirical and theoretical analysis by defining multimodal landscapes with straightforward binary encodings (see Sect. 3). We seek to narrow the gap between theory and practice by performing an empirical study on the Jansen-Zarges multimodal function classes, complementing existing rigorous theory for TwoMax [1,2,6,13] with empirical results of more complex functions with multiple different peaks, slopes and heights. The main goal is to provide insights into the working principles of these mechanisms by testing their ability to find and maintain many local optima in the population as well as their ability to escape from local optima with different basins of

[1] In [6] an additional fitness value for 1^n was added to distinguish between a local optimum 0^n and a unique global optimum 1^n. The discussion of previous work from [6] is adapted to a TwoMax with two optima [1,2,13] (see Table 1 and [19]).

Algorithm 1. (μ+1) EA

1: Initialise P with μ individuals chosen uniformly at random.
2: **while** stopping criterion **not** met **do**
3: Choose $x \in P$ uniformly at random.
4: Create y by flipping each bit in x independently with probability $1/n$.
5: Choose $z \in P$ uniformly at random from all individuals with worst fitness in P.
6: **if** $f(y) \geq f(z)$ **then** $P = P \setminus \{z\} \cup \{y\}$.

attraction. We use previous theoretical results to inform the choice of algorithm parameters and to discuss in how far our empirical results agree or disagree with theoretical results obtained for TwoMax.

2 Diversity Mechanisms and Previous Results for TwoMax

Following previous theoretical work, we consider diversity mechanisms embedded in a bare-bones EA: a (μ+1) EA with uniform parent selection in a population of size μ, using standard mutation (and no crossover). The offspring y replaces a worst individual z from the population if $f(y) \geq f(z)$ (see Algorithm 1).

Table 1 summarises previous work for the (μ+1) EA with diversity mechanisms on TwoMax (details for each (μ+1) EA variant can be found in the respective publications and in [19]). Some mechanisms succeed in finding both optima on TwoMax efficiently, that is, in (expected) time $O(\mu n \log n)$. Others have a very low success probability. Friedrich, Oliveto, Sudholt, and Witt [6] showed that the plain (μ+1) EA (PL, Algorithm 1) is not able to maintain individuals on both branches for a long time; the whole population is likely to converge to one of the two peaks [6, Theorem 1].

Introducing other simple mechanisms to the (μ+1) EA like *avoiding genotype duplicates (NGD)*, where (after initialisation) identical copies of individuals are prevented from entering the population [6, Algorithm 2], and *avoiding fitness duplicates (NFD)*, rejecting individuals with the same fitness [6, Algorithm 3], are not able to prevent the extinction of one branch, ending with the population converging to one optimum with high probability [6, Theorem 2 and 3, resp.].

In the (μ+1) EA with *probabilistic crowding (PC)* [2, Algorithm 1], an offspring competes with its parent and the survivor is chosen with a probability proportional to their fitness [12]. Covantes Osuna and Sudholt showed that this mechanism is unable to evolve solutions of significantly higher fitness than that obtained during initialisation (or, equivalently, through random search), even when given exponential time [2, Theorem 2.2]. The reason is that fitness-proportional selection between parent and offspring results in an almost uniform choice as both have very similar fitness, hence fitness-proportional selection degrades to uniform selection for replacement. In *deterministic crowding (DC)*, the offspring competes against its parent and replaces it if the offspring is at least as good [11]. For the (μ+1) EA with deterministic crowding [6, Algorithm 4],

Table 1. Overview of runtime analyses for the $(\mu{+}1)$ EA with diversity mechanisms on TwoMax, adapted from [2]. The success probability is the probability of finding both optima within (expected) time $O(\mu n \log n)$. Conditions include restrictions on the population size μ, the sharing/clearing radius σ, the niche capacity κ, window size w, and $\mu' := \min(\mu, \log n)$.

	Diversity mechanism	Success prob.	Conditions
PL	Plain $(\mu{+}1)$ EA [6]	$o(1)$	$\mu = o(n/\log n)$
NGD	No Genotype Duplicates [6]	$o(1)$	$\mu = o(\sqrt{n})$
NFD	No Fitness Duplicates [6]	$o(1)$	$\mu = \text{poly}(n)$
PC	Probabilistic Crowding [2]	$2^{-\Omega(n)}$	all μ
DC	Deterministic Crowding [6]	$1 - 2^{-\mu+1}$	all μ
RTS	Restricted Tournament Selection [2]	$\geq 1 - 2^{-\mu'+3}$	$w \geq 2.5\mu \ln n$
PFS	Population-based Fitness Sharing*$(\sigma = n/2)$ [6]	1	$\mu \geq 2$
FS	Individual-based Fitness Sharing*$(\sigma = n/2)$ [13]	1	$\mu \geq 3$
CL	Clearing $(\sigma = n/2)$ [1]	1	$\mu \geq \kappa n^2$

*Fitness sharing uses phenotypic sharing based on the number of ones

this mechanism with a sufficiently large population is able to reach both optima with high probability in expected time $O(\mu n \log n)$ [6, Theorem 4].

In *restricted tournament selection (RTS)*, for every offspring created, RTS selects uniformly at random (u. a. r.) w (*window size*) members from the population with replacement. Each offspring competes with the closest element from this set and the offspring replaces it if its fitness is at least as good [8]. For the $(\mu{+}1)$ EA with RTS [2, Algorithm 2], the mechanism succeeds in finding both optima of TwoMax in the same way as deterministic crowding, provided that w is chosen large enough [2, Theorem 3.1]. However, if w is too small, then it cannot prevent one branch taking over the other, leading to exponential running times with high probability [2, Theorem 3.4].

Fitness sharing derates the real fitness of an individual by an amount that represents the similarity to other population members. A *population-based fitness sharing (PFS)* approach [6, Algorithm 5], constructing the best possible new population amongst parents and offspring is able to find both optima in expected time $O(\mu n \log n)$ for any population size $\mu \geq 2$ [6, Theorem 5]. A drawback of this approach is that examining all possible new populations is computationally expensive. The conventional *fitness sharing (FS)*, where selection is based on individuals, was studied by Oliveto, Sudholt, and Zarges [13, Algorithm 1]. Population size $\mu = 2$ is not sufficient to find both optima in polynomial time; the success probability is only $1/2 - \Omega(1)$ [13, Theorem 1]. However, with $\mu \geq 3$, the $(\mu{+}1)$ EA finds both optima in expected time $O(\mu n \log n)$ [13, Theorem 3]. In all the above results, fitness sharing used a phenotypic distance: the distance between two search points x and y is the absolute difference in their number of ones. This choice is tailored to TwoMax and is not applicable in our scenario.

Hence our experiments must rely on fitness sharing with genotypic distances (Hamming distance), for which no runtime analyses are available.

In *clearing (CL)*, individuals are sorted in decreasing fitness and are processed in this order. Each individual is compared against other individuals according to its fitness with distance $< \sigma$ (*clearing radius*) which determines if both individuals belong to the same subpopulation (niche) or not. Then, the procedure iterates through all remaining individuals (i. e., those with lower or equal fitness) that haven't been cleared yet, until κ (*niche capacity*) best individuals (also called *winners*) have been found, and all remaining individuals from the same niche are cleared to the minimum fitness value possible [14]. Finally, the individuals with best fitness are selected (set of winners) and individuals coming from the new generation are preferred [1, Algorithm 1 and 2]. Clearing, with a clearing radius of $\sigma = n/2$, niche capacity $\kappa = 1$, and $\mu \geq \kappa n^2$ is able to find both optima in expected time $O(\mu n \log n)$ [1, Theorem 5.6].

3 Jansen-Zarges Multimodal Function Classes

Jansen and Zarges [9] introduced several problem classes spanned by k peaks $p_1, p_2, \ldots, p_k \in \{0,1\}^n$ for an arbitrary number $k \in \mathbb{N}$ of peaks. Each peak i has a position $p_i \in \{0,1\}^n$, a slope $a_i \in \mathbb{R}^+$, and an offset $b_i \in \mathbb{R}_0^+$. The fitness value of a search point depends on peaks in its vicinity as defined as follows.

Definition 1 (Definition 3 in [9]). *Let $k \in \mathbb{N}$ and k peaks $(p_1, a_1, b_1), (p_2, a_2, b_2), \ldots, (p_k, a_k, b_k)$ be given, then*

- $\mathrm{JZ}_1(x) := a_{\mathrm{cp}(x)} \cdot \mathrm{G}\big(x, p_{\mathrm{cp}(x)}\big) + b_{\mathrm{cp}(x)}$*, called nearest peak function,*
- $\mathrm{JZ}_2(x) := \max\limits_{i \in \{1,2,\ldots,k\}} a_i \cdot \mathrm{G}(x, p_i) + b_i$*, called weighted nearest peak function,*

where $\mathrm{cp}(x) := \arg\min_{i \in \{1,2,\ldots,k\}} \mathrm{H}(x, p_i)$ *is defined by the closest peak to a search point, and* $\mathrm{G}(x, p_i) := n - \mathrm{H}(x, p_i)$ *indicates the proximity of x to p_i.*

For the nearest peak function, $\mathrm{JZ}_1(x)$, the fitness of a search point x is determined by the proximity to the closest peak $i = \mathrm{cp}(x)$ along with its slope a_i and its offset b_i. In cases where multiple i minimise $\mathrm{H}(x, p_i)$, i should additionally maximise $a_i \cdot \mathrm{G}(x, p_i) + b_i$.

The weighted nearest peak function, $\mathrm{JZ}_2(x)$, takes the height of peaks into account. The peak i yielding the largest value $a_i \cdot \mathrm{G}(x, p_i) + b_i$ determines the function value. The bigger the height of a peak, the bigger its influence on the search space in comparison to smaller peaks. Note that, in case of equal slopes $a_1 = \cdots = a_k$ and equal heights $b_1 = \cdots = b_k$, both functions JZ_1 and JZ_2 using parameters $a_1, \ldots, a_k, b_1, \ldots, b_k$ are identical as for JZ_2 the maximum over all terms $a_i \cdot \mathrm{G}(x, p_i) + b_i$ for all $1 \leq i \leq k$ is attained for the closest peak $i = \mathrm{cp}(x)$.

Theorem 2. *For JZ_1 and JZ_2 using the same parameters $a_1 = \cdots = a_k$ and $b_1 = \cdots = b_k$ we have $\mathrm{JZ}_1 = \mathrm{JZ}_2$.*

In the case of two peaks p_1 and p_2, if these peaks are complementary, that is, $p_2 = \overline{p_1}$, then JZ_1 and JZ_2 generalise the TwoMax function, with TwoMax being the special case of $p_1 = 0^n, p_2 = 1^n, a_1 = a_2 = 1$ and $b_1 = b_2 = 0$ [9]. This setting was studied for the $(\mu{+}1)$ EA with clearing in [3].

We consider peaks being placed independently and u. a. r., as this strategy is simple, fair, and it scales towards an arbitrary number of peaks. The slopes are chosen equal to 1 for all peaks for the sake of simplicity. Even though the peaks are placed randomly, if the peaks have moderately similar heights, the resulting fitness landscape has a clear structure: with high probability all peaks are local optima, and all search points within a Hamming ball of radius $\Omega(n)$ belong to a peak's basin of attraction. This holds for both functions JZ_1 and JZ_2 as they have equal fitness values within the mentioned Hamming balls (but may have different values on other search points).

Theorem 3. *Assume k peaks $p_1, \ldots, p_k$ chosen independently and u. a. r. from $\{0,1\}^n$. If $a_1 = \cdots = a_k = 1$ and $\max_{1 \le i \le k} b_i - \min_{1 \le i \le k} b_i \le cn$ for a constant $c < 1/2$ then with probability $1 - k^2 e^{-\Omega(n)}$ for radius $r := (1/2 - c)/3 \cdot n$ we have:*

1. *all k peaks $p_1, \ldots, p_k$ are local optima in both f_1 and f_2,*
2. *for all $1 \le i \le k$, all search points in $\mathcal{B}_i := \{x \mid \mathrm{H}(x, p_i) \le r\}$, a Hamming ball of radius r around p_i, belong to the basin of attraction of p_i with respect to both JZ_1 and JZ_2, that is, there is a Hamming path from x to p_i on which the values of JZ_1 and JZ_2 are strictly increasing, and*
3. *for all search points $x \in \bigcup_{i=1}^{k} \mathcal{B}_i$, $\mathrm{JZ}_1(x) = \mathrm{JZ}_2(x)$.*

Proof. Assume without loss of generality $\min_{1 \le i \le k} b_i = 0$ (as adding a fixed value does not affect the problem structure), hence $b_i \le cn$ for all $1 \le i \le k$.

By Chernoff bounds, the probability that two different peaks will have Hamming distance at most $n/2 - r$ is $e^{-\Omega(n)}$. By the union bound, the probability that this holds for any pair of peaks is at most $k^2 \cdot e^{-\Omega(n)}$. We assume in the following that every two peaks have a Hamming distance larger than $n/2 - r$.

Now consider a search point $x \in \mathcal{B}_i$, that is, $\mathrm{H}(x, p_i) \le r$. Since $r \le n/6$ we have $n/2 \ge 3r$, and thus for all $j \ne i$ we have $\mathrm{H}(x, p_j) \ge \mathrm{H}(p_i, p_j) - \mathrm{H}(x, p_i) > n/2 - r - r \ge r \ge \mathrm{H}(x, p_i)$. So p_i is a unique closest peak, $\mathrm{cp}(x) = i$. By definition of JZ_1, the second statement follows for JZ_1 as on every shortest Hamming path from x to p_i, subsequently decreasing the Hamming distance to p_i increases the fitness by $a_i = 1$. Since $r \ge 1$ if n is large enough, p_i is a local optimum for JZ_1.

It only remains to show the third statement as then the first two statements also apply to JZ_2. To prove that $\mathrm{JZ}_2(x) = \mathrm{JZ}_1(x)$ for $x \in \mathcal{B}_i$, we need to show that the maximum over terms $a_j \cdot \mathrm{G}(x, p_j) + b_j = n - \mathrm{H}(x, p_j) + b_j$ from the definition of JZ_2 is attained for $j = i$. We have $n - \mathrm{H}(x, p_i) + b_i \ge n - r$ as $\mathrm{H}(x, p_i) \le r$ and $b_i \ge 0$. For $j \ne i$ we have $n - \mathrm{H}(x, p_j) + b_j < n/2 + 2r + cn$ as $b_j \le cn$ and $\mathrm{H}(x, p_j) \ge \mathrm{H}(p_i, p_j) - \mathrm{H}(x, p_i) > n/2 - r - r = n/2 - 2r$. Noting that $n/2 + 2r + cn = n/2 + 3r + cn - r = n/2 + (n/2 - cn) + cn - r = n - r$ establishes $n - \mathrm{H}(x, p_j) + b_j \le n - \mathrm{H}(x, p_i) + b_i$ and hence $\mathrm{JZ}_2(x) = \max_{1 \le j \le k}(n - \mathrm{H}(x, p_j) + b_j) = n - \mathrm{H}(x, p_i) + b_i = \mathrm{JZ}_1(x)$. □

4 Experimental Analysis

For the experimental analysis we test each of the algorithms from Table 1 (referred to by the acronyms defined in the first column) on Jansen-Zarges multimodal function classes. We consider a problem size $n = 100$, genotypic distance for all algorithms that require a dissimilarity measure and stop runs after $10\mu n \ln n$ generations. This time limit is motivated by [2, Lemma 3.3] stating that, loosely speaking, $2e\mu n \ln n \approx 5.44\mu n \ln n$ generations are sufficient to perform hill climbing on two peaks with high probability.

The experimental framework is divided in 3 experimental set-ups. In Sect. 4.1 we assess the ability of each mechanism to find many peaks with equal height, and in Sect. 4.2, we assess the ability of each mechanism to maintain the population diversity when considering peaks with different heights to yield global and local optima. For both sections, the number of peaks was increased exponentially as $k = \{2, 4, 8, \ldots, 64\}$. For each k, we generated 100 different instances choosing k peaks u. a. r. from $\{0, 1\}^n$. In each experiment, all algorithms are tested on the same set of 100 instances to ensure a fair comparison. The challenge for each mechanism is to find and maintain as many peaks as possible before reaching the $10\mu n \ln n$ generations; we record the fraction of the peaks found. The population size is chosen large enough ($\mu = 100$) to be able to accommodate all peaks.

The analysis in Sect. 4.3 is inspired by [3, Sect. 7.3] and focusses on landscapes with two peaks. In this section we take a closer look at the ability of the diversity mechanisms to deal with different basins of attraction, including a wider range of two-peaked landscapes than the ones likely to be generated by placing peaks u. a. r.. The goal is to observe which mechanisms are able to escape from local optima by tunnelling through the fitness valley that separates two peaks. We choose $\mu = 32$ as in [3, Sect. 7.3] and also consider the same two initialisations: the standard uniform random initialisation and biased initialisation where the whole population is initialised with copies of one peak (0^n for TwoMax). Biased initialisation is used in order to observe how the mechanisms are able to escape from a local optimum and how fast it is compared to a random initialisation.

Based on the theoretical analysis in [3] we define the window size $w = 2.5\mu \ln n$ for RTS. We know from [6,13] that both FS approaches with phenotypic sharing and $\sigma = n/2$ are always efficient on TwoMax but no theory for genotypic sharing is available. Preliminary experiments for genotypic sharing and $\sigma = n/2$ on TwoMax yielded poor results; however with $\sigma = n$ (which implies that all individuals always share fitness) both peaks were found in most runs. This makes sense on other landscapes as well as if σ is set smaller than the radius or basin of attraction around a local optimum, then FS is unable to push individuals away from said local optimum. Thus it seems best to err on the side of choosing σ too large rather than too small.

For CL the situation is different. If σ is chosen too large, such that there are several optima within a distance of σ, then global optima may be cleared, making it impossible to maintain many optima in the population. So for CL it seems best to err on the side of choosing σ too small rather than too large. We choose $\sigma = n/3$ for Sects. 4.1 and 4.2 as with high probability every two

different peaks will have a Hamming distance larger than $n/3$ (cf. Theorem 3). For Sect. 4.3, we use the recommendation $\sigma = \min\{\mathrm{H}(p_1, p_2), n/2\}$ from [3].

4.1 Finding Peaks of Equal Height

We consider the JZ_1 function with equal slopes $a_1 = \cdots = a_k = 1$ and offsets $b_1 = \cdots = b_k = 0$. We know from Theorem 2 that with equal parameters $\mathrm{JZ}_1 = \mathrm{JZ}_2$. In Fig. 1 (blue/left box plots), we show the fraction of peaks obtained in each of the 100 instances and its variance for each choice of k.

As can be seen, the PL, NGD and NFD perform poorly; these have already been proven to perform poorly on TwoMax [6]. PC as predicted in [2] is not able to find even one peak. FS performs best for an intermediate number of peaks, $k \in \{4, 8, 16\}$, but still far worse than the best mechanisms. This is in contrast to theoretical results [6,13] where FS in both variants was shown to be very effective on TwoMax. These differences may be down to the differences between TwoMax and JZ_1 with random peaks and/or they may be caused by the differences between phenotypic and genotypic sharing. Interestingly, PFS performs far worse than the conventional FS. This is surprising as PFS uses a significant amount of computation time to search for the best possible population (in terms of shared fitness) it can create out of all parents and offspring, hence we would have expected it to perform better than FS. A possible explanation for the poor performance of FS is that even when the population is able to locate basins of attraction of several peaks, we found several individuals scattered around each peak, apparently repelling each other and preventing each other from reaching the peak.

Finally, DC, RTS and CL perform surprisingly well: they find all optima most of the time for $k \leq 16$, and find most optima for $k = 32$. Only for a large number of $k = 64$ peaks, performance deteriorated to around 80% of peaks found. This deterioration is not surprising as the population size was fixed to $\mu = 100$. RTS with $w = 2.5\mu \ln n$ seems to behave similarly to DC as predicted in [2].

4.2 Finding Peaks with Different Height

For this case we make use of the JZ_2 function with $a_1 = \cdots = a_k = 1$ and $b_1 \cdots b_k$ chosen independently and u. a. r. from $[0, 1, \ldots, n/3]$. This range is motivated by Theorem 3, as here two peaks differ in their heights by at most $n/3$, choosing the leading constant $c := 1/3$ as the simplest constant smaller than $1/2$. Theorem 3 then yields that all search points within Hamming balls of radius $n/18$ centred at a peak are located in the peak's basin of attraction. Figure 1 (red/centre box plots) shows the fraction of peaks obtained in each of the 100 instances and its variance for each $k = \{2, 4, 8, \ldots, 64\}$ peaks. To gauge the quality of the peaks found, we also plot the normalised best fitness found (green/right box plots), formally f_i^*/opt_i where f_i^* is the fitness of the best peak found on instance i and opt_i is the optimal value of instance i.

In this setting PL, NGD and NFD manage to find the global optimum in up to 80% of instances. This suggests that on this function class it is fairly easy

to find a global optimum. However, they rarely find more than one peak, hence they seem to suffer from premature convergence. PC continues to show the worst performance of all mechanisms. PFS and FS find fewer peaks on JZ_2 compared to JZ_1. This makes sense since the former setting is more difficult than the latter; both mechanisms seem to suffer from the issues mentioned in Sect. 4.1.

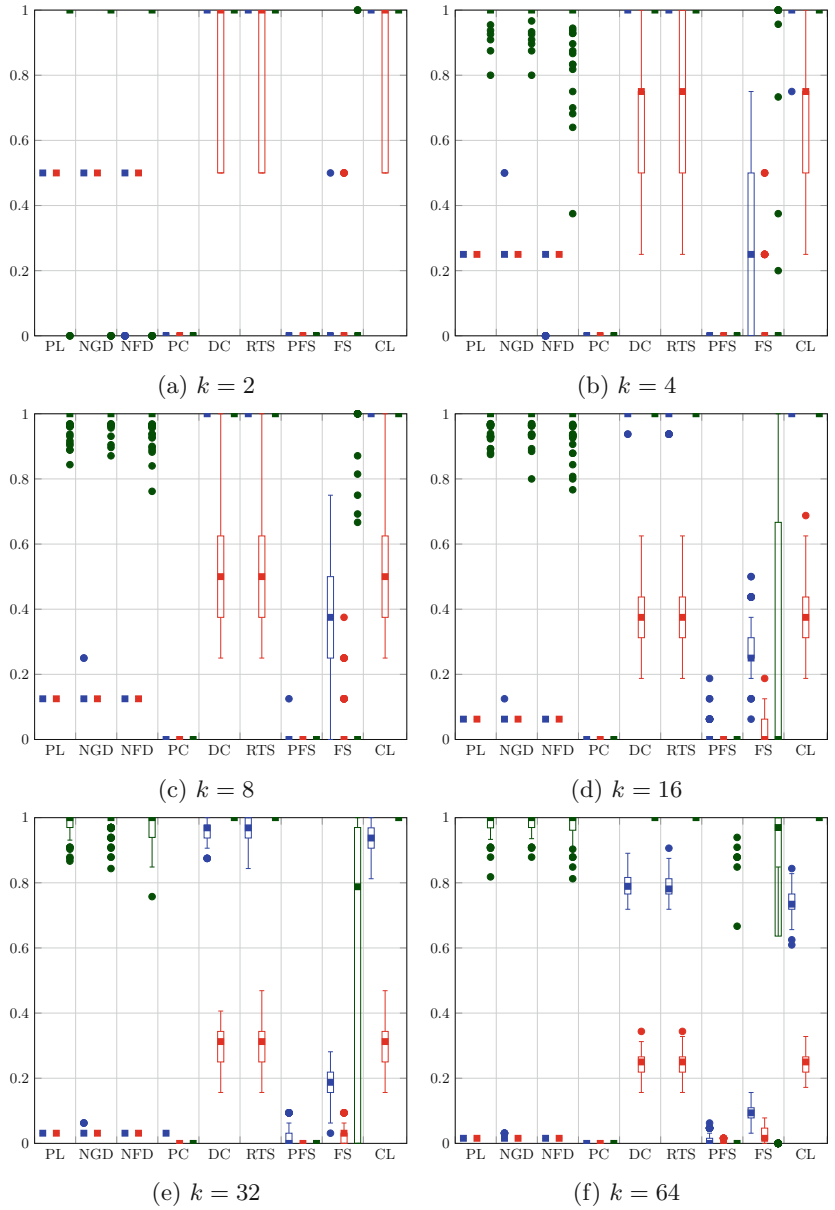
Finally, DC, RTS and CL also find fewer peaks due to the difficulty of this setting, but still show the best performance of all mechanisms analysed in this paper and they manage to find the global optimum in all instances. For $k \leq 8$ is not possible to always find all peaks any more, but they still manage to find at least 50% of the peaks. Then, for $k \geq 16$ the performance deteriorates in such a way that it is not possible to reach any more 50% of the peaks but still the mechanisms manage to find some of the peaks. The general cause of the drop in the performance seems to be that all mechanisms struggle to escape from the optimum found, also that low-quality optimums are being dropped when better peaks have been found.

4.3 Escaping from Local Optima

Theorem 3 and its proof suggest that when peaks are chosen u. a. r., they will have a Hamming distance close to $n/2$. We would like to investigate how the diversity mechanisms behave if peaks have different Hamming distances. Following [2], we focus on two peaks and vary their Hamming distance between 1 and n by choosing $p_1 = 0^n$ and $p_2 \in \{0^{n-1}1, 0^{n-2}1^2, \ldots, 1^n\}$, along with $a_1 = a_2 = 1$ and $b_1 = b_2 = 0$. As argued in [2], this captures the performance across all possible JZ_1 functions with two complementary peaks and the given slopes and offsets. In particular, it includes many functions that only have an exponentially small probability to be generated when choosing peaks independently and u. a. r. With biased initialisation, the algorithms have to find the other optimum by tunnelling through the fitness valley that separates these two peaks. This is a much harder task compared to hill climbing on various hills, where the aim is for the population to maintain a good spread over the search space.

We use the set-up and empirical data for CL from [3] and report the average number of generations of 100 runs, with two stopping criteria: both optima have been found or $t = 10\mu n \ln n$ generations were reached.

From Fig. 2a all mechanisms are effective when the Hamming distance is so small that the peaks are very close together such that the second peak can be found by a mutation of the first peak found (except for PC, that is not able to reach a single peak). But as the distance increases, the time for some mechanisms increases rapidly; they are inefficient on all non-trivial settings. DC and RTS seem to be agnostic of Hamming distances as they show a very stable and equal performance across the whole range of Hamming distances. This make sense as DC climbs up both peaks with equal probability (cf. the analysis on TwoMax [6]) and RTS behaves similarly to DC. CL is very effective and only mildly worse than DC and RTS. We see that for FS with genotypic sharing is only effective if the peaks have a Hamming distance that is very close to n or trivially small. For intermediate values, FS fails badly.

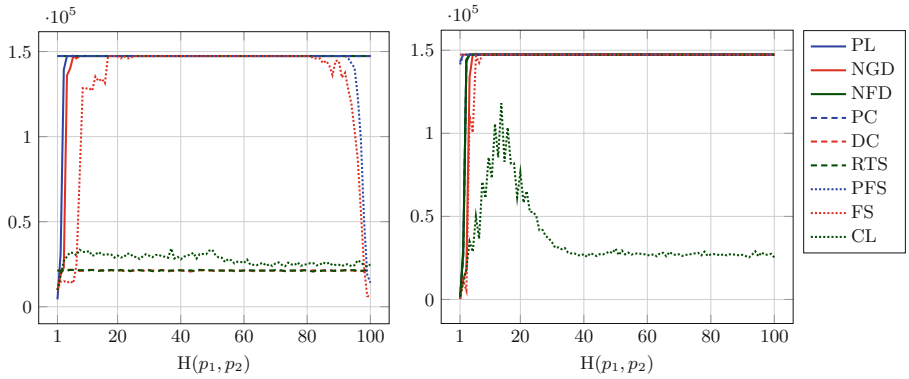


(a) $k = 2$ (b) $k = 4$

(c) $k = 8$ (d) $k = 16$

(e) $k = 32$ (f) $k = 64$

Fig. 1. Experimental results for all (μ+1) EA variants from Table 1 among 100 instances generated u. a. r. for each number of peaks $k = \{2, 4, 8, \ldots, 64\}$, $\mu = 100$ and $n = 100$, stopping runs after $10\mu n \ln n$ generations. Blue/left: fraction of peaks found on JZ_1 with peaks of equal height. Red/centre: fraction of peaks found on JZ_2 with peaks with different heights, $b_1 \ldots b_k$ chosen u. a. r. from $\{0, 1, \ldots, n/3\}$. Green/right: normalised best fitness found on JZ_2 experiments. Squares indicate median values. (Color figure online)



(a) Random Initialisation (b) Biased Initialisation

Fig. 2. The average number of generations among 100 runs for finding both peaks $p_1 = 0^n$ and $p_2 = \{0^{n-1}1, 0^{n-2}1^2, \ldots, 1^n\}$ on the fitness landscape defined by JZ_2 with $a_1 = a_2 = 1$ and $b_1 = b_2 = 0$ or $t = 10\mu n \ln n$ generations were reached, for all $(\mu+1)$ EA variants mentioned in Table 1, using $n = 100$ and $\mu = 32$. Results for both random and biased initialisation are shown.

With biased initialisation (Fig. 2b), CL is the only mechanism able to escape from local optima with different basins of attraction. As shown theoretically in [3], this is because cleared individuals are able to explore the fitness landscape by performing a random walk. We know from [6,13] that both FS approaches with phenotypic sharing and sharing radius $\sigma = n/2$ are able to escape from local optima as well, if the two peaks are complementary. With genotypic sharing both FS approaches perform very poorly and seem unable to escape from local optima. Also the other mechanisms fail as they are unable to accept worse search points.

5 Conclusions

We have performed an extensive empirical study involving 9 common diversity mechanisms on Jansen-Zarges multimodal function classes, covering various degrees of multimodality from 2 to 64 peaks and peaks having equal or different heights, reflected in their basins of attraction. Our results show that the plain $(\mu+1)$ EA, the simple mechanisms: avoiding genotype and fitness duplicates cannot maintain subpopulations on several peaks; once a peak has been found it seems impossible to escape from such a peak. Probabilistic crowding shows a terrible performance as it is unable to locate even a single peak. These findings are in line with theoretical results on TwoMax [2,6].

Previous theoretical results have shown that both fitness sharing approaches are always efficient on TwoMax if phenotypic distances are being used and parameters are set appropriately [6,13]. This includes the ability to climb down a peak and to tunnel through fitness valleys to reach other niches. Unfortunately this is not the case for fitness sharing with genotypic distance. Only when the

peaks have a Hamming distance that is trivially small or very close to n they seem to be effective; for any other intermediate case they show a poor performance.

Deterministic crowding, restricted tournament selection and clearing perform well for peaks with the same slope and height, much better than all other diversity mechanisms. Only for large numbers of peaks ($k = 64$) and different heights the performance starts to deteriorate. Finally, only clearing has shown the ability to escape from local optima since all other mechanisms seem unable to accept worse search points or unable to tunnel through fitness valleys.

References

1. Covantes Osuna, E., Sudholt, D.: Analysis of the clearing diversity-preserving mechanism. In: Proceedings of FOGA 2017, pp. 55–63. ACM (2017)
2. Covantes Osuna, E., Sudholt, D.: Runtime analysis of probabilistic crowding and restricted tournament selection for bimodal optimisation. In: Proceedings of GECCO 2018. (2018 to appear). http://arxiv.org/abs/1803.09766
3. Covantes Osuna, E., Sudholt, D.: On the runtime analysis of the clearing diversity-preserving mechanism. Evol. Comput. http://arxiv.org/abs/1803.09715
4. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput. Surv. **45**(3), 35:1–35:33 (2013)
5. Das, S., Maity, S., Qu, B.Y., Suganthan, P.: Real-parameter evolutionary multimodal optimization – a survey of the state-of-the-art. Swarm Evol. Comput. **1**(2), 71–88 (2011)
6. Friedrich, T., Oliveto, P.S., Sudholt, D., Witt, C.: Analysis of diversity-preserving mechanisms for global exploration. Evol. Comput. **17**(4), 455–476 (2009)
7. Glibovets, N.N., Gulayeva, N.M.: A review of niching genetic algorithms for multimodal function optimization. Cybern. Syst. Anal. **49**(6), 815–820 (2013)
8. Harik, G.R.: Finding multimodal solutions using restricted tournament selection. In: Proceedings of the 6th ICGA, pp. 24–31. Morgan Kaufmann Publishers Inc. (1995)
9. Jansen, T., Zarges, C.: Example landscapes to support analysis of multimodal optimisation. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 792–802. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_74
10. Lozano, M., García-Martínez, C.: Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: overview and progress report. Comput. Oper. Res. **37**(3), 481–497 (2010)
11. Mahfoud, S.W.: Niching methods for genetic algorithms. Ph.D. thesis, University of Illinois at Urbana-Champaign (1995)
12. Mengsheol, O., Goldberg, D.: Probabilistic crowding: deterministic crowding with probabilistic replacement. In: Proceedings of GECCO 1999, pp. 409–416 (1999)
13. Oliveto, P.S., Sudholt, D., Zarges, C.: On the runtime analysis of fitness sharing mechanisms. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 932–941. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_92
14. Pétrowski, A.: A clearing procedure as a niching method for genetic algorithms. In: Proceedings of ICEC 1996, pp. 798–803 (1996)
15. Sareni, B., Krahenbuhl, L.: Fitness sharing and niching methods revisited. IEEE Trans. Evol. Comput. **2**(3), 97–106 (1998)

16. Shir, O.M.: Niching in evolutionary algorithms. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 1035–1069. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9_32
17. Singh, G., Deb, K.: Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: Proceedings of GECCO 2006, pp. 1305–1312. ACM (2006)
18. Squillero, G., Tonda, A.: Divergence of character and premature convergence: a survey of methodologies for promoting diversity in evolutionary optimization. Inf. Sci. **329**, 782–799 (2016)
19. Sudholt, D.: The benefits of population diversity in evolutionary algorithms: a survey of rigorous runtime analyses (2018). http://arxiv.org/abs/1801.10087

Linear Combination of Distance Measures for Surrogate Models in Genetic Programming

Martin Zaefferer[1(✉)], Jörg Stork[1], Oliver Flasch[2], and Thomas Bartz-Beielstein[1]

[1] Institute of Data Science, Engineering, and Analytics, TH Köln, Steinmüllerallee 6, 51643 Gummersbach, Germany
{martin.zaefferer,joerg.stork,thomas.bartz-beielstein}@th-koeln.de
[2] sourcewerk GmbH, Roseggerstraße 59, 44137 Dortmund, Germany
oliver.flasch@sourcewerk.de

Abstract. Surrogate models are a well established approach to reduce the number of expensive function evaluations in continuous optimization. In the context of genetic programming, surrogate modeling still poses a challenge, due to the complex genotype-phenotype relationships. We investigate how different genotypic and phenotypic distance measures can be used to learn Kriging models as surrogates. We compare the measures and suggest to use their linear combination in a kernel.

We test the resulting model in an optimization framework, using symbolic regression problem instances as a benchmark. Our experiments show that the model provides valuable information. Firstly, the model enables an improved optimization performance compared to a model-free algorithm. Furthermore, the model provides information on the contribution of different distance measures. The data indicates that a phenotypic distance measure is important during the early stages of an optimization run when less data is available. In contrast, genotypic measures, such as the tree edit distance, contribute more during the later stages.

Keywords: Genetic programming · Surrogate models
Distance measures

1 Introduction

Genetic programming (GP) automatically evolves computer programs that aim to solve a task. This idea goes back to fundamental work by Koza [1] and follows the principles of evolutionary computation. The computer programs are individuals subject to an evolutionary process, which improves them based on their fitness, i.e., their ability to solve a problem. Examples for GP tasks are symbolic regression (SR), classification, and production scheduling [2,3].

Expensive fitness functions pose a challenge to evolutionary algorithms, including GP. This occurs, e.g., when the fitness function requires laboratory

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 220–231, 2018.
https://doi.org/10.1007/978-3-319-99259-4_18

experiments or extensive simulations. Frequently, Surrogate Model-Based Optimization (SMBO) is used to deal with expensive evaluations [4]. Most SMBO research focuses on problems with continuous variables, where many competitive regression models are available. In the context of GP, the use of surrogates is not well researched. This might seem surprising, as the computational bottleneck of most GP applications is the evaluation of fitness cases. Unfortunately, surrogate modeling of GP tasks, such as SR, is difficult, because it subsumes modeling of a complex genotype-phenotype-fitness mapping. Recent work in deep learning suggests that this mapping can be approximated, at least in certain domains of program synthesis [5].

In the last years, combinatorial search spaces were treated successfully with SMBO, by using distance-based models [6,7]. However, there is no generic way for choosing an adequate distance measure. For complex tree shaped structures, which occur in GP, it is challenging to select a suitable distance measure and find a feasible modeling approach. For that reason, we will focus on the following research questions regarding SMBO for GP and tree-shaped structures:

1. How do different distance measures compare to each other?
2. What impact do these distances have on the model?
3. How does SMBO based on a linear combination of these distances compare to a model-free Evolutionary Algorithm (EA) and random search?

To answer these questions, we will utilize bi-level optimization problems based on different SR tasks as test functions. While these test functions are not that expensive to evaluate (and hence are not a natural use-case for surrogate models), they present a challenging benchmark for the proposed models. They allow us to gain insights into the topics summarized by our research questions. We expect that our result can be transferred to other problems with tree shaped structures, such as program synthesis for general purpose or domain-specific languages.

2 Related Work

In the following, we will differentiate between two approaches, which will be further referred to as (a) SMBO and (b) SAEA.

(a) Sequential SMBO generates new candidate solutions by performing a search procedure on the surrogate model, e.g., as described for the Efficient Global Optimization (EGO) algorithm by Jones et al. [8].
(b) Approaches that utilize surrogates to assist an EA (SAEA), e.g., as described by Jin [9]. For example, the surrogate is utilized to support the selection process of an EA by predicting the fitness of proposed offspring.

Most studies on GP and surrogate modeling focus on SAEA. Kattan and Ong [10] describe an SAEA approach with two distinct Radial Basis Function Network (RBFN) models (semantic and fitness). The conjunction of both models is used to evolve a subset of the population. They report superiority of their approach over standard GP for three different tasks, including SR.

Hildebrandt and Branke [11] present a phenotypic distance. They optimize job dispatching rules with an SAEA approach. Their surrogate model is a nearest neighbor regression model based on the phenotypic distance. They demonstrate that their model allows for a faster evolution of good solutions. This approach is also discussed and extended by Nguyen et al. [12,13].

To the best of our knowledge, only Moraglio and Kattan [14] describe an SMBO approach to GP where a very limited number of function evaluations is allowed. They use an RBFN with appropriate distance measures. Their results did not indicate a significant improvement over the use of a model-free optimization approach.

In contrast to these works, we aim to learn Kriging models (following the idea of EGO [8]) and employ them in an SMBO framework with a severely limited number of 100 fitness function evaluations. Our models are based on a linear combination of three diverse distances. Like several of the above described studies, we use SR as a test case. We want to show that the relation between complex structures and their associated fitness can be learned and exploited for optimization purposes. Although SR is not particularly expensive, we argue that it presents a difficult and challenging test case to investigate whether our proposed models are able to learn such a complex search landscape.

3 A Test Case for SMBO-GP: Bi-level Symbolic Regression

In SR, a regression task is solved by evolving symbolic expressions. In essence, SR searches for a formula that best represents a given data set. The formulas can be represented by trees. Each tree consists of nodes and leaves, as well as the discrete labels on the nodes (mathematical operators, e.g., $+, -, *, /$) and leaves (variables and real-valued constants). Figure 1 shows the tree structure of the symbolic expression $\sqrt{c_1 - z_2} + (z_1 c_2)$. Our goal is to develop models that learn the relation between discrete tree structures and their fitness. For now, we are not interested in the influence of the real-valued constants. Hence, we suggest a bi-level problem definition.

3.1 Problem Definition

The upper level is the optimization of the discrete tree structure. For each fitness evaluation of the upper level, the lower level optimization problem has to be solved, which comprehends the optimization of the constants. Therefore, the upper level problem is defined by

$$\min_x F(x, c) \qquad \text{subject to} \quad c \in \arg\min_c f(x, c),$$

where x is the tree structure representation, $c \in \mathbb{R}^d$ is the set of d_c constant values, and $f(x, c)$ is the lower level objective function. Note, that the number of constants d_c depends on x. In extreme cases, the tree x may not contain any

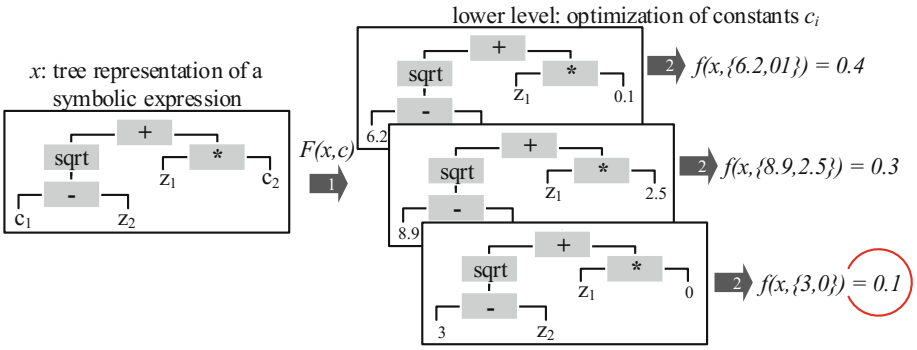


Fig. 1. Example for the upper level candidate $x = \sqrt{c_1 - z_2} + (z_1 c_2)$. To estimate its fitness $F(x, c)$, a lower level optimizer (step 1) estimates the fitness $f(x, c)$ for different constants (step 2) and returns the best to $F(x, c)$ (red circle). (Color figure online)

constants ($d_c = 0$), which eliminates the lower level problem. The fitness will be determined as

$$f(x, c) = 1 - |\text{cor}(\hat{y}(x, c), y)|, \tag{1}$$

where $\hat{y}(x, c)$ denotes the output of the symbolic expression for the data set, y is the corresponding vector of true observations, and $\text{cor}(\cdot, \cdot)$ is the Pearson correlation coefficient. If $\hat{y}(x, c)$ becomes infeasible (e.g., due to a negative square root or division by zero), we assign a penalty value. To that end, we use the upper bound of our fitness function, $f_{\text{penalty}}(x, c) = 1$. An example of an upper level candidate's evaluation is visualized in Fig. 1. If not stated otherwise, fitness evaluations refer to evaluations of the upper level function F.

3.2 Surrogate Model-Based Optimization

The SMBO approach we employ for the upper level optimization is loosely based on the EGO algorithm [8]. Initially, the search space is randomly sampled. The resulting data is used to learn a suitable regression model. This surrogate model is subject to a search via an optimization algorithm (e.g., an EA), which optimizes an infill criterion based on the model. An iteration ends with evaluating the actual (upper level) fitness of the new individual. Then, the surrogate model is updated with the new data and the procedure iterates.

As in standard EGO, we utilize a Kriging regression model, which assumes that the observed data is derived from a Gaussian process [15]. One reason for the popularity of Kriging in SMBO is that it allows to estimate its own uncertainty. The uncertainty estimate can be used to calculate the expected improvement (EI) infill criterion, which allows to balance exploitation and exploration in an optimization process [8,16].

Importantly, Kriging is based on correlation measures or kernels, which describe the similarity of samples. Exponential kernels, e.g., $\text{k}(x, x') = \exp(-\theta||x - x'||_2)$, with the parameter θ determined by Maximum Likelihood

Estimation (MLE), are often used. It is straightforward to extend kernel-based models to combinatorial search spaces [6,7]. The core idea is to replace the distance measure, e.g., in the exponential kernel $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$. The distance measure $\mathrm{d}(x, x')$ can be some adequate measure of distance between candidate solutions, such as an edit distance. Our study follows this idea. We will compare different distance measures and test how much they can contribute to Kriging models in an SMBO algorithm.

4 Kernels for Bi-level Symbolic Regression

We investigate four distance measures between trees or symbolic expressions, that will embedded into an exponential kernel.

4.1 Phenotypic Distance

The Phenotypic Distance (PhD) estimates the dissimilarity of two individuals (trees) based on their program output/phenotype, instead of using their code/genotype. This idea has been suggested by Hildebrandt and Branke for evolving dispatching rules via GP [11]. They defined a phenotypic dissimilarity by comparing the outcome of a decision rule based on a small set of test situations. Our SR tasks require a different definition of the phenotypic distance. We propose to measure the correlation between the outcomes of two symbolic expressions, with all numeric constants set to one. Hence, we save the effort of the optimization of the constants and compare the outputs of the expressions $\hat{y}(x, \mathbf{1})$ with

$$\mathrm{d}_{\mathrm{PhD}}(x, x') = 1 - |\mathrm{cor}(\hat{y}(x, \mathbf{1}), \hat{y}(x', \mathbf{1}))|.$$

If either of the two expressions is infeasible (e.g., due to division by zero), the distance will be set to one. Setting all constants to one is of course arbitrary. A random sample would also be possible but potentially problematic. A difference in phenotype could be perceived due to a different assignment of the constants on the leaves, rather than an actually different behavior of the symbolic expressions.

4.2 Tree Edit Distance

As an alternative to the PhD, we will also employ genotypic distances, i.e., distances between trees. One possible definition of distance between trees is the minimal number of edit operations required to transform one tree into another. This approach is denoted as the Tree Edit Distance (TED). We use the TED implementation that was introduced by Pawlik and Augsten [17]. It is available in the APTED library version 0.1.1 [18]. The APTED implementation counts the following edit operations: node deletion, node insertion, and node relabeling.

4.3 Structural Hamming Distance

The Structural Hamming Distance (SHD) [19] has been used to express genotypic dissimilarity for model-based GP in several studies [10,11,14]. Roughly speaking, it compares two trees by recursively checking each node that the two trees have in common. To compare nodes, it uses the Hamming Distance (HD), which is one if two labels are different and zero otherwise. The original SHD (SHD1) is defined as

$$\mathrm{d}_{\mathrm{SHD1}}(x, x') = \begin{cases} 1, & \text{if } \mathrm{arity}(x_0) \neq \mathrm{arity}(x'_0) \\ \mathrm{HD}(x_0, x'_0), & \text{if } \mathrm{arity}(x_0) = \mathrm{arity}(x'_0) = 0 \\ \Delta(x, x'), & \text{if } \mathrm{arity}(x_0) = \mathrm{arity}(x'_0) = m, \end{cases}$$

with

$$\Delta(x, x') = \frac{1}{m+1} + \mathrm{HD}(x_0, x'_0) + \sum_{i=1}^{m} \mathrm{d}_{\mathrm{SHD1}}(x_i, x'_i). \tag{2}$$

Here, x and x' are trees, x_0 indicates a root node of x, x_i with $i \geq 1$ is the i-th subtree of x, and $\mathrm{arity}(x_0)$ implies the number of subtrees linked to the corresponding node. We use a slight variation, which we refer to as SHD2. For the sake of simplicity, we define it for trees with a maximum arity of two. SHD1 and SHD2 are identical, except for the case $\mathrm{arity}(x_0) = \mathrm{arity}(x'_0) = m > 1$. Then, Eq. (2) becomes

$$\begin{aligned} \Delta(x, x') = & \frac{1}{m+1} + \mathrm{HD}(x_0, x'_0) + \\ & \min\left\{\mathrm{d}_{\mathrm{SHD2}}(x_1, x'_1) + \mathrm{d}_{\mathrm{SHD2}}(x_2, x'_2), \mathrm{d}_{\mathrm{SHD2}}(x_1, x'_2) + \mathrm{d}_{\mathrm{SHD2}}(x_2, x'_1)\right\}. \end{aligned}$$

That means, when two subtrees x_1, x_2 are compared with their counterparts x'_1, x'_2, we use the pairing or alignment between x and x' which yields the smaller distance. Potentially, this is more accurate, since it does not depend on the (arbitrary) initial alignment of the two trees. But SHD2 requires additional computational effort, even more so for larger arities.

The reason for using this modified variant lies in the nature of our SMBO algorithm. SAEAs yield datasets where some individuals will have common ancestors (or are ancestors of each other), and hence, are inherently more likely to be aligned with each other. Contrarily, SMBO generates new trees via a randomly initialized search that avoids direct an cestor relationships among individuals. This implies that two trees are more likely to have different alignments. Then, SHD2 is a potentially more accurate (but costly) measure.

4.4 Comparison and Linear Combination of Distances

For the comparison of the four different distance measures, we first calculated the distance matrices for 100 randomly generated trees (symbolic expressions). We used the same random tree-generation method as in Sect. 5. We computed the Pearson correlation between the different distance matrices. For this sample,

the SHD variants yielded a strong correlation of 0.99, which indicates that they reflect very similar information. For the remaining samples, the correlation was 0.51 (PhD, SHD2), 0.29 (PhD, TED), and 0.37 (TED, SHD2). That is, the largest diversity was observed between PhD and TED. Figure 2 visualizes the corresponding distance matrices. It shows that the SHD does have problems with differentiating between trees of different complexity. Several large blocks of the SHD matrices have a value of one, indicating that the respective trees are at maximum distance. This lack of perceiving a more fine-grained difference is problematic. It implies that any model based on SHD is potentially inaccurate for trees of a complexity that has not been observed so far. TED and PhD tend to see larger distances for more complex trees. This is obvious for TED, as complex trees require more operations to be transformed into each other. For PhD it is clear that complex trees can produce more diverse phenotypic behavior.

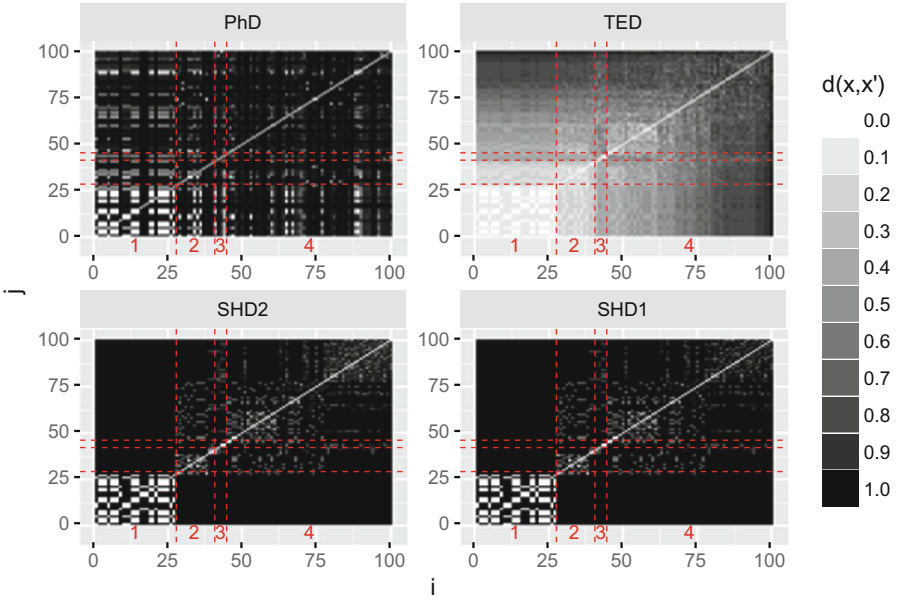


Fig. 2. Image plot of the four different tree distance measures. Each image cell is an element of a distance matrix. The trees are sorted by their complexity (tree depth and number of nodes). Trees in the lower left corner are less complex than those in the upper right. The tree depth is annotated in red at the bottom of each plot. (Color figure online)

With regards to the computational effort, we note that TED is by far the most expensive measure. It is followed by the PhD and the cheapest measure is SHD1. While the specifics strongly depend on the implementation, we note that the TED required at least an order of magnitude more computation time than the others. This is not surprising, as determining the minimal number of edit operations requires to solve an optimization problem.

The PhD measure seems most promising in terms of generalizability. Most GP problems involve some phenotypic behavior that may be measured/compared. SHD and TED are limited to problems with tree structures and discrete labels.

The diversity of the different distances suggests that it is promising to combine them. We propose a linear combination of the PhD, TED, and SHD2. We decided to focus on one of the SHD variants due to their similarity and chose the SHD2 variant due to its potentially increased accuracy. Also, its increased computational cost disappears compared to the larger costs of the TED. The linear combination in the kernel is

$$\mathrm{k}(x, x') = \exp\left\{-\beta_1 \mathrm{d}_{\mathrm{SHD2}}(x, x') - \beta_2 \mathrm{d}_{\mathrm{PhD}}(x, x') - \beta_3 \mathrm{d}_{\mathrm{TED}}(x, x')\right\}. \quad (3)$$

Each distance receives a weight $\beta_i \in \mathbb{R}^+$ that is determined by MLE. The linear combination allows for a potentially more accurate Kriging model. As we do not know a-priori which distance measure is appropriate for a certain problem (or whether they complement each other), the combination shifts this decision problem to the model. Furthermore, the weights provide insights into when and how much each distance contributes to the model.

5 Case Study

We performed a case study, testing the SMBO algorithm with six SR tasks.

Symbolic Regression Test Problems: We chose the Newton, sine-cosine, Kotanchek2D, and Salustowicz1D problems as used in [2] and the sqr and sqr+log problem as used in [10]. All problem configurations remained unchanged, i.e., operator set, data set size, and bounds for variables. We did not evaluate the derived symbolic expressions on an additional test set since our goal was to determine the ability of the SMBO algorithm to learn the connection between candidate solutions and fitness.

Lower level optimization of the constants: To optimize the lower level objective function, we decided to use the locally biased version of the Dividing RECTangles (DIRECT) algorithm [20] for a global search. DIRECT uses $1000 \times d_c$ evaluations of the objective function. The result of the DIRECT run is further refined with a Nelder-Mead local search [21] (also $1000 \times d_c$ evaluations).

Upper level optimization of the structure: All algorithms received a budget of 100 upper-level objective function evaluations to emulate an expensive optimization problem. We used Random Search (RS) and a model-free EA as baselines. All operators were taken from the `rgp` package [22]. For creating new individuals, both baselines used `randfuncRampedHalfAndHalf`, parameterized with a maximum tree depth of 4 and a probability to generate constants of 0.2. Furthermore, the EA employed `crossoverexprFast` for recombination, which randomly exchanges subtrees. For mutation, `mutateSubtreeFast` was used. The parameters of the mutation operator are as follows: 0.1 (probability to insert a subtree), 0.1 (probability to delete a subtree), 0.1 (probability of creating a subtree instead of a leaf), 0.2 (constant generation probability), and 4 (maximum

tree depth). Since constant values were not considered at the upper level, the respective bounds in the operator are both set to one. We employed a standard EA (based on `optimEA` in the `CEGO` package [23]) that used the above described operators. The EA used truncation selection, and a fixed number of children in each generation. The population size and number of children were tuned (see Sect. 5.1).

The upper level problem was also solved by the SMBO algorithm. We used the Kriging model from the `CEGO` package, with the kernel given in Eq. (3). The model was trained within $1,000$ likelihood evaluations (via DIRECT). The EA searched on the surrogate model with $10,000$ evaluations of the EI criterion in each iteration. The SMBO search was initialized with 20 random trees.

For the analysis, we recorded the best individual for each run. In addition, we recorded the weights used for linear combination of the distances in each iteration, to evaluate the contribution of each distance function over time. Each algorithm run was repeated 20 times.

5.1 Algorithm Tuning

We decided to tune some potentially sensitive parameters to allow for a more fair comparison between the model-based and model-free algorithm. The model-free GP algorithm's population size μ and number of children λ produced in each iteration were tuned. All combinations of $\mu = \{5, 10, 15, 20\}$ and $\lambda = \{1, 2, 3, 4, 5\}$ were tested. The optimization performance was expected to be sensitive to these parameters, due to the extremely small fitness evaluation budget.

For the SMBO algorithm, we did not tune μ and λ. Due to the overall larger complexity we decided to set the parameters based on experience only, without a detailed tuning. In fact, due to the larger number of evaluations (of the surrogate model) the algorithm should be less sensitive to μ and related parameters. Since $10,000$ evaluations of the surrogate model were allowed, a (relative to the model-free EA) large $\mu = 200$ was given to the EA and correspondingly larger $\lambda = 10$.

We also performed preliminary experiments with the mean square error (MSE) instead of the correlation-based fitness measurement in Eq. (1). The MSE-based experiments yielded rather poor results with SMBO. This may be explained by the penalty for infeasible candidates. The penalty value is very difficult to set for the MSE case. A poor choice may severely impair the ability to train a good Kriging model because of strong jumps or plateaus in the fitness landscape. While our preliminary experiments were not very detailed, they can be counted as additional tuning effort, since they influenced the choice of the correlation measure used in the phenotypic distance.

5.2 Analysis and Discussion

Boxplots of the best observed fitness after 50 and 100 evaluations of the objective function F are shown in Fig. 3. We report results of the tuned, model-free EA that achieved the best mean rank on all problems ($\mu = 15$, $\lambda = 1$). The minimal λ makes sense, as it allows to perform a large number of iterations despite the

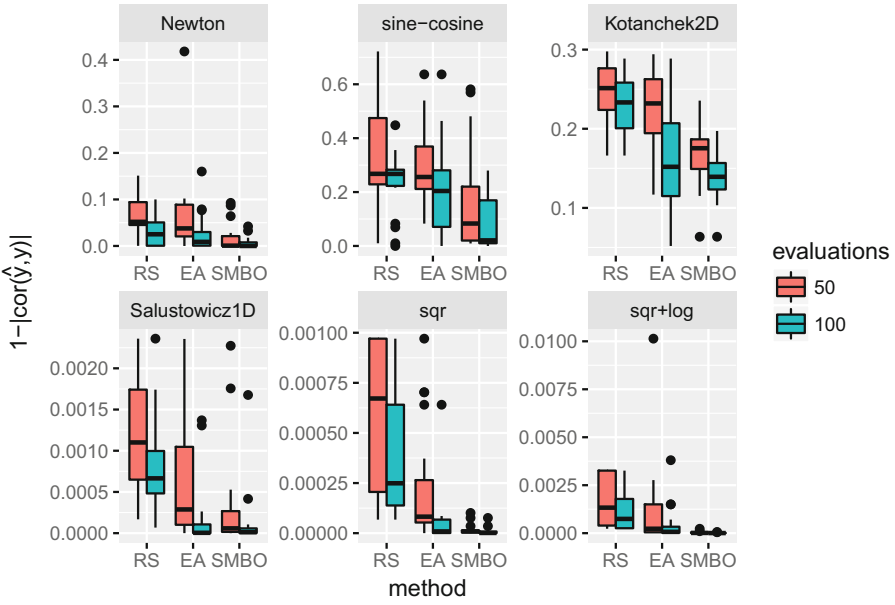


Fig. 3. Boxplot of best found values after 50 and 100 evaluations respectively.

small budget. For each problem and number of evaluations, we tested for statistical significance of the observed differences via the non-parametric Kruskal-Wallis rank sum test and Conover posthoc test, with a significance level of 0.05. The SMBO was significantly better than its two competitors in most cases, except for Salustowicz1D and Kotanchek2D after 100 evaluations, where no evidence for significant differences to the model-free EA is found. The EA was significantly better than the plain RS, except for Newton and sine-cosine (50 and 100 evaluations) as well as Kotanchek2D (50 evaluations).

To determine which distance measures contributed to these results, the weights of the linear combination are shown in Fig. 4. The weights are normalized so that they sum up to one. We show results for two problems, since they are similar in the other four cases. Usually, the PhD received the largest weights in the beginning, whereas the importance of the TED increased throughout the run, sometimes overtaking the PhD. SHD usually does not contribute as much, except for the sqr problem instance. Here, SHD overtakes both other distances at the end of the run. The generally larger importance of the PhD compared to SHD is in agreement with previous results by Hildebrandt and Branke [11], where a similar distance achieved better results than SHD.

We confirmed these results by additional optimization experiments for each single distance (i.e., without a linear combination). Runs with PhD tended to suggest good candidate solutions early, whereas TED and SHD performed better later on. The linear combination performed at least as well as the best of the single-distance models.

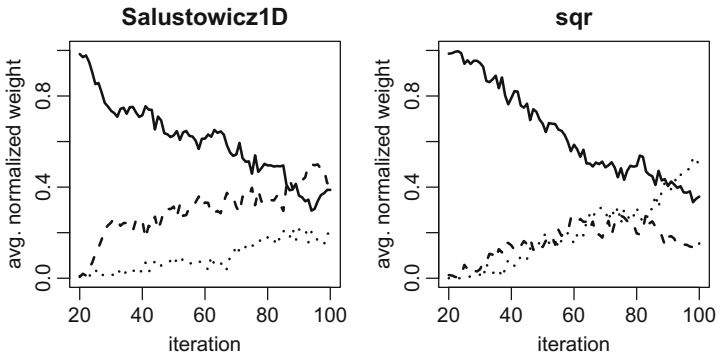


Fig. 4. Average normalized weights for the different kernels/distances. Solid line: PhD, dashed line: TED, dotted line: SHD2.

6 Conclusion and Outlook

We investigated whether three distance measures can be employed in an SMBO algorithm based on a Kriging model. We tested the algorithm with SR tasks. With respect to the research questions stated in Sect. 1, our results can be summarized as follows:

1. The distance measures PhD, SHD and TED are quite diverse. The SHD differentiates poorly between trees with different complexities. Especially the TED seems to be much more fine grained, but it requires the most computational effort. On the other hand, the PhD is comparatively cheap to evaluate and independent of the genotype.
2. Interestingly, the PhD seemed to contribute most, followed by the TED. This was especially true for small data sets at the beginning of an optimization run. Later on, TED and to a lesser extent SHD gained importance.
3. A Kriging model based on a linear combination of the three distances seems to be beneficial for SMBO. The SMBO algorithm outperformed a model-free algorithm and random search. All algorithms used no more than 100 fitness evaluations.

In future work, we would like to determine how well these results apply to other problem classes. Furthermore, alternatives to the linear combination of distances should be investigated.

References

1. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**(2), 87–112 (1994)
2. Flasch, O.: A modular genetic programming system. Ph.D. thesis, TU Dortmund (2015)
3. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. **3**(1), 41–66 (2017)

4. Bartz-Beielstein, T., Zaefferer, M.: Model-based methods for continuous and discrete global optimization. Appl. Soft Comput. **55**, 154–167 (2017)
5. Parisotto, E., Mohamed, A., Singh, R., Li, L., Zhou, D., Kohli, P.: Neuro-symbolic program synthesis (2016). arXiv e-prints 1611.01855
6. Moraglio, A., Kattan, A.: Geometric generalisation of surrogate model based optimisation to combinatorial spaces. In: Merz, P., Hao, J.-K. (eds.) EvoCOP 2011. LNCS, vol. 6622, pp. 142–154. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20364-0_13
7. Zaefferer, M., Stork, J., Friese, M., Fischbach, A., Naujoks, B., Bartz-Beielstein, T.: Efficient global optimization for combinatorial problems. In: Proceedings of the 2014 Genetic and Evolutionary Computation Conference, GECCO 2014, pp. 871–878. ACM, New York (2014)
8. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optim. **13**(4), 455–492 (1998)
9. Jin, Y.: Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evol. Comput. **1**(2), 61–70 (2011)
10. Kattan, A., Ong, Y.S.: Surrogate genetic programming: a semantic aware evolutionary search. Inf. Sci. **296**, 345–359 (2015)
11. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evol. Comput. **23**(3), 343–367 (2015)
12. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Selection schemes in surrogate-assisted genetic programming for job shop scheduling. In: Dick, G., et al. (eds.) SEAL 2014. LNCS, vol. 8886, pp. 656–667. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13563-2_55
13. Nguyen, S., Zhang, M., Tan, K.C.: Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Trans. Cybern. **47**(9), 1–15 (2016)
14. Moraglio, A., Kattan, A.: Geometric surrogate model based optimisation for genetic programming: Initial experiments. Technical report, University of Birmingham (2011)
15. Forrester, A., Sobester, A., Keane, A.: Engineering Design via Surrogate Modelling. Wiley, Hoboken (2008)
16. Mockus, J., Tiesis, V., Zilinskas, A.: The application of Bayesian methods for seeking the extremum. In: Towards Global Optimization 2, North-Holland, pp. 117–129 (1978)
17. Pawlik, M., Augsten, N.: Tree edit distance: robust and memory-efficient. Inf. Syst. **56**, 157–173 (2016)
18. Pawlik, M., Augsten, N.: APTED release 0.1.1. GitHub (2016). https://github.com/DatabaseGroup/apted. Accessed 01 June 2017
19. Moraglio, A., Poli, R.: Geometric landscape of homologous crossover for syntactic trees. In: 2005 IEEE Congress on Evolutionary Computation, Edinburgh, UK. IEEE (2005)
20. Gablonsky, J., Kelley, C.: A locally-biased form of the direct algorithm. J. Global Optim. **21**(1), 27–37 (2001)
21. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**(4), 308–313 (1965)
22. Flasch, O., Mersmann, O., Bartz-Beielstein, T., Stork, J., Zaefferer, M.: RGP: R genetic programming framework. R package version 0.4-1 (2014)
23. Zaefferer, M.: Combinatorial efficient global optimization in R - CEGO v2.2.0 (2017). https://cran.r-project.org/package=CEGO Accessed 10 Jan 2018

On Pareto Local Optimal Solutions Networks

Arnaud Liefooghe[1,2(✉)], Bilel Derbel[1,2], Sébastien Verel[3], Manuel López-Ibáñez[4], Hernán Aguirre[5], and Kiyoshi Tanaka[5]

[1] Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL, 59000 Lille, France
arnaud.liefooghe@univ-lille.fr
[2] Inria Lille – Nord Europe, 59650 Villeneuve d'Ascq, France
[3] Univ. Littoral Côte d'Opale, LISIC, 62100 Calais, France
[4] Alliance Manchester Business School, University of Manchester, Manchester, UK
[5] Faculty of Engineering, Shinshu University, Nagano, Japan

Abstract. Pareto local optimal solutions (PLOS) are believed to highly influence the dynamics and the performance of multi-objective optimization algorithms, especially those based on local search and Pareto dominance. A number of studies so far have investigated their impact on the difficulty of searching the landscape underlying a problem instance. However, the community still lacks knowledge on the structure of PLOS and the way it impacts the effectiveness of multi-objective algorithms. Inspired by the work on local optima networks in single-objective optimization, we introduce a PLOS network (PLOS-net) model as a step toward the fundamental understanding of multi-objective landscapes and search algorithms. Using a comprehensive set of ρmnk-landscapes, PLOS-nets are constructed by full enumeration, and selected network features are further extracted and analyzed with respect to instance characteristics. A correlation and regression analysis is then conducted to capture the importance of the PLOS-net features on the runtime and effectiveness of two prototypical Pareto-based heuristics. In particular, we are able to provide empirical evidence for the relevance of the PLOS-net model to explain algorithm performance. For instance, the degree of connectedness in the PLOS-net is shown to play an even more important role than the number of PLOS in the landscape.

1 Introduction

Context and motivation. In landscape analysis, the search space is regarded as an object having spatial and structural properties which are believed to characterize the intrinsic difficulties underlying the solving of an optimization problem and, hence, the dynamics and performance of recognized optimization techniques [16]. This aim may be achieved in an incremental manner: (i) gaining fundamental understanding of the information provided by the landscape structure, (ii) using this information to catalyze the good practice and design of search techniques, and (iii) developing out-of-the-box effective heuristics, for example, by contributing to the foundation of landscape-aware selection methods based on statistical

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 232–244, 2018.
https://doi.org/10.1007/978-3-319-99259-4_19

and machine learning prediction models [11]. In this paper, we make a step towards leveraging the so-called *local optima network* model, with the primary goal of providing empirical evidence on its relevance and usefulness for multi-objective combinatorial optimization.

Overview of (single-objective) local optima network. Given a combinatorial optimization problem, it is widely accepted that understanding the underlying (landscape) properties of *local optimal* solutions is of high importance. This is especially because heuristics are essentially navigating through the landscape in search of high-quality local optima that are hopefully as close as possible to the global one. A major issue, however, comes from the curse of dimensionality, which makes it difficult to define metrics and extract statistics that are meaningful and easy to interpret. In single-objective optimization, the *local optima network* (LON) is a relatively new model [13,14] that adapts the notion of the inherent network of energy surfaces in chemical physics [6] to compress the information given by the whole search space into a smaller and synthetic mathematical object described as a graph. The local optima are thereby the nodes of the graph and there is an edge or a transition (arc) between two nodes if the search process is potentially able to jump from one local optima (or basin of attraction) to another one. Variants of the definition of edges and nodes exist that capture different facets of search algorithms, however, these variants do not alter the primary purpose behind the analysis of LONs. The LON model not only aims at capturing in detail the number and distribution of local optima in the search space, but it also aims at enabling the definition of novel features having interesting correlation with the behavior of local search heuristics [4,14].

Overview of related work in multi-objective optimization. To the best of our knowledge, there are no investigations on defining and studying LONs in multi-objective combinatorial optimization, although local optimality also constitutes a central issue in the multi-objective case. For instance, the distribution and the connection of local optima induced by a scalarizing (single-objective) function is studied in [1,9], which is informative when multiple scalarizations of the objectives are considered. In [15], local optimality under Pareto dominance is defined, which is then related to the convergence of Pareto local search. Such a definition of Pareto local optimal solutions (PLOS) was later used in a number of studies. For example, it is shown in [18] how the characteristics of a problem instance can be related to the number of PLOS. Alternative definitions of local optima in terms of sets of nondominated solutions are also being investigated in relation to different multi-objective search paradigms [12]. A definition of local optimal sets for continuous multi-objective landscapes can also be found in [8]. In fact, although a multi-objective search heuristic aims at providing a whole set of solutions approximating the Pareto set, information about PLOS is found to influence the global search performance, especially when considering algorithms using local search and Pareto dominance as core components [3].

Paper contributions. The work described in this paper aims at pushing further the understanding of the structure of PLOS, eventually leading to new insights into what makes a multi-objective problem instance difficult and what makes a particular multi-objective search algorithm effective. More specifically, our contribution can be summarized following two lines:

- We introduce a model, inspired by the single-objective LON, that describes the network of Pareto local optimal solutions (PLOS-net) for multi-objective optimization. Using a comprehensive set of ρmnk-landscapes, we conduct a preliminary visual inspection and comparison of the corresponding PLOS-nets, showing how such networks enlighten the nature of the multi-objective landscape where the search is expected to operate.
- We further analyze the proposed PLOS-nets by introducing features from network analysis. Based on a comprehensive statistical analysis, the predictive importance of the newly-defined PLOS-net features is then studied with respect to the performance of pure Pareto local search [15] and of a global evolutionary multi-objective optimizer [10]. Our analysis indicates that PLOS, and more critically their connections in the PLOS-net, have the largest influence on the runtime of PLS, even larger than the number of objectives and their conflicting nature.

Outline. The rest of this paper is organized as follows. In Sect. 2, we recall some definitions for multi-objective optimization, and we describe the ρmnk-landscapes that will be later used in our empirical investigations. In Sect. 3, we define the notion of PLOS-net for multi-objective optimization, and we conduct a visual inspection of selected PLOS-nets. We also define features extracted from the PLOS-net, and we discuss how they relate with search performance. In Sect. 4, we study the effect and importance of PLOS-net features on algorithm performance. In Sect. 5, we conclude the paper and discuss open issues.

2 Multi-objective Optimization and ρmnk-Landscapes

We assume that we are given a (black-box) optimization problem characterized by a set of feasible solutions X (the *decision space*) and an *objective function* $f\colon X \mapsto \mathbb{R}^m$, to be maximized. We denote by $\mathcal{Z} = f(X) = \{z \in \mathbb{R}^m \,|\exists x \in X : z = f(x)\}$ the image of X in the *objective space*. Given two solutions $x, x' \in X$, x is said to *dominate* x' ($x \prec x'$) iff $f_i(x') \leqslant f_i(x)$ for all $i \in \{1, \ldots, m\}$ and $f_i(x') < f_i(x)$ for at least one $i \in \{1, \ldots, m\}$. The set $X^\star \subseteq X$ for which there exists no solution $x \in X$ such that $f(x) \prec f(x^\star)$ for all $x^\star \in X^\star$, is the *Pareto set* and its image $Z^\star = f(X^\star)$ in the objective space is the *Pareto front*.

We consider ρmnk-landscapes as a problem-independent model used for multi-objective multi-modal landscapes with objective correlation [18]. Solutions are binary strings of size n, i.e., $X = \{0,1\}^n$. The objective vector $f = (f_1, f_2, \ldots, f_m)$ is defined as $f\colon \{0,1\}^n \mapsto [0,1]^m$ such that each objective function f_i is a pseudo-boolean function to be maximized. The value $f_i(x)$ of a solution $x = (x_1, x_2, \ldots, x_n)$ is the average value of the contributions associated

with each variable x_j. Given objective f_i and variable x_j, a component function $f_{ij}\colon \{0,1\}^{k+1} \mapsto [0,1]$ assigns a real-valued contribution to every combination of x_j and its k *epistatic interactions* $\{x_{j_1}, \ldots, x_{j_k}\}$. For every $i \in \{1, \ldots, m\}$, the objective function is then defined as $f_i(x) = \frac{1}{n} \sum_{j=1}^{n} f_{ij}(x_j, x_{j_1}, \ldots, x_{j_k})$. The epistatic interactions, i.e., the k variables that influence the contribution of x_j, are set uniformly at random among the $(n-1)$ variables other than x_j [7]. By increasing the number of epistatic interactions k from 0 to $(n-1)$, problem instances can be gradually tuned from smooth to rugged. In ρmnk-landscapes, f_{ij}-values follow a multivariate uniform distribution of dimension m, defined by an $m \times m$ positive-definite symmetric covariance matrix (c_{pq}) such that $c_{pp} = 1$ and $c_{pq} = \rho$ for all $p, q \in \{1, \ldots, m\}$ with $p \neq q$, where $\rho > \frac{-1}{m-1}$ defines the correlation among the objectives [18]. The positive (respectively, negative) objectives correlation ρ decreases (respectively, increases) the degree of conflict between the different objective function values. We use the same correlation coefficient, and the same epistatic degree and interactions for all objectives.

We generate 520 ρmnk-landscapes as follows. The problem size is set to $n = 16$; the problem non-linearity to $k \in \{0, 1, 2, 4\}$, from linear to rugged landscapes; the number of objectives to $m \in \{2, 3\}$; and the objective correlation to $\rho \in \{-0.7, -0.4, -0.2, 0, 0.2, 0.4, 0.7\}$ subject to $\rho > \frac{-1}{m-1}$, from conflicting to correlated objectives. We generate 10 instances independently at random for each parameter combination.

3 Pareto Local Optimal Solutions Network

3.1 Definition and Visual Inspection of PLOS-net

The Pareto local optimal solutions network (PLOS-net) proposed in this paper can be constructed for a given optimization problem (X, f) and *neighborhood relation* $\mathcal{N}\colon X \mapsto 2^X$. For ρmnk-landscapes, the neighborhood relation is the *1-bit-flip* operator: two solutions are neighbors if the Hamming distance between them is one. Let us first define *Pareto local optimal solution* (PLOS) [15].

Definition 1. *A solution $x \in X$ is a* Pareto local optimal solution *if it is not dominated by any of its neighbors: $\forall x' \in \mathcal{N}(x)$, $\neg(x' \prec x)$.*

For $m = 1$, this is equivalent to the conventional definition of a single-objective local optimal solution. Based on PLOS, we then define a PLOS-net as follows.

Definition 2. *A Pareto local optimal solutions network (PLOS-net) is a (undirected unweighted simple) graph $G = (N, E)$, such that the set of vertices N are the Pareto local optimal solutions, and there is an edge $e_{ij} \in E$ between two nodes x^i and x^j iff $x^i \in \mathcal{N}(x^j)$ or $x^j \in \mathcal{N}(x^i)$.*

Two solutions connected by an edge in the PLOS-net are necessarily mutually nondominated. Moreover, the Pareto (global) optimal solutions (POS) are particular nodes of the PLOS-net.

A visual inspection of the so-defined PLOS-nets is given in Fig. 1 for some selected landscapes. The different PLOS-nets are extracted by full enumeration.

The two first rows are for a (fixed) number of objectives ($m = 2$), while the last two are for $m = 3$. In the first and third rows, the degree of objective correlation ρ varies for a fixed value of $k = 1$, whereas in the second and fourth rows, the value of k varies for a fixed value of $\rho = 0.0$. The x– and y–axes correspond to the first two objectives (f_1, f_2) and the scale is intentionally different for the different instances for a better visualization. For $m = 3$, we see a 2D-projection while the color intensity depicts f_3–values.

By mapping the PLOS-nets into the objective space, we intend to illustrate the shape of the network while providing a first hint on the impact of instance parameters on the distribution of PLOS. As somewhat expected, the objective correlation ρ impacts the shape and region spanned by the Pareto front; e.g., nodes of the PLOS-net at the upper objective space limit for $m = 2$. More interestingly, the number of nodes in the PLOS-nets increases when the number of objectives m, when the degree of conflict between the objectives $-\rho$, or when the problem non-linearity k increase. A similar trend for the number of connections between nodes can be observed. However, the increase in the PLOS-net density seems to be proportionally lower than the number of PLOS.

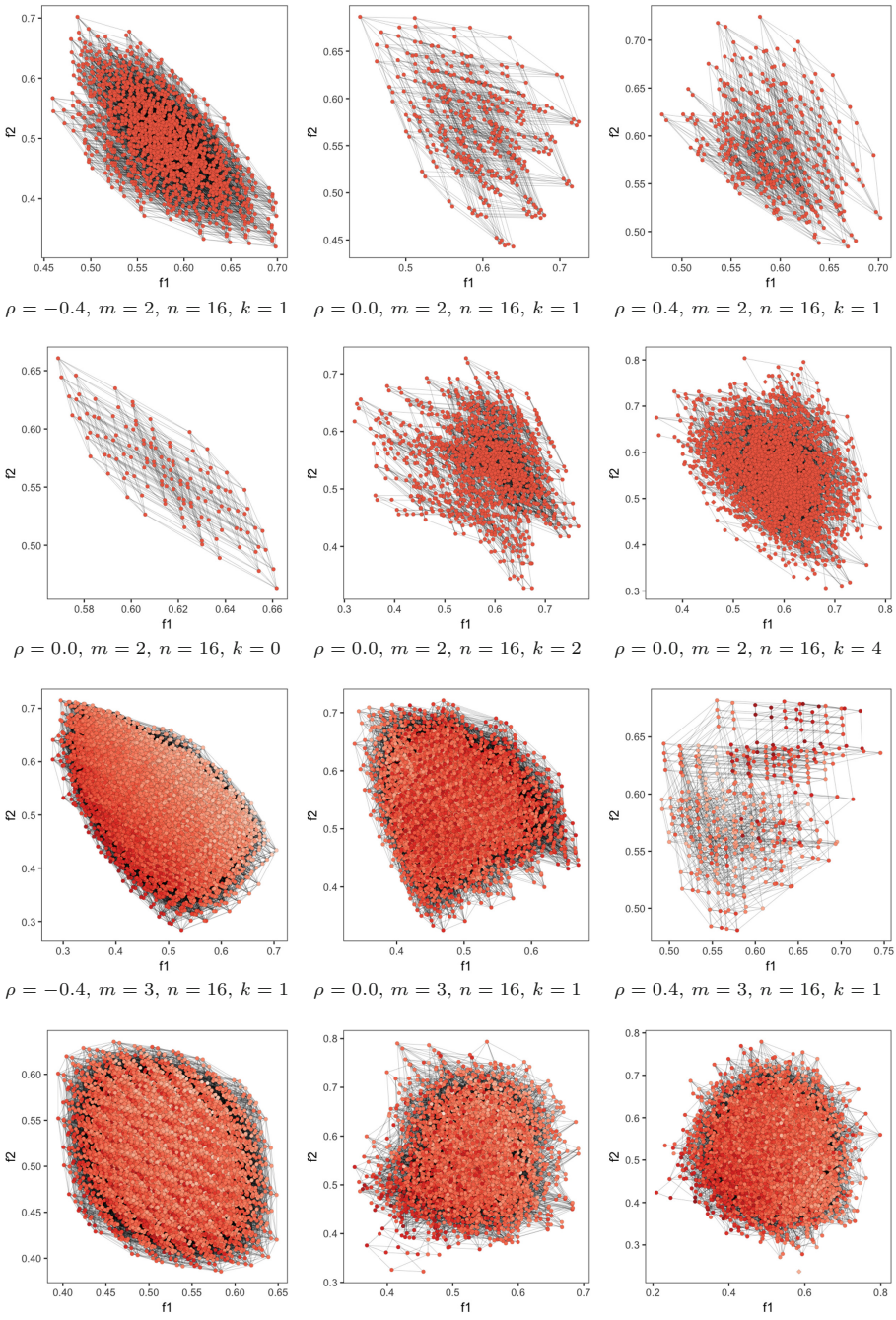
This visual projection of PLOS-nets in the objective space is certainly not sufficient to elicit the structure and complexity of the underlying graphs in a comprehensive manner. For instance, at first sight, PLOS-nets might look fully connected. As it will be highlighted later in our analysis, this is definitely not true in general for the considered instances. Interestingly, the connectedness between PLOS is one critically important aspect for multi-objective local search. A closer investigation of the PLOS-net model will enable us to fully capture such an aspect, among others, in a very natural manner. Because of its roots in graph theory and complex networks [17], the PLOS-net model allows us to define informative metrics and statistics with respect to the structure of PLOS, as detailed in the following.

3.2 Definition of PLOS-net Features

Looking at the PLOS-net as a mathematical object, we propose to define and analyze a number of graph-based features inspired by previous studies from single-objective LON analysis [4,13,14] but taking into account the Pareto dominance relation when necessary, and hence accommodating the multi-objective nature of the considered problems.

The first two considered features give a general idea on the node degrees, that is the number of edges leaving a PLOS and connecting it to other PLOS.

- `node_prop`: The number of nodes in the network, i.e., the number of PLOS, proportional to the search space size.
- `degree_avg`: The average degree of a node, proportional to the number of nodes. This metric is equivalent to the density of edges, that is, the number of edges proportional to the maximum number of edges in a complete graph.



$\rho = -0.4,\ m = 2,\ n = 16,\ k = 1$ $\rho = 0.0,\ m = 2,\ n = 16,\ k = 1$ $\rho = 0.4,\ m = 2,\ n = 16,\ k = 1$

$\rho = 0.0,\ m = 2,\ n = 16,\ k = 0$ $\rho = 0.0,\ m = 2,\ n = 16,\ k = 2$ $\rho = 0.0,\ m = 2,\ n = 16,\ k = 4$

$\rho = -0.4,\ m = 3,\ n = 16,\ k = 1$ $\rho = 0.0,\ m = 3,\ n = 16,\ k = 1$ $\rho = 0.4,\ m = 3,\ n = 16,\ k = 1$

$\rho = 0.0,\ m = 3,\ n = 16,\ k = 0$ $\rho = 0.0,\ m = 3,\ n = 16,\ k = 2$ $\rho = 0.0,\ m = 3,\ n = 16,\ k = 4$

Fig. 1. Exemplary PLOS-nets. For $m = 3$, a two-dimensional projection is displayed, the darker the node color, the higher the f_3-value. Notice the different scales of axes.

The next two features are related to the connectedness between PLOS. We argue that connectedness affects whether a local search with exhaustive neighborhood exploration will be able (or not) to find all PLOS.

- `comp_prop`: The number of connected components in the graph, proportional to the number of nodes. A connected component is a maximal subgraph in which there exist a path between any pairs of nodes.
- `comp_max_size`: The size of the largest connected components, proportional to the number of nodes.

The next three features deal with the paths that connect PLOS, either among them or to PLOS that are also Pareto (global) optimal solutions (POS). This is mostly intended to provide an information about how fast navigating between PLOS can lead to the Pareto front.

- `path_length`: The average path length between any pair of nodes, proportional to the maximum distance between pairs of solutions. When the graph is not connected, only existing paths are considered.
- `path_pos_exist`: The average number of nodes that are connected with (at least) one POS in the graph, proportional to the number of nodes.
- `path_length_pos`: The average path length between nodes and their closest POS in the graph, proportional to the maximum distance between pairs of solutions. Nodes not connected to any POS are *not* taken into account.

The last three features describe the similarity of PLOS and are intended to capture how the PLOS may be clustered together, eventually inferring some preferred connections as observed in network communities.

- `assort_degree`: The tendency of nodes to be connected to nodes with a similar degree.
- `assort_pos`: The tendency of local (resp. global) optimal nodes to be connected to local (resp. global) optimal nodes.
- `assort_rank`: The tendency of nodes to be connected to nodes with a similar rank. The rank of nodes is here defined in terms of nondominated sorting within the whole search space (dominance depth) [5].

3.3 Exploratory Analysis

Based on the 10 aforementioned features, we are able to provide a more detailed analysis on the effect of instance parameters on the structure of the PLOS-net, as reported in Fig. 2. Confirming our visual inspection, one can clearly notice the positive correlation of the number of PLOS with $-\rho$, m and k. This correlation is actually reverted when looking at the density of the graph `degree_avg`. This indicates that not only the number of PLOS increases with the intrinsic difficulty of an instance, but PLOS become more sparsely connected. A similar situation occurs when looking at the size of the induced connected components. For instances with a low degree of non-linearity ($k = 1$), the proportional size

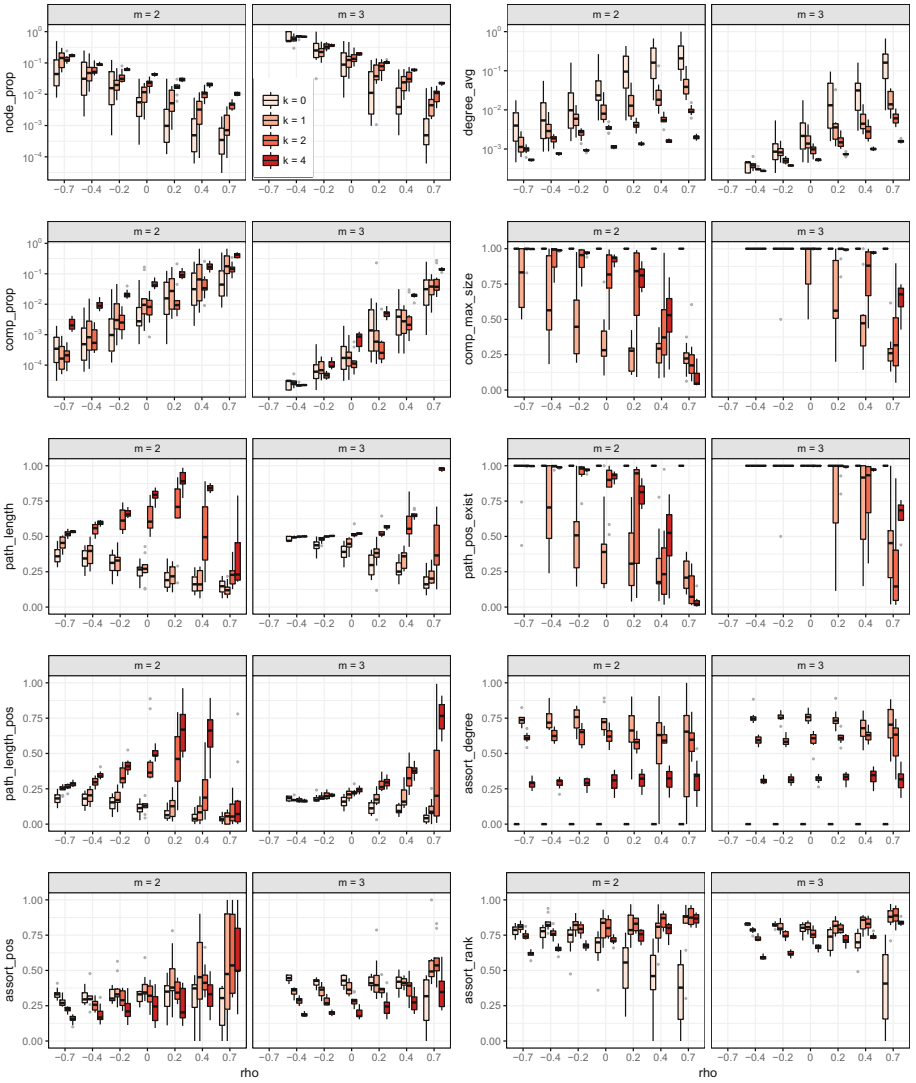


Fig. 2. Distribution of feature-values with respect to instance parameters.

of the largest connected component `comp_max_size` is 1 in most cases, meaning that all PLOS are connected. This is confirmed by the `path_pos_exist` feature, measuring how many PLOS are connected to (global) POS, which is also found to be close to 1 for $k = 1$. However, with the exception of 3-objective instances with highly conflicting objectives ($\rho < 0$), the PLOS-nets are clearly disconnected (`path_pos_exist`) and PLOS are relatively far from the Pareto set (`path_length_pos`). Surprisingly, the degree of connectedness seems to increase substantially for the highest value of $k = 4$. Finally, we can remark

that the tendency of a node to be connected with nodes having the same degree (`assert_degree`) is mainly impacted by the degree of non-linearity k, while the other parameters ρ and m have only a marginal effect. This trend is roughly the same when looking at how the PLOS are mutually connected as a function of their nondominated rank (`assort_rank`). To summarize, we found that the connectedness of the PLOS-net is highly related to instance parameters, especially to the degree of non-linearity k, which is obviously related to search difficulty. In the next section, we go deeper in the analysis by providing a comprehensive statistical study of the predictive power of the features with respect to two conventional Pareto-based multi-objective search algorithms.

4 PLOS-net Features vs. Search Performance

4.1 Algorithms and Search Performance

In the following, we consider the relative impact of PLOS-net features on both Pareto Local Search (PLS) [15] and the Global Simple Evolutionary Multi-objective Optimizer (G-SEMO) [10]. For PLS, we are interested in the total number of evaluations performed by the algorithm before falling into a *maximal* Pareto local optimum *set* [15]. For G-SEMO, the stopping condition is arbitrarily set to 10^4 solutions evaluated. For both algorithms, we are interested in the quality of the approximation set, measured in terms of the *Pareto front resolution*, i.e., the proportion of nondominated solutions identified. We perform 30 independent runs of each algorithm per instance.

The expected number of evaluations performed by PLS is reported in Fig. 3. The expected Pareto front resolution obtained by PLS and G-SEMO is reported in Fig. 4. These two figures illustrate how strong is the effect of the intrinsic instance parameters on search performance. Notice in particular the linear slope of the PLS runtime as a function of ρ, and the strong similar effect of k on the accuracy of PLS and G-SEMO, independently of ρ and m.

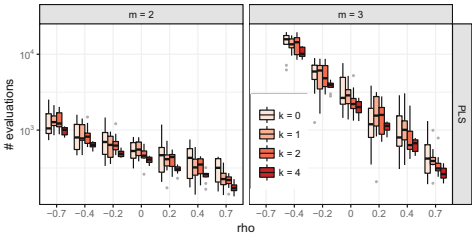


Fig. 3. Expected number of evaluations performed by PLS.

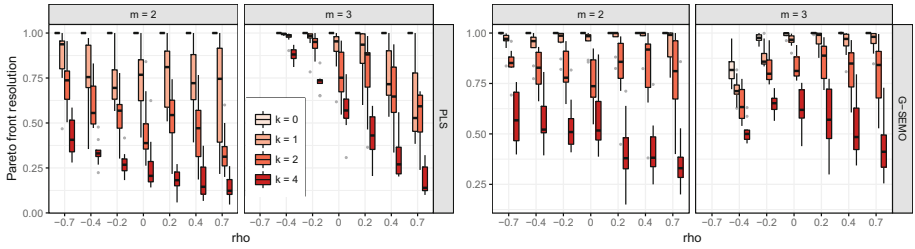


Fig. 4. Expected Pareto front resolution obtained by PLS and G-SEMO.

4.2 Effect of PLOS-net Features on Search Performance

The correlation of instance parameters (ρ, m, k) and PLOS-net features with search performance is reported in Fig. 5. First, both the number of objectives m and their correlation degree ρ are highly correlated with the runtime of PLS and very little with search quality, while the opposite holds for the problem non-linearity k. Second, interesting correlations can clearly be observed for PLOS-net features. On the one hand, the number of connected components as well as the density of the network are correlated both to runtime and quality. We argue that this is due to the fact that local search is basically exploring the connected components in a pseudo-exhaustive manner. Hence, the performance of the considered algorithms is naturally correlated to how local optimal solutions are connected together. On the other hand, the features related to path length and node connection similarity have the lowest correlation with the runtime of PLS. However, this is no more true when examining the search resolution where the correlation of such features to the Pareto front resolution obtained by PLS and G-SEMO is consistently similar and more pronounced. In fact, the distance between local and global optima, as captured by `path_lenght_pos`, is a natural outcome to understand how likely a Pareto local search can find its way to Pareto optimal solutions when starting from a local optima and navigating throughout the PLOS-net under Pareto dominance.

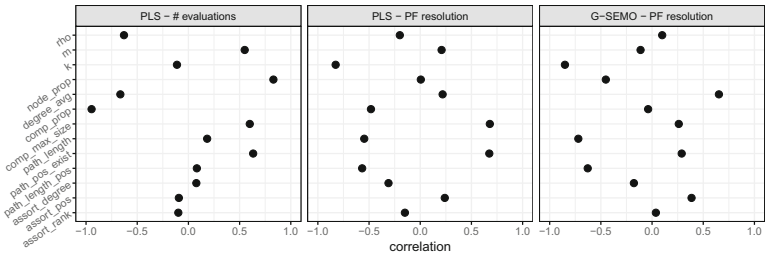


Fig. 5. Spearman's rank correlation coefficient between parameters/feature-values and the expected number of evaluations performed by PLS (left), the expected Pareto front resolution obtained by PLS (middle) and by G-SEMO (right).

Table 1. Variance explained by the regression model for different input variables.

	Instance parameters	PLOS-net parameters	Instance parameters + PLOS-net parameters
PLS – # evaluations	75.15 %	94.37 %	95.06 %
PLS – PF resolution	70.31 %	83.84 %	84.77 %
G-SEMO – PF resolution	67.01 %	80.07 %	81.37 %

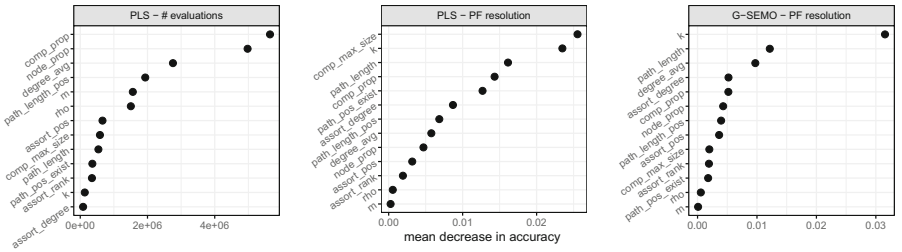


Fig. 6. Relative importance (mean decrease in accuracy) of input variables (instance parameters and PLOS-net features) on the random forest regression model.

4.3 Importance of PLOS-net Features on Search Performance

In this last section, we build a machine learning regression model to predict search performance based on three sets of input variables: (i) instance parameters only (i.e. ρ, m, k), (ii) features from the network only (cf. Sect. 3.2), and (iii) the combination of the two. Given the non-linearity observed on the data, we decide to use random forest as a regression model [2], which also allows us to calculate the relative importance of input variables on the quality of the model. The proportion of variance explained by the random forest regression model for different input variables and search performance measures are reported in Table 1. As we can see, the addition of PLOS-net features as input variables largely improve the regression accuracy, and consequently the predictive power of the regression model. Additionally, the importance of model's input variables [2] are depicted in Fig. 6. For a given search performance measure, the instance parameters and PLOS-net features are sorted in the decreasing order of importance, from top to bottom. Again, the PLOS-net features, especially those related to connectedness of PLOS, appear to have a high importance. By contrast, for the search quality measure, the number of objectives m and the objective correlation ρ has a relatively low importance, whereas the degree of non-linearity k has a relatively high importance. Finally, the number of PLOS, although being relatively important for the regression model, is less important than other PLOS-net features such as the structure of connected components.

5 Conclusions

In this paper, we introduced the Pareto local optimal solutions network model as an alternative to capture the structure of multi-objective landscapes. We also investigated its relation with the performance of multi-objective search algorithms. Our work is to be viewed as the first step towards more systematic investigations on the accuracy of PLOS-nets as a powerful fundamental tool for enhancing our understanding and our practice of multi-objective optimization. In fact, several questions are left open. For example, it would be interesting to test whether our conclusions still hold for other state-of-the-art algorithms and standard multi-objective combinatorial optimization problems. Studying the scalability of the PLOS-net model for large-size problem instances is a challenging issue that should allow us to consider more practical prediction scenarios. It is for instance still unclear how to estimate the proportion of nodes in the PLOS-net and their relative connectedness. Considering adaptation of a simple solution-based Pareto adaptive walk [3] for this purpose can be a promising sampling methodology, which is worth investigating in the future. It would also be interesting to investigate how leveraging PLOS-net features could help improving the performance of the multi-objective optimization algorithms.

Acknowledgments. The authors are thankful to Joshua Knowles and Tea Tušar for fruitful discussions relating to this paper. This research was partially conducted in the scope of the MODŌ International Associated Laboratory, and was partially supported by the French National Research Agency (ANR-16-CE23-0013-01).

References

1. Borges, P., Hansen, M.: A basis for future successes in multiobjective combinatorial optimization. Technical report, IMM-REP-1998-8, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark (1998)
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
3. Daolio, F., Liefooghe, A., Verel, S., Aguirre, H.E., Tanaka, K.: Problem features versus algorithm performance on rugged multiobjective combinatorial fitness landscapes. Evol. Comput. **25**(4), 555–585 (2017)
4. Daolio, F., Verel, S., Ochoa, G., Tomassini, M.: Local optima networks and the performance of iterated local search. In: GECCO, pp. 369–376 (2012)
5. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley, Hoboken (2001)
6. Doye, J.P.K.: The network topology of a potential energy landscape: a static scale-free network. Phys. Rev. Lett. **88**, 238701 (2002)
7. Kauffman, S.A.: The Origins of Order. Oxford University Press, Oxford (1993)
8. Kerschke, P., et al.: Towards analyzing multimodality of continuous multiobjective landscapes. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 962–972. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_90
9. Knowles, J., Corne, D.: Towards landscape analyses to inform the design of a hybrid local search for the multiobjective quadratic assignment problem. In: Soft Computing Systems, vol. 2002, pp. 271–279 (2002)

10. Laumanns, M., Thiele, L., Zitzler, E.: Running time analysis of evolutionary algorithms on a simplified multiobjective knapsack problem. Nat. Comput. **3**(1), 37–51 (2004)
11. Liefooghe, A., Derbel, B., Verel, S., Aguirre, H., Tanaka, K.: Towards landscape-aware automatic algorithm configuration: preliminary experiments on neutral and rugged landscapes. In: Hu, B., López-Ibáñez, M. (eds.) EvoCOP 2017. LNCS, vol. 10197, pp. 215–232. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55453-2_15
12. López-Ibáñez, M., Liefooghe, A., Verel, S.: Local Optimal sets and bounded archiving on multi-objective NK-landscapes with correlated objectives. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 621–630. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_61
13. Ochoa, G., Tomassini, M., Verel, S., Darabos, C.: A study of NK landscapes' basins and local optima networks. In: GECCO, pp. 555–562 (2008)
14. Ochoa, G., Verel, S., Daolio, F., Tomassini, M.: Local optima networks: a new model of combinatorial fitness landscapes. In: Richter, H., Engelbrecht, A. (eds.) Recent Advances in the Theory and Application of Fitness Landscapes. ECC, vol. 6, pp. 233–262. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-41888-4_9
15. Paquete, L., Schiavinotto, T., Stützle, T.: On local optima in multiobjective combinatorial optimization problems. Ann. Oper. Res. **156**(1), 83–97 (2007)
16. Richter, H., Engelbrecht, A.E.: Recent Advances in the Theory and Application of Fitness Landscapes. Emergency, Complexity, and Computation. Springer, Berlin (2014). https://doi.org/10.1007/978-3-642-41888-4
17. Tomassini, M., Verel, S., Ochoa, G.: Complex-network analysis of combinatorial spaces: the NK landscape case. Phys. Rev. E **78**(6), 066114 (2008)
18. Verel, S., Liefooghe, A., Jourdan, L., Dhaenens, C.: On the structure of multiobjective combinatorial search space: MNK-landscapes with correlated objectives. Eur. J. Oper. Res. **227**(2), 331–342 (2013)

Perturbation Strength and the Global Structure of QAP Fitness Landscapes

Gabriela Ochoa[1(✉)] and Sebastian Herrmann[2]

[1] University of Stirling, Stirling, Scotland, UK
gabriela.ochoa@stir.ac.uk
[2] Hochschule RheinMain, Wiesbaden, Germany
sebastian.herrmann@hs-rm.de

Abstract. We study the effect of increasing the perturbation strength on the global structure of QAP fitness landscapes induced by Iterated Local Search (ILS). The global structure is captured with Local Optima Networks. Our analysis concentrates on the number, characteristics and distribution of funnels in the landscape, and how they change with increasing perturbation strengths. Well-known QAP instance types are considered. Our results confirm the multi-funnel structure of QAP fitness landscapes and clearly explain, visually and quantitatively, why ILS with large perturbation strengths produces better results. Moreover, we found striking differences between randomly generated and real-world instances, which warns about using synthetic benchmarks for (manual or automatic) algorithm design and tuning.

Keywords: Local Optima Network · Quadratic Assignment Problem
QAP · Iterated Local Search · Perturbation strength
Fitness landscapes

1 Introduction

The Quadratic Assignment Problem (QAP) requires the assignment at minimal cost of a set of facilities to a set of locations, given the flows between facilities and the distances between locations. The QAP is one of the most difficult combinatorial optimisation problems. Current exact algorithms can solve mostly problems of up to 30 to 40 facilities, therefore, metaheuristics are frequently used to solve larger instances. The most successful are Hybrid Evolutionary Algorithms [1,2] and Iterated Local Search (ILS) with variable perturbation strengths [3,4].

Several studies have analysed QAP fitness landscapes [1,3,5]. The local properties are usually studied through an autocorrelation analysis, while the global structure through a fitness distance correlation analysis. The existence of plateaus and the structure of basins have also been studied [5]. These studies suggest that QAP instances have *unstructured* landscapes. The distances between local optima and the best-known solutions, as well as the average distances between local optima are very close to the landscape diameter (the maximum

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 245–256, 2018.
https://doi.org/10.1007/978-3-319-99259-4_20

possible distance between any two solutions). In some QAP instances, the local optima are neither restricted to a small region of the search space, nor seem to be correlated, QAP landscapes do not always have a 'big-valley'.

Beyond fitness distance correlation analysis, there are no tools available to understand the global structure of fitness landscapes. Local Optima Networks (LONs) [6], fill this gap by providing a compressed model of landscapes, where nodes are local optima, and edges possible transitions among them. They model the distribution and connectivity pattern of local optima, and thus help to characterise the underlying landscape global structure. LONs have been recently used to study the multi-funnel structure of fitness landscapes [7–9]. A funnel refers to a grouping of local optima, forming a coarse-level gradient towards a low cost solution at the end. When sub-optimal funnels exist, search can get trapped and fail to reach the global optimum despite a large computing time.

Iterated local search is a simple yet powerful search strategy. It works by iteratively alternating an intensification stage (local search) with a diversification stage (perturbation or kick). We will refer to the strength of a perturbation as the number of solution components that are modified. Such components are, for example, the number of jobs to move in production scheduling, or the number of edges to interchange in the Travelling Salesperson Problem (TSP) [10]. Recent studies using LONs have shown that increasing the perturbation strength of ILS can 'smooth' the multi-funnel structure, i.e., cause some funnels to disappear or merge, with the effect of improving the algorithm performance [11,12].

For some problems, such as the TSP [12], an appropriate perturbation strength can be small and rather independent of the instance size. ILS implementations to solve the QAP, instead, have shown to benefit from large perturbation sizes. As reported in [10], the best perturbation size depends on the particular instance. For some instances, altering as many as 75% of the solution components produced the best performance. We argue that this behaviour can be better understood by studying the underlying landscape global structure.

The effect of increasing the perturbation strength on the global structure of QAP fitness landscapes has not yet been studied. In this article, we use LONs to characterise and contrast the landscapes of QAP instances of different classes and perturbation strengths. Our contribution is twofold. First, we offer a deeper understanding of why an increased perturbation strength proves advantageous for the QAP. Second, we illustrate, visually and quantitatively, the structural differences between real-world and synthetic QAP instances. Another contribution of this article is a more rigorous description of the notion of funnels, formalising the notion of *monotonic sequences* from the study of energy landscapes in theoretical chemistry [13] to the context of LONs for combinatorial optimisation.

2 The Quadratic Assignment Problem

A solution to the QAP is generally written as a permutation s of the set $\{1, 2, ..., n\}$, where s_i gives the location of item i. Therefore, the search space is of size $n!$. The cost, or fitness function associated with a permutation s is a

quadratic function of the distances between the locations, and the flow between the facilities, $f(s) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{s_i s_j}$, where n denotes the number of facilities/locations and $A = \{a_{ij}\}$ and $B = \{b_{ij}\}$ are the distance and flow matrices, respectively.

Our goal is to visualise and characterise in detail the global structure of QAP fitness landscapes. Therefore, we consider a group of 8 instances from the QAPLIB[1] [14] with sizes ranging from 30 to 42 facilities, which are of moderate size, yet not trivial to solve. Specifically, we selected the largest available real-world instances around this range, and complemented them with instances of similar sizes from the other types. According to [3,15], most QAPLIB instances can be classified into four types. We selected two instances of each type as indicated below.

1. *Uniform random distances and flows.* In these problems, denoted by *tainna*, where *nn* is the problem size, flows and distances are randomly drawn from a uniform distribution. They are known to be the hardest to solve optimally, however, heuristic methods generally find solutions 1 or 2 per cent above the optimum in short computation time [15]. We selected two instances of this group: **tai30a** and **tai35a**.
2. *Random flows on grids.* These problems consider a rectangular tiling of $n_1 \times n_2$ squares of unit size. A location is one of these squares and the distances between them are measured. The flows are randomly generated, but not necessarily uniformly. These problems are known to be symmetrical and to have multiple of 4 ($n_1 \neq n_2$) or 8 ($n_1 = n_2$) different optimal solutions. We selected two instances of this group: **nug30** and **sko42**.
3. *Real-world problems.* These problems arise from practical applications. We briefly describe the instances used in this article. The Krarup and Pruzan instances, denoted by *kra*, contain real world data and were used to plan the Klinikum Regensburg in Germany. In the Steinberg's instances, denoted by *ste*, the goal is to minimise the length of connections between units that have to be placed on a rectangular grid [14]. We selected two instances of this group: **kra30a** and **ste36a** (notice that there are no real-world instances of size around 40 in QAPLIB).
4. *Random real-world like problems.* These instances, denoted by *tainnb*, are randomly generated in a way that they resemble the structure of the real-world instances. We selected two instances of this group: **tai30b** and **tai40b**.

3 Algorithms and Definitions

We use the implementation of ILS by Stützle [3], which allows us to explore the effect of different perturbation strengths. The local search stage uses a first improvement hill-climbing variant with the pairwise (2-exchange) neighbourhood. This operator swaps any two positions in a permutation. The perturbation operator exchanges k randomly chosen items, which corresponds to a random move in the k-opt neighbourhood. Algorithm 1 outlines the pseudo-code.

[1] http://www.seas.upenn.edu/qaplib/.

Algorithm 1. Iterated Local Search (ILS)

Require: Search space S, Fitness function $f(S)$,
Perturbation strength k, Stopping threshold t
1: Choose initial random solution $s_0 \in S$
2: $l \leftarrow \text{LocalSearch}(s_0)$
3: $i \leftarrow 0$
4: **repeat**
5: $s' \leftarrow \text{Perturbation}(l, k)$
6: $l' \leftarrow \text{LocalSearch}(s')$
7: **if** $f(l') \leq f(l)$ **then**
8: $l \leftarrow l'$
9: $i \leftarrow 0$
10: **end if**
11: $i \leftarrow i + 1$
12: **until** i ≥ t
13: **return** l

3.1 Monotonic LON Model

Monotonic LON. Is the directed graph $MLON = (L, E)$, where nodes are the local optima L, and edges E are the monotonic perturbation edges.

Local Optima. We assume a search space S with a fitness function $f(S)$ and a neighbourhood function $N(s)$. A local optimum, which in the QAP is a minimum, is a solution l such that $\forall s \in N(l)$, $f(l) \leq f(s)$. Notice that the inequality is not strict, in order to allow the treatment of neutrality (local optima of equal fitness), which we found to occur in some QAP instances. The set of local optima, which corresponds to the set of nodes in the network model, is denoted by L.

Monotonic Perturbation Edges. Edges are directed and based on the perturbation operator (k-exchange, $k > 2$). There is an edge from local optimum l_1 to local optimum l_2, if l_2 can be obtained after applying a random perturbation (k-exchange) to l_1 followed by local search, and $f(l_2) \leq f(l_1)$. These edges are called *monotonic* as they record only non-deteriorating transitions between local optima. Edges are weighted with estimated frequencies of transition. We determined the edge weights in a sampling process. The weight is the number of times a transition between two local optima basins occurred with a given perturbation (see Sect. 4). The set of edges is denoted by E.

3.2 Compressed Monotonic LON Model

We have observed neutrality at the LON level (i.e. connected sets of optima that share the same fitness value) on several combinatorial problems. This lead us to propose a coarser LON model [9], which compresses the local optima that are connected by neutrality into single nodes. We found this type of neutrality on some QAP instances, specifically, the grid-based and real-world instances (see Sect. 2). Therefore, we considered the compressed model in this study.

Compressed Monotonic LON. Is the directed graph $CMLON = (CL, CE)$, where nodes are compressed local optima CL as defined below. The edges CE are aggregated from the monotonic edge set E by summing up the edge weights.

Compressed Local Optima. A compressed local optimum is a set of connected nodes in the MLON with the same fitness value. Two nodes in the MLON are connected if there is a monotonic perturbation edge between them. The set of connected MLON optima with the same fitness, denoted by CL, corresponds to the set of nodes in the Compressed Monotonic LON model.

Monotonic Sequence. A monotonic sequence is a path of connected nodes $MS = \{cl_1, cl_2, \ldots, cl_s\}$ where $cl_i \in CL$. By definition of the edges, $f(cl_i) \leq f(cl_{i-1})$. There is a natural end to every monotonic sequence, cl_s, when no improving transitions can be found. In the directed CMLON network, cl_s will be a node without outgoing edges (or sink)[2].

Funnel. A funnel can be loosely described a grouping of local optima, conforming a coarse-grained gradient towards a low cost optimum. More formally, we characterise funnels in the CMLON as the aggregation of all monotonic sequences ending at the same point (or sink). Funnels can be seen as basins of attraction at the level of local optima [7].

4 Experimental Setting

Exhaustive enumeration of the search space, and thus complete extraction of the LON models, is not feasible for permutation sizes larger than 10. Our instances have sizes $n \geq 30$, therefore sampling is required. The sampling procedure consists of running an instrumented version of Stützle [3] ILS (described in Sect. 3), where the stopping threshold is set to $t = 10\,000$ iterations without any improvement. This serves the purpose of empirically estimating the funnel ends, i.e., solutions at the end of the ILS trajectory, where escaping is difficult, if not impossible. Our ILS only accepts improving or equal fitness from perturbation moves in order to model monotonic sequences. While running ILS, we store in a set L all the unique optima obtained after the local search stage, and in a set E all the unique edges obtained after a perturbation followed by local search. To construct the MLONs for each instance and perturbation strength, these sets of nodes and edges are aggregated over 200 runs, started from different random configurations. Five perturbation strengths p, corresponding to k-exchanges, are explored (summarised in Table 1), ranging from $k = \frac{n}{8}$ to a complete restart. Once the MLONs are constructed, we proceed to identify the connected components with shared fitness, and thus construct the respective compressed models.

There are multiple performance and network metrics that can be computed and used to understand search difficulty and landscape structure. We selected a minimal subset characterising the key global structural properties. ILS performance is measured with two metrics: (i) success rate, i.e., the proportion of runs

[2] In directed graphs, a node without outgoing edges is called a *sink*.

Table 1. Perturbation strengths.

Flag (p)	1	2	3	4	5
Size (k)	$\frac{n}{8}$	$\frac{n}{4}$	$\frac{n}{2}$	$\frac{3n}{4}$	*restart*

that reached a global optimum, and (ii) the normalised proximity to the global optimum evaluation (i.e. the inverse of the performance gap). These metrics are gathered from 200 independent runs on each instance and perturbation strength.

Table 2. Performance and local optima network metrics.

Perf. metrics	*success*	Success rate of finding a global optimum
	proximity	Normalised proximity to the global optimum evaluation
Network metrics	*noptima*	Number of optima (including both local and global)
	compresssion	Ratio of number of compressed to total number of optima
	pglobal	Proportion of funnels that are globally optimal
	strength	Normalised incoming strength of globally optimal sink(s)

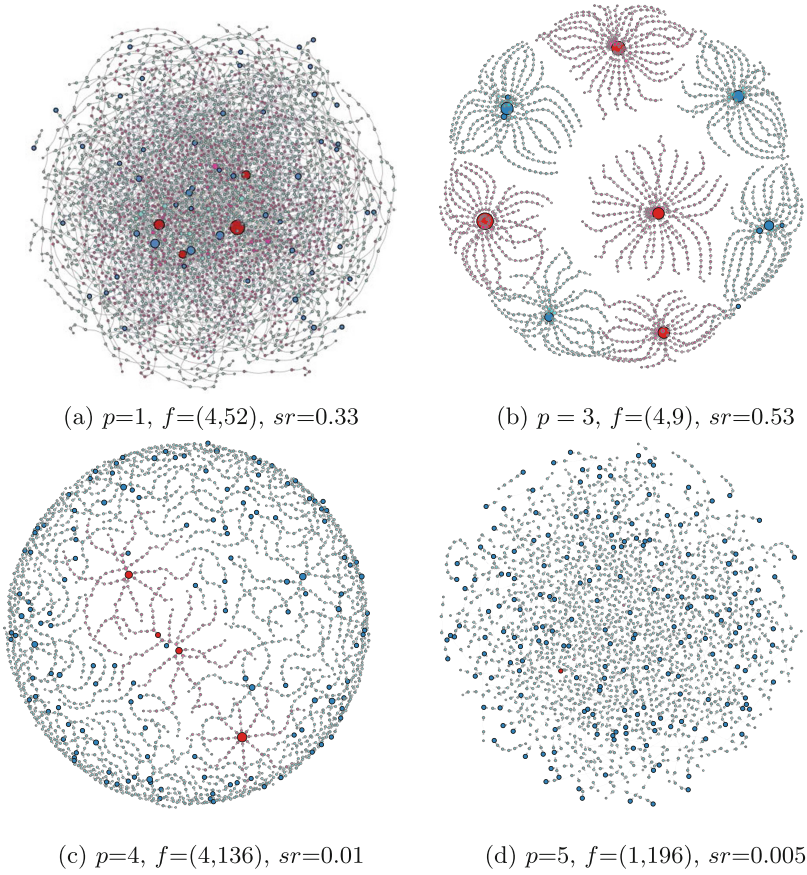
The landscapes' global structure is assessed using four characteristics, summarised in Table 2. The number of optima (*noptima*) is the number of nodes of the MLON model. Since we compute the compressed model CMLON, we report also the ratio of the number of compressed optima to the number of optima (*compression*). For characterising the multi-funnel structure, we measure the proportion of funnels that end at the global optimum level (*pglobal*). This is the ratio of the number of global funnels to the total number of funnels. The centrality of good solutions has been found to correlate with search difficulty [16]. As a measure of the centrality and reachability of the global optima, we compute the normalised incoming strength (weighted degree) of the global optimal sinks (*strength*). This is computed as the sum of the incoming strengths of the globally optimal sinks divided by the sum of the incoming strengths for all sinks.

5 Results

5.1 Visualisation

A first step in analysing the structure of networks is to visualise them. Figure 1 illustrates four compressed MLONs of a single QAP instance, **sko42** at different perturbation strengths $p \in \{1, 3, 4, 5\}$. **sko42** is representative for other QAP

instances and also the largest in our set. The number of optimal and suboptimal funnels and the ILS success rate are also indicated. Each node is a compressed optimum, and edges are monotonic transitions with the corresponding perturbation strength.



(a) p=1, f=(4,52), sr=0.33 (b) p = 3, f=(4,9), sr=0.53

(c) p=4, f=(4,136), sr=0.01 (d) p=5, f=(1,196), sr=0.005

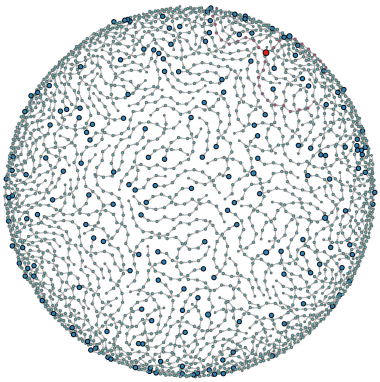
Fig. 1. Compressed monotonic local optima networks for instance sko42 at four different perturbation strengths $p \in \{1, 3, 4, 5\}$. The number of funnels f $(optimal, suboptimal)$ and the ILS success rate sr are indicated. The size of nodes is proportional to their incoming strength. Red highlights the global optimal funnels, while blue suboptimal funnels. Funnel ends (sinks) are indicated in more intense colours. (Color figure online)

The networks in Fig. 1 capture the whole set of sampled nodes and edges for each network, whose sizes range from 1,976 nodes (plot (d)) to 3,234 nodes (plot (a)). Plots were produced with the R statistical language using *force-directed* layout methods as implemented in the igraph library [17]. The decorations reflect

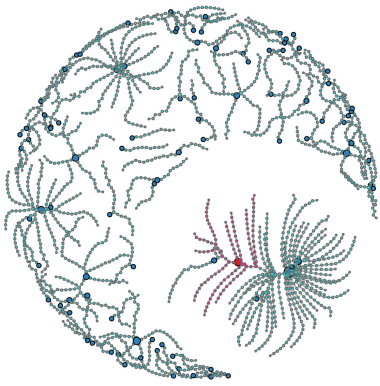
features relevant to search dynamic; the size of nodes is proportional to their incoming weighted degree (strength), which indicates how much a node 'attracts' the search process. Red nodes belong to the funnel(s) containing a global optimum, while blue nodes belong to suboptimal funnels. The funnels' terminating nodes (sinks) are highlighted with a black outline and a more intense colour. A visual inspection reveals clear structural differences among the four perturbation strengths. For the smallest strength ($p = 1$, plot (a)), four global optima can be seen in red, but there is no clear funnel basin structure. However, for $p = 2$ (plot (b)), a clear pattern emerges, showing the basins (i.e. collection of monotonic sequences with a common ending point) of four optimal funnels in red, and a number of suboptimal funnels in blue. The success rate increases to 0.53, the highest for this instance. For a stronger perturbation, ($p = 4$, plot (c)) the performance deteriorates. The four global optimal funnels can still be seen, but their basins become small, and a multitude of sub-optimal funnels appear. Finally, for a complete restart ($p = 5$, plot (d)), only one of the four global optima was found by the sampling process, the number of sub-optimal funnels is almost 200 and the success rate is close to zero.

In order to have a view of the global structure of the remaining three types of QAP instances, Fig. 2 shows the network plots for a uniform random instance (tai30a, plots (a) and (b)), a real-world instance (kra30a, plots (c) and (d)), and a random real-world like instance (tai40a, plots (e) and (f)). Due to space constraints, only two perturbation strengths are shown, $p = 1$ (left plots) and the value of p producing the highest success rate (right plots). The most striking observation from these plots are the structural differences between the randomly generated instances and the real-world instance. The random instances have a single global optimum, whereas the real-world (and indeed the grid based instances, see Fig. 1), generally have several global optima. The random uniform instances (tai30a, plots (a) and (b)) are very difficulty to solve to optimality. Even for the best-performing perturbation strength ($p = 3$, plot (b)), the single global optimum belongs to a tiny funnel, so it is difficult to find. On the other extreme, the random real-world like instances (e.g. tai40b, plots (e) and (f)), are far too easy to solve, reaching 100% success for $p = 3$. Importantly, for both perturbation strengths, a single funnel structure is observed (there are no blue nodes in plots (e) and (f)), which clearly contrasts with the multi-funnel nature of the other instances.

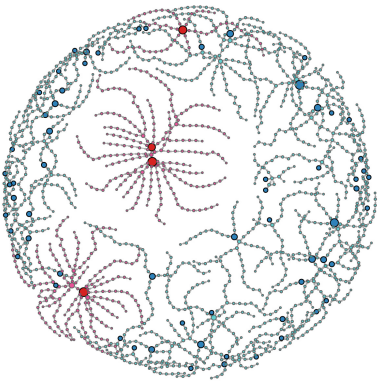
The tai40b networks (plots (e) and (f)) show some intermediate nodes with high strengths (larger pink nodes) in the path to the global optima (red node with black outline). However, there are escape edges from these intermediate attractors, so the perturbation operator is able to overcome them and reach the global optimum. These instances are therefore easy to solve, and the success rate will be 1.0 even for a low perturbation, given enough running time. Regarding the real-world instance kra30a, for a perturbation strength $p = 3$ (plot (d)), the four optimal funnels observed for $p = 1$ (plot (c)) merge into a single funnel with a large incoming strength, explaining the high success rate in this case.



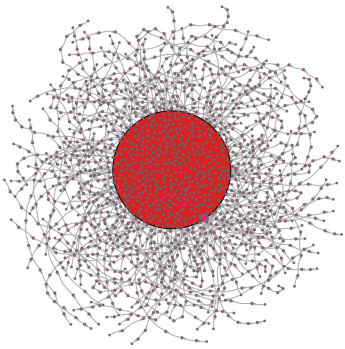
(a) tai30a, p=1, f=(1,188), sr=0.015



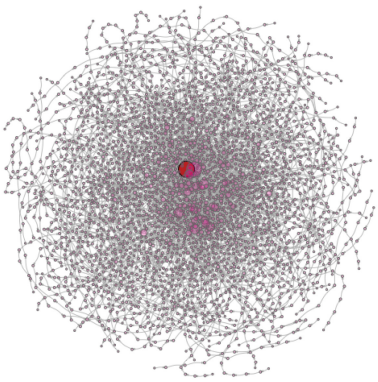
(b) tai30a, p=3, f=(1,94), sr=0.03



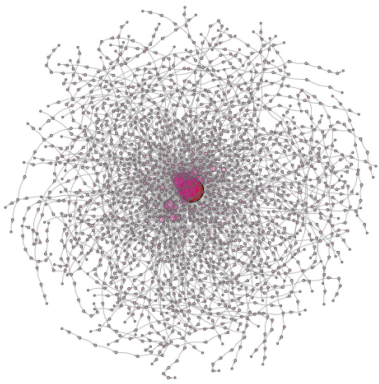
(c) kra30a, p=1, f=(4,63), sr=0.18



(d) kra30a, p=4, f=(1,0), sr=0.97



(e) tai40b, p=1, f=(1,0), sr=0.48



(f) tai40b, p=3, f=(1,0), sr=1.0

Fig. 2. Compressed monotonic local optima networks for instances of the three remaining types and two perturbation strengths, $p = 1$ (left) and the best performing perturbation (right). For more details see caption of Fig. 1

5.2 Structural and Performance Metrics

Figure 3 shows the network and performance metrics described in Table 2 for the eight QAP instances selected. Results were collected for five perturbations strengths as indicated in Table 1. For most instances, the best performance, as measured by both success rate and proximity to the optimum cost, is achieved with an intermediate perturbation strength of $p = 3$, which corresponds to 50% changes to the solution. A perturbation strength of $p = 4$ (75% alterations to the solution), produces the best performance for one of the instances, kra30a. For the random uniform instances (tai30 and tai35), success rates are rather low, however, the solutions found are not far in evaluation from the global optimum

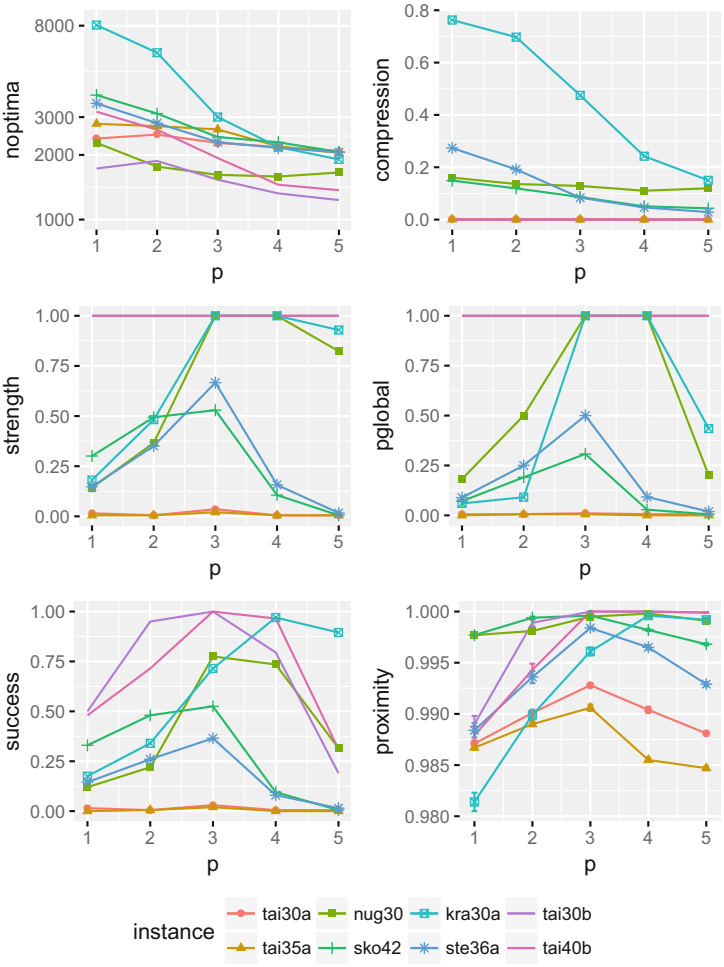


Fig. 3. Structural and performance metrics (as defined in Table 2) for five perturbation strengths (as defined in Table 1).

as indicated by the proximity metric. The network metrics, *strength* and *pglobal*, seem to correlate well with the performance metrics. The reduction of the number of sub-optimal funnels and the increased strength of the global optimal sink, obtained with perturbation strengths $p = \{3, 4\}$, offer an explanation for the increased success rate from the point of view of the landscape global structure.

The most remarkable observations from the plots in Fig. 3 are the notable differences between the randomly generated instances and the real-world and grid-based instances. Looking at the *compression* metric, we can observe that both the random uniform and random real-world like instances show zero compression, which means that there is no neutrality at the level of local optima. The grid-based (nug30 and sko42) and real-world (kra30a and sko42) instances, on the other hand, show neutrality at the local optima level, which decreases with the perturbation strength, indicating that local optima with the same fitness become connected and thus merged. The uniform random instances (tai30a and tai35a) have very low levels of *strength* and ratio of global optimal funnels (*pglobal*), whereas the random real-world like instances (tai30b and tai40b) go to the other extreme, showing a single funnel leading to the global optimum.

6 Conclusion

We have extracted and analysed the compressed monotonic local optima networks induced by iterated local search with different perturbation strengths on QAP instances of different types. Our results confirm that intermediate perturbation strengths of around 50%, or occasionally even 75% alterations to the solution, produce the best performance. This is consistent with previous findings [16]. Our analysis explains this behaviour; a multi-funnel structure generally occurs with low perturbation strengths. With increased perturbation strength, the number of sub-optimal funnel decreases, while the size of the optimal funnels increases, which facilitates reaching the global optimum. We found striking differences between the randomly generated instances, the real-world instances and the grid-based instances. The latter have more than one global optimum, and several local optima sharing the same fitness, which is not the case on the random instances. Moreover, the uniform random instances are very difficult to solve to optimality; they show multiple funnels with very small basins, which do not merge with increased perturbation. On the other extreme, the random real-world like instances are very easy to solve to optimality; their global structure shows a single funnel with a large basin. This is in stark contrast with the multi-funnel structure of the real-world and grid-based instances.

We argue that care should be taken when using randomly generated instances to improve the manual or automatic design of heuristic methods. It is not clear that knowledge extracted from random instances will generalise to real-world instances. Future work will explore the effect of adding crossover, and whether the most effective perturbation strength can be inferred from cheap estimations of the global structure of the underlying landscapes.

Acknowledgements. Thanks are due to Thomas Stützle who shared with us his QAP Iterated Local Search source code.

References

1. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Trans. Evol. Comput. **4**(4), 337–352 (2000)
2. Drezner, Z., Misevicius, A.: Enhancing the performance of hybrid genetic algorithms by differential improvement. Comput. Oper. Res. **40**(4), 1038–1046 (2013)
3. Stützle, T.: Iterated local search for the quadratic assignment problem. Eur. J. Oper. Res. **174**(3), 1519–1539 (2006)
4. Benlic, U., Hao, J.K.: Breakout local search for the quadratic assignment problem. Appl. Math. Comput. **219**(9), 4800–4815 (2013)
5. Tayarani-N, M.H., Prügel-Bennett, A.: Quadratic assignment problem: a landscape analysis. Evol. Intell. **8**(4), 165–184 (2015)
6. Ochoa, G., Tomassini, M., Verel, S., Darabos, C.: A study of NK landscapes' basins and local optima networks. In: Genetic and Evolutionary Computation Conference, GECCO, pp. 555–562. ACM (2008)
7. Herrmann, S., Ochoa, G., Rothlauf, F.: Communities of local optima as funnels in fitness landscapes. In: Genetic and Evolutionary Computation Conference GECCO, pp. 325–331 (2016)
8. Ochoa, G., Veerapen, N.: Mapping the global structure of TSP fitness landscapes. J. Heurist. **24**, 1–30 (2017)
9. Ochoa, G., Veerapen, N., Daolio, F., Tomassini, M.: Understanding phase transitions with local optima networks: number partitioning as a case study. In: Hu, B., López-Ibáñez, M. (eds.) EvoCOP 2017. LNCS, vol. 10197, pp. 233–248. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55453-2_16
10. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, pp. 363–397. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-1665-5_12
11. Herrmann, S., Herrmann, M., Ochoa, G., Rothlauf, F.: Shaping communities of local optima by perturbation strength. In: Genetic and Evolutionary Computation Conference, GECCO, pp. 266–273 (2017)
12. McMenemy, P., Veerapen, N., Ochoa, G.: How perturbation strength shapes the global structure of TSP fitness landscapes. In: Liefooghe, A., López-Ibáñez, M. (eds.) EvoCOP 2018. LNCS, vol. 10782, pp. 34–49. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77449-7_3
13. Berry, R.S., Kunz, R.E.: Topography and dynamics of multidimensional interatomic potential surfaces. Phys. Rev. Lett. **74**, 3951–3954 (1995)
14. Burkard, R.E., Karisch, S.E., Rendl, F.: QAPLIB - a quadratic assignment problem library. J. Glob. Optim. **10**(4), 391–403 (1997)
15. Taillard, E.: Comparison of iterative searches for the quadratic assignment problem. Locat. Sci. **3**(2), 87–105 (1995)
16. Herrmann, S., Ochoa, G., Rothlauf, F.: Pagerank centrality for performance prediction: the impact of the local optima network model. J. Heurist. **24**, 243–264 (2017)
17. Csardi, G., Nepusz, T.: The igraph software package for complex network research. InterJournal, Complex Systems, 1695 (2006). http://igraph.org

Sampling Local Optima Networks of Large Combinatorial Search Spaces: The QAP Case

Sébastien Verel[1(✉)], Fabio Daolio[2], Gabriela Ochoa[2], and Marco Tomassini[3]

[1] EA 4491 - LISIC - Laboratoire d'Informatique Signal et Image de la Côte d'Opale, Université du Littoral Côte d'Opale, 62228 Calais, France
verel@univ-littoral.fr

[2] Computing Science and Mathematics, University of Stirling, Stirling, Scotland, UK

[3] Information Systems Department, Faculty of Business and Economics, University of Lausanne, Lausanne, Switzerland

Abstract. Local Optima Networks (LON) model combinatorial landscapes as graphs, where nodes are local optima and edges transitions among them according to given move operators. Modelling landscapes as networks brings a new rich set of metrics to characterize them. Most of the previous works on LONs fully enumerate the underlying landscapes to extract all local optima, which limits their use to small instances. This article proposes a sound sampling procedure to extract LONs of larger instances and estimate their metrics. The results obtained on two classes of Quadratic Assignment Problem (QAP) benchmark instances show that the method produces reliable results.

1 Introduction

Fitness landscapes are a commonly-used metaphor to describe heuristic search of a globally optimal, or at least of a satisfying solution, among the set of admissible solutions (see Richter and Engelbrecht [1] for a recent review of the state of the art in the field). The number and distribution of local optima in combinatorial fitness landscapes are known to have an impact on the performance of search heuristics. Local Optima Networks (LONs) have been recently proposed as a model of combinatorial landscapes that specifically captures these landscape features [2–4]. In this network model, the nodes are the local optima of the underlying optimisation problem and the edges account for transitions among them using a neighbourhood operator. Modelling combinatorial landscapes as networks brings a whole new set of metrics for capturing the topology and structure of combinatorial search spaces, and provides tools for estimating search difficulty. Most previous work with this model required the full enumeration of the search space in order to extract the nodes and edges of the local optima network, therefore it was applicable only to small problems. We present a sampling methodology for extracting local optima networks of large combinatorial problem instances, and estimating the relevant landscape network metrics

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 257–268, 2018.
https://doi.org/10.1007/978-3-319-99259-4_21

for benchmark instances of the Quadratic Assignement Problem (QAP). The fitness landscape of QAP have been studied several times (for example see the algebraic analysis of the autocorrelation function in Chicano *et al.* [5]). In this work, we propose to increase the number of relevant features to analyse large size fitness landscape that could be potentially be used for performance prediction of QAP algorithms.

The article is structured as follows. The next section briefly overviews previous work on local optima networks, describes the QAP, and the combinatorial landscapes considered. Section 3 describes local optima networks and the metrics employed as features. Section 4 outlines our approach for network sampling and Sect. 5 describes the empirical validation of the obtained estimates. Finally, Sect. 6 summarises our findings and suggest directions for future work.

2 Combinatorial Landscapes

Given a discrete optimization problem, a fitness landscape for its instances is defined as a finite set S of possible solutions, a neighbourhood $\mathcal{N}(s)$ given by the set of solutions that can be reached from any solution $s \in \mathcal{X}$ by applying a simple move operator, and a function $f : S \to \mathbb{R}$ that, given a solution, provides its objective value or fitness [6]. One can define a number of useful concepts such as global and local optima, and basins of attraction among others. It is also possible to define ways in which the search space can be traversed in a random or adaptive way in order to collect configuration space statistics or to improve the current solution.

Starting from the above notions, the *local optima network* (LON) model for combinatorial landscapes was first proposed in [2], with follow up work appearing in [3,7] using Kauffman's NK [8]. Subsequently, more complex and realistic search spaces were studied: the quadratic assignment problem [9], and the permutation flowshop problem [10] which are known to be NP-hard. In a LON, vertices correspond to solutions that are minima or maxima of the associated combinatorial problem, and edges correspond to weighted transitions among them. Initially, weighed edges represented an approximation to the probability of transition between the respective basins in a given direction [2,3,7]. This definition, although informative, produced densely connected networks and required exhaustive sampling of the basins of attraction. A second version, *escape edges* was proposed in [4], which does not require a full computation of the basins. Instead, these edges account for the chances of escaping a local optimum after a controlled mutation (*e.g.* 1 or 2 bit-flips in binary space) followed by hill-climbing. It is this later version that is used here. In order to demonstrate the methodologies proposed in this study, we consider the Quadratic Assignment Problem which is described below.

2.1 The Quadratic Assignment Problem

The Quadratic Assignment Problem [9] is a combinatorial problem in which a set of facilities with given inter-facilities flows has to be assigned to a set of

locations with given inter-locations distances in such a way that the sum of the product of flows and distances is minimised. A solution to the QAP is generally written as a permutation π of the set $\{1, 2, ..., n\}$. The cost associated with a permutation π is given by:

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi_i \pi_j}$$

where n denotes the number of facilities/locations and $A = \{a_{ij}\}$ and $B = \{b_{ij}\}$ are referred to as the distance and flow matrices, respectively. The structure of these two matrices characterises the class of instances of the QAP problem.

The results presented in this article are based on two instance generators proposed in [11] which are in turn inspired by [12] included in the QAPLib. In [11] the generators were devised for the multi-objective QAP, but are adapted here for the single-objective QAP. In order to perform a statistical analysis of the extracted local optima networks, we consider 30 problem instances for each class and size combination.

- *Uniform generator*: produces uniformly random instances where all flows and distances are integers sampled from uniform distributions. The distances are random integer numbers between 0 and 99 (bounds included). The flow matrix is symmetric with random integer entries between 1 and 99. This leads to the kind of problems known in literature as *Tai*nn*a*, being nn the problem dimension [12].
- *Real-like generator*: makes instances where the distance and flow matrices have structured entries. To generate the symmetric distance matrix, N points (integer coordinates) are randomly distributed in a circle of radius 100, and the entries are given by the distances between these N points. The flow matrix is also symmetric with entries following the law $\lceil 10^r \rceil$ where r is a uniform random integer from $[L, U]$. This procedure generates non-uniformly random instances of type *Tai*nn*b* which have the so called "real-like" structure (see QAPLib) since they resemble the structure of QAP problems found in practical applications. The problem instances from the QAPLib are often unique example of a class of problems, so our study considers two parameterisations of real-like instances which allows a statically analysis: *rl1*: $L = -10$ and $U = 5$, *rl2*: $L = -2$ and $U = 4$.

3 Obtaining the Local Optima Networks

Our study considers the permutation representation for QAP solutions. In this case, the most basic neighbourhood structure in the search space is given by the pairwise exchange operation that exchanges any two positions in a permutation, thus transforming it into another permutation.

In what follows, we define how the LON graphs are obtained from the fitness landscapes corresponding to QAP instances.

Nodes: The nodes in the network are local optima (LO) in the search space. For a minimisation problem such as QAP, a solution $x \in \mathcal{X}$ is a local optimum iff $\forall x' \in \mathcal{N}(x), f(x) \leqslant f(x')$. Notice that in this work we do not target specifically neutral fitness landscape with large plateaus. However, this definition of local optima is still relevant for small amounts of neutrality. For fitness landscapes with high levels of neutrality, please refer to the definitions of previous work [13] where the nodes are local optima plateaus. LO are extracted using a best-improvement hill-climber (hc), as given in Algorithm 1. Thereby, when selecting the fittest neighbour (line 4), ties are broken at random.

Algorithm 1. Best-improvement hill-climbing (minimisation)

```
1: procedure HillClimbing
2:     x ← random initial solution
3:     while x ≠ Local Optimum do
4:         set x' ∈ N(x), s.t. f(x') = min_{y∈N(x)} f(y)
5:         if f(x') < f(x) then
6:             x ← x'
7:         end if
8:     end while
9: end procedure
```

Escape Edges: The edges in the network are defined according to a distance function $dist$ and a positive integer $D > 0$. The distance function represents the minimal number of moves between two solutions for a given search (mutation) operator. There is an edge e_{ij} between LO_i and LO_j if a solution x exists such that $dist(x, LO_i) \leqslant D$ and $hc(x) = LO_j$. In other words, if LO_j can be reached after mutating LO_i and running hill-climbing from the mutated solution. The weight $\tilde{w}_{ij}$ of this edge is $\tilde{w}_{ij} = \sharp\{x \in \mathcal{X} \mid dist(x, LO_i) \leqslant D \text{ and } hc(x) = LO_j\}$. That is, the number of LO_i mutations that reach LO_j after hill-climbing. This weight can be normalised by the total number of solutions, $\sharp\{x \in \mathcal{X} \mid dist(x, LO_i) \leqslant D\}$, within reach at distance D: $w_{ij} = \tilde{w}_{ij} / \sum_j \tilde{w}_{ij}$.

Local Optima Network: The weighted local optima network $G_w = (N, E)$ is the graph where the nodes $n_i \in N$ are the local optima, and there is an edge $e_{ij} \in E$, with weight w_{ij}, between two nodes n_i and n_j if $w_{ij} > 0$. According to the definition of weights, w_{ij} may be different than w_{ji}. Thus, two weights are needed in general, and we have a weighted, oriented transition graph.

3.1 Complex Network Metrics

The previous section described how to obtain the LONs. A number of models and statistical metrics have been proposed to study the structure and function of large networks [14]. The first section of Table 1 summarises the metrics for weighted networks considered in this study. First, we introduce some

basic network notation before defining more advanced metrics. Let us denote a_{ij} an element of the graph's *adjacency matrix* A for a weighted oriented graph $G_w = (N, V)$, defined as $a_{ij} = 1$ if $w_{ij} > 0$, $a_{ij} = 0$ if $w_{ij} = 0$. Finally, $k_i = \sum_{j \neq i} a_{ij}$ is the degree of node i, whereas $s_i = \sum_{j \neq i} w_{ij}$ is a generalisation of a node's degree for weighted networks called the node's *strength*. From those basic definitions, we can define the average outdegree, *zout*, as the average of k_i for all nodes.

Disparity of a node n_i measures how heterogeneous are the contributions of the edges of node n_i to the total weight (strength):

$$Y_2(i) = \sum_{j \neq i} \left(\frac{w_{ij}}{s_i} \right)^2$$

Thus, the *average disparity* y_2 is defined as the average for all nodes of $Y_2(i)$.

The standard clustering coefficient [14] does not consider weighted edges. We thus use the *weighted clustering* $c^w(i)$ measure of a node n_i proposed in [15], which combines the topological information with the weight distribution of the network:

$$c^w(i) = \frac{1}{s_i(k_i - 1)} \sum_{j,h} \frac{w_{ij} + w_{ih}}{2} a_{ij} a_{jh} a_{hi}.$$

For each triple formed in the neighbourhood of the node n_i, $c^w(i)$ counts the weight of the two participating edges of the node n_i. The average weighted clustering coefficient *wcc* is defined as $wcc = 1/|N| \sum_{n_i \in N} c^w(i)$. The reader is referred to [15] for more details.

A network is said to show assortative mixing if the nodes in the network that have many connections tend to be connected to other nodes with many connections [16]. Assortativity can be measured using the Pearson correlation coefficient r of degree between pairs of linked nodes. Positive values of r indicate a correlation between nodes of similar degree, whereas negative values indicate relationships between nodes of different degree. We use here the *weighted assortativity*, denoted *knn*, which measures the nearest-neighbours degree correlation. This metric reflects the affinity to connect with high or low-degree neighbours.

The *fitness-fitness correlation* (fnn) measures the correlation between the fitness values of adjacent local optima. It is the Pearson correlation coefficient between the fitness value f_i of node n_i and the weighted-average of it nearest-neighbours fitness, defined as $f_{n,w}(i) = 1/s_i \sum_{j \neq i} w_{ij} f_j$.

4 Sampling Local Optima Networks

Most of the previous work has considered small search spaces (problem sizes up to 18 for binary spaces and up to 10 for permutation spaces), where it was possible to exhaustively enumerate and fully extract the local optima network models. For larger search spaces, i.e., those corresponding to realistic problem

Table 1. Set of features used for network characterisation and for sampling.

Network metrics	
fit	Average fitness of local optima in the network
wii	Average weight of self-loops
zout	Average outdegree, i.e., number of outgoing edges
y_2	Average disparity for outgoing edges
knn	Weighted assortativity
wcc	Weighted clustering coefficient. Measures "cliquishness" of a neighbourhood
fnn	Fitness-fitness correlation. Measures the correlation between the fitness values of adjacent local optima
snowball sampling metrics	
lhc	Average length of hill-climbing to local optima
mlhc	Maximum length of hill-climbing to local optima
nhc	Number of hill-climbing paths to local optima

sizes, a methodology for sampling the local optima networks is required. We propose here an original method for extracting a significant subset of the local optima and transition edges. Thereafter, the network metrics are estimated from the sampled network. The sampling follows a random walk over the local optima network coupled with a snowball process, also known as chain-referral [17]. Snowball sampling is a non-probabilistic technique used in sociology where existing subjects recruit future subjects from among their acquaintances. The sample population then grows like a rolling snowball, similarly to breadth-first search. In the computational implementation, the snowball procedure enlarges an original node sample by joining adjacent nodes. Two control parameters are required: the number of neighbours to consider m (how many acquaintances a recruit should name), and the depth of the sampling d (how many referral steps).

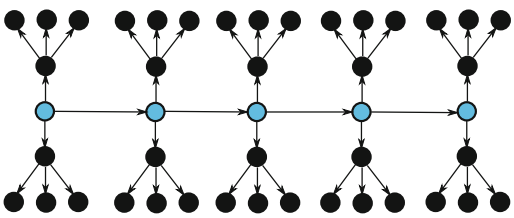


Fig. 1. Illustration of the sampling procedure, featuring a random walk of length $l = 5$, a number of sampled edges $m = 3$, and a sampling depth $d = 2$. The light circles in the center are solutions x_i on the random walk, while dark circles on the outside are solutions sampled during the snowball procedure.

Algorithm 2. Sampling methodology for local optima networks

```
1: procedure LONSampling(d, m, l)
2:     x_0 ← hc(x)                                  ▷ where x is randomly initialised
3:     N̂ ← {x_0}
4:     Ê ← ∅
5:     for t ← 0, ... l − 1 do
6:         Snowball(d, m, x_t)
7:         x_{t+1} ← RandomWalkStep(x_t)
8:     end for
9: end procedure

1: procedure Snowball(d, m, x)
2:     if d > 0 then
3:         for j ← 1, ... m do
4:             x' ← hc(op(x))
5:             N̂ ← N̂ ∪ {x'}                          ▷ Add node to the sample
6:             if (x, x') ∈ Ê then
7:                 ŵ_{x,x'} ← ŵ_{x,x'} + 1
8:             else
9:                 Ê ← Ê ∪ {(x, x')}                 ▷ Add edge to the sample
10:                ŵ_{x,x'} ← 1
11:                Snowball(d − 1, m, x')
12:            end if
13:        end for
14:    end if
15: end procedure

1: procedure RandomWalkStep(x_t)
2:     neighbourSet ← {x : (x_t, x) ∈ Ê ∧ x ∉ {x_0, ..., x_t}}
3:     if neighbourSet ≠ ∅ then                     ▷ Randomly select a neighbour
4:         Select randomly x_{t+1} ∈ neighbourSet
5:     else                                         ▷ Restart from a random solution x
6:         x_{t+1} ← hc(x)
7:         N̂ ← N̂ ∪ {x_{t+1}}
8:     end if
9:     return x_{t+1}
10: end procedure
```

Figure 1 and Algorithm 2 illustrate the local optima network sampling procedure, which requires a hill-climbing algorithm (Algorithm 1), a mutation operator *op*, and a snowball sampling procedure. An initial local optimum is obtained using hill-climbing (*hc*) starting from a randomly generated solution. This initial local optimum is the starting point of the random walk, whose length is controlled by a parameter l indicating the number of steps. At each step of the walk, a snowball procedure is computed as follows: let x_t be the local optimum sampled at step t of the random walk. From x_t, a snowball sampling is performed, by applying m times the mutation operator *op* followed by hill-climbing

to produce neighbouring local optima. Then, the edges and the corresponding weights from x_t are updated. Using recursion, the snowball procedure (with a decreasing depth) is invoked from each adjacent node. For the next step in the walk x_{t+1}, a neighbouring node of x_t that is not already in the walk is selected. If this is not possible (*i.e.* if all x_t adjacent nodes are already in the walk), then x_{t+1} is set as a local optimum obtained from a randomly generated solution, that is $x_{t+1} = hc(x)$ where x is a random solution. The second part of Table 1 summarises the main sampling metrics.

The random walk allows for the estimation of the network metrics that are based on the correlations between neighbouring nodes. The snowball procedure permits the estimation of metrics that require higher-oder interactions (that is, "neighbours of neighbours") such as the clustering coefficient. Moreover, along the sampling, a number of hill-climbing runs are performed, allowing us to opportunistically extract other metrics such as the average length of adaptive walks, the average maximum length of adaptive walks to reach each local optimum of the sampled set, and the corresponding average number of adaptive walks to each local optimum[1]. These sampled values have been used, together with network metrics, to predict the performance of metaheuristics on large problem instances (work to be presented elsewhere).

5 Empirical Validation

In order to validate the sampling methodology, we compared the estimated network metrics against their exact values obtained from previous work on small instances. Various sampling parameters and instance types were considered. We report here the QAP landscape experiments. Three instance types were tested, namely, uniform (uni) and two settings of real-like instances ($rl1$) and ($rl2$) as described in Sect. 2.1. For each type and size, 30 instances were generated. The depth of the snowball sampling procedure was set to $d = 2$. Table 2 summarizes the remaining parameters for both the small and larger instances.

Table 2. Parameters for the empirical validation of the sampling procedure. N: problem dimension. Sampling parameters, m: number of edges, l: size of the random walk.

	Small	Large
Problem size (N)	10	$30, 50, 70, 100, 150$
Sampling (m, l)	$(15, 50), (30, 50), (15, 100)$	$(30, 100), (60, 100), (30, 400)$
op strength (D)	2	4

[1] Here an adaptive walk means that from a given point the walk goes to a randomly chosen neighbor if the neighbor's fitness is better, otherwise it tries another random neighbor.

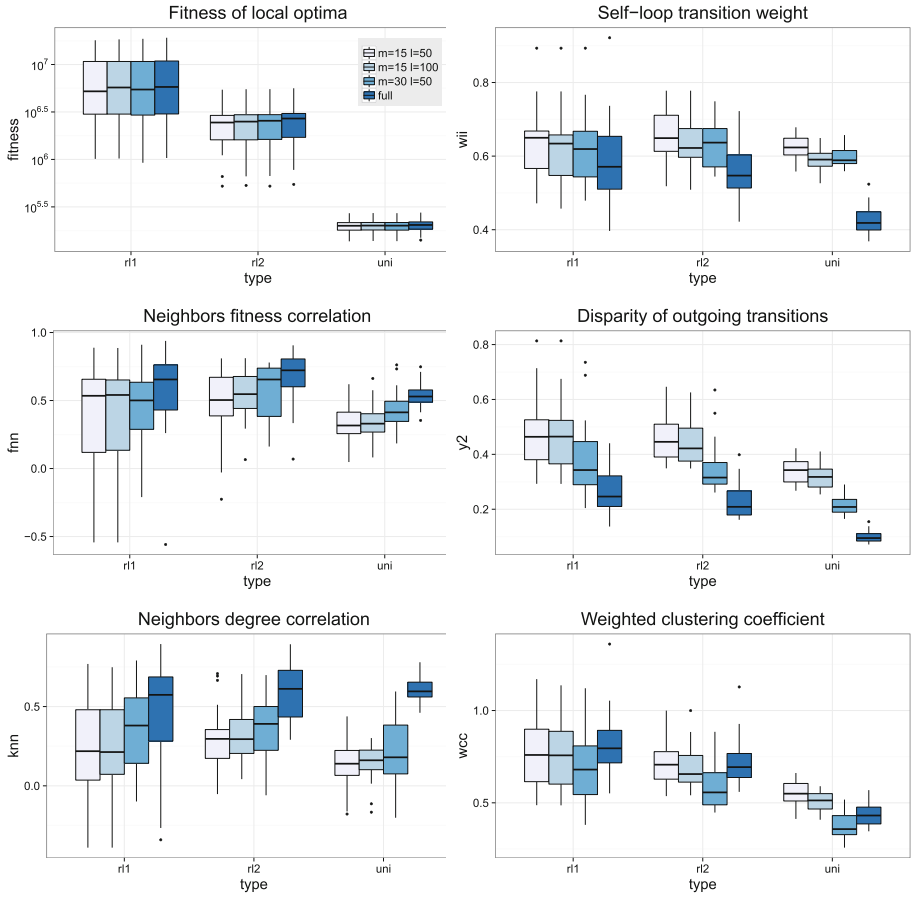


Fig. 2. Small QAP instances ($n = 10$). Estimated network metrics depending on instance type. Boxplots labelled 'full' correspond to the metrics calculated from the complete networks, whereas the remaining boxplots illustrate different sampling parameter pairs (m, l).

The plots in Fig. 2 show the network metrics: *fit, wii, fnn, y2, knn and wcc*, described in Table 1, for the small QAP instances. These metrics have been shown to be significantly related to search performance in previous work [18,19]. The colours in the plot refer to the different sampling parameter pairs (m, l), whereas the curves named 'full' correspond to the metrics calculated from fully enumerated networks, which is only possible for these small instances. The estimated metrics for the different instance classes follows the trend of the full metrics, and their values depend on the instance class. The sampling parameter m has a larger impact on the estimation with higher values producing better estimates. Some metrics are better estimated than others with fitness of local optima and weighted clustering coefficient (wcc) producing the best match.

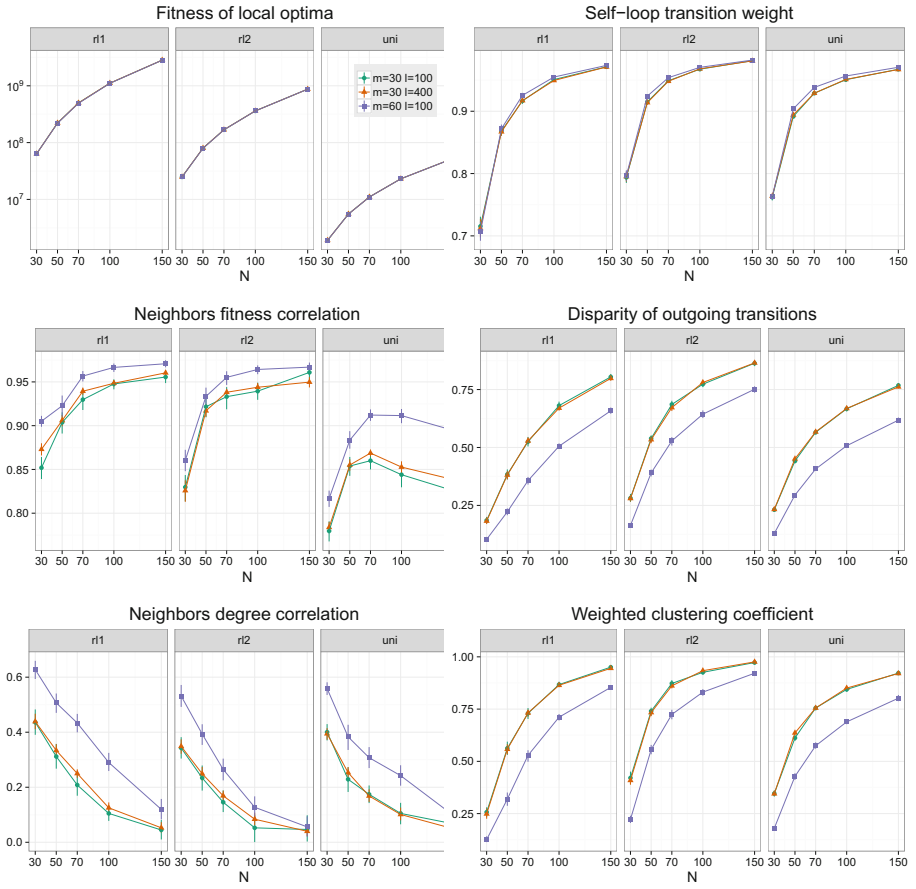


Fig. 3. Large QAP instances. Estimated network metrics as a function of the instance type (*uni, rl1, rl2*) and size ($n \in \{30, 50, 70, 100, 150\}$). The curves represent the sampling parameter pairs (m, l).

Figure 3 shows the estimated network metrics for the larger QAP instances. The curves represent the sampling parameter pairs (m, l), grouped from left to right according to the instance type: *uni, rl1, rl2*. The instance sizes explored, $n \in \{30, 50, 70, 100, 150\}$, are indicated in the X axis labels. The differences among the estimated metrics are small for the three sampling parameter pairs. As for the small instances, the m parameter has a larger impact on the estimation. The curves for $(m, l) \in \{(30, 100), (30, 400)\}$ are almost identical. Therefore, for a fixed computational cost it seems preferable to increase the sampling parameter m (number of edges) rather than the length of the random walk (parameter l). Beyond the quality of the estimation, the main metrics of LON are different according to the class of the QAP instances. For example, the weight clustering coefficient is lowest for uniform type of instance which corresponds to less dense

network; or the self-loop transition weight is highest for the real-like instances of type 2. Without giving all details, the metrics seems to give useful information on the structure of the LON, and problem difficulty. The sampling methodology opens research directions on the performance prediction for large size instance.

6 Conclusions and Future Work

The fitness landscape metaphor has been often used in the context of metaheuristics to search for solutions to difficult problems. In the last few years, we have put forward a novel view of fitness landscapes based on a network called the Local Optima Network which captures fundamental features of the underlying fitness landscape, as well as information about the transitions among local optima basins. In previous work on relatively small instances of typical combinatorial problems we have been able to show that selected LON network statistics correlated with the problem instance difficulty and can be used to predict the performance of well-known metaheuristics on these search spaces. Although it is useful to show these capabilities in principle, the main limitation of the approach was that it required complete enumeration of all local optima in the search space, which of course can only be done for relatively small problem instances. In the present study we have shown that it is possible to sample larger search spaces without losing much in accuracy. This has been done by first comparing sampled and exhaustively enumerated spaces results for small instances, which give similar results, and then extending the procedure to larger sizes. The results obtained on uniformly random, as well as real-like QAP instances, are satisfactory and consistent.

In previous work with small instances, it was found that some network statistics were useful to predict performance [18,19]. As a follow-up work, a similar analysis could be conducted with larger sampled instances in order to find whether LON features could be more correlated with performance than basic fitness landscape features. This will allow to predict performance on larger instances using LON features. Finally, we note that the proposed methodology is not limited to sampling problem instance LON's. The same or a very similar approach could also be used to sample other features of a combinatorial search space. Many aspects remain to be studied and we intend to extend the methodology to other important combinatorial problems and their fitness landscapes. Future work is also planned to extend the sampling method to fitness landscapes that have a significant amount of neutrality.

References

1. Richter, H., Engelbrecht, A.E.: Recent Advances in the Theory and Application of Fitness Landscapes. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-41888-4
2. Ochoa, G., Tomassini, M., Verel, S., Darabos, C.: A study of NK landscapes' basins and local optima networks. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008, pp. 555–562. ACM (2008)

3. Tomassini, M., Verel, S., Ochoa, G.: Complex-network analysis of combinatorial spaces: the NK landscape case. Phys. Rev. E **78**(6), 066114 (2008)
4. Vérel, S., Daolio, F., Ochoa, G., Tomassini, M.: Local optima networks with escape edges. In: Hao, J.-K., Legrand, P., Collet, P., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) EA 2011. LNCS, vol. 7401, pp. 49–60. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35533-2_5
5. Chicano, F., Luque, G., Alba, E.: Autocorrelation measures for the quadratic assignment problem. Appl. Math. Lett. **25**(4), 698–705 (2012)
6. Reidys, C., Stadler, P.: Combinatorial landscapes. SIAM Rev. **44**(1), 3–54 (2002)
7. Ochoa, G., Verel, S., Tomassini, M.: First-improvement vs. best-improvement local optima networks of NK landscapes. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN 2010. LNCS, vol. 6238, pp. 104–113. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_11
8. Kauffman, S., Levin, S.: Towards a general theory of adaptive walks on rugged landscapes. J. Theor. Biol. **128**, 11–45 (1987)
9. Koopmans, T.C., Beckmann, M.: Assignment problems and the location of economic activities. Econometrica **25**(1), 53–76 (1957)
10. Daolio, F., Verel, S., Ochoa, G., Tomassini, M.: Local optima networks of the permutation flow-shop problem. In: Legrand, P., Corsini, M.-M., Hao, J.-K., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) EA 2013. LNCS, vol. 8752, pp. 41–52. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11683-9_4
11. Knowles, J., Corne, D.: Instance generators and test suites for the multiobjective quadratic assignment problem. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Thiele, L., Deb, K. (eds.) EMO 2003. LNCS, vol. 2632, pp. 295–310. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36970-8_21
12. Taillard, E.D.: Comparison of iterative searches for the quadratic assignment problem. Locat. Sci. **3**(2), 87–105 (1995)
13. Verel, S., Ochoa, G., Tomassini, M.: Local optima networks of NK landscapes with neutrality. IEEE Trans. Evol. Comput. **15**(6), 783–797 (2011)
14. Newman, M.E.J.: Networks: An Introduction. Oxford University Press, Oxford (2010)
15. Barthélemy, M., Barrat, A., Pastor-Satorras, R., Vespignani, A.: Characterization and modeling of weighted networks. Phys. A **346**, 34–43 (2005)
16. Newman, M.E.: Assortative mixing in networks. Phys. Rev. Lett. **89**(20), 208701 (2002)
17. Goodman, L.: Snowball sampling. Ann. Math. Stat. **32**(1), 148–170 (1961)
18. Chicano, F., Daolio, F., Ochoa, G., Vérel, S., Tomassini, M., Alba, E.: Local optima networks, landscape autocorrelation and heuristic search performance. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012. LNCS, vol. 7492, pp. 337–347. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32964-7_34
19. Daolio, F., Verel, S., Ochoa, G., Tomassini, M.: Local optima networks and the performance of iterated local search. In: Genetic and Evolutionary Computation Conference, GECCO 2012, pp. 369–376. ACM Press (2012)

Algorithm Configuration, Selection, and Benchmarking

Algorithm Configuration Landscapes: More Benign Than Expected?

Yasha Pushak[1(✉)] and Holger Hoos[1,2]

[1] Department of Computer Science, The University of British Columbia, Vancouver, Canada
ypushak@cs.ubc.ca
[2] LIACS, Universiteit Leiden, Leiden, The Netherlands
hh@liacs.nl

Abstract. Automated algorithm configuration procedures make use of powerful meta-heuristics to determine parameter settings that often substantially improve the performance of highly heuristic, state-of-the-art algorithms for prominent $\mathcal{NP}$-hard problems, such as the TSP, SAT and mixed integer programming (MIP). These meta-heuristics were originally designed for combinatorial optimization problems with vast and challenging search landscapes. Their use in automated algorithm configuration implies that algorithm configuration landscapes are assumed to be similarly complex; however, to the best of our knowledge no work has been done to support or reject this hypothesis. We address this gap by investigating the response of varying individual numerical parameters while fixing the remaining parameters at optimized values. We present evidence that most parameters exhibit uni-modal and often even convex responses, indicating that algorithm configuration landscapes are likely much more benign than previously believed.

1 Introduction

Automated algorithm configuration procedures are able to find configurations that are often substantially better than expert-determined default settings. Current methods are heavily-based on meta-heuristics (such as ParamILS [12], an iterated local search procedure; GGA [2], a gender-based genetic algorithm, and SMAC [9], a model-based stochastic search algorithm), which are typically used to solve $\mathcal{NP}$-hard combinatorial optimization problems with complex search landscapes. To the best of our knowledge, the properties of the algorithm configuration search landscapes have not been investigated so far. In this work, we conduct such an investigation, focusing on numerical parameters, which prominently occur in many algorithm configuration scenarios, and provide evidence that these configuration landscapes are more benign than one might assume.

As a motivating example, consider the problem of configuring a stochastic local search algorithm A with a numerical parameter p that controls the trade-off between intensification and diversification. Let us assume that low values result in low intensification, and high values in high intensification. Intuitively,

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 271–283, 2018.
https://doi.org/10.1007/978-3-319-99259-4_22

one would expect a single optimal setting to exist for this parameter; for lower values of p, the performance of A would deteriorate due to insufficient diversification (resulting in stagnation behaviour), and for higher values of p, insufficient intensification would degrade the performance of A. Therefore, the response of A's performance to p would be uni-modal, perhaps even convex. We hypothesize that many – perhaps: most – numerical parameters have similar characteristics.

Specifically, in this work, we investigate two research questions regarding the way the performance of a given algorithm depends on its parameter settings. **RQ 1. When all other parameters are fixed, are the responses of individual numerical parameters uni-modal or convex; if so, how often?** Here, the response of a parameter p refers to the function mapping values of p to the performance of the given algorithm. In the context of automated configuration of a given target algorithm A, the performance of A is assessed (and optimized) on a set I of problem instances. Following many state-of-the-art configurators [9,12,14], we asses the performance of A using PAR10 on I, *i.e.*, mean running time with timed-out runs counted at 10 times their running time cutoff. Ideally, the response of A to its parameters would be identical for all instances in I; however, we cannot assume that this will always be the case. This gives rise to our second research question: **RQ 2. When all other parameters are fixed, are the responses of individual numerical parameters uni-modal or convex on individual instances; if so, how often?** To obtain robust estimates for the performance of A on each $i \in I$, we took medians over 10 independent runs per instance. We note that the aggregation of performance over a set of instances (as in RQ 1) could lead to more complex parameter responses – *i.e.*, a negative answer to RQ 1 – even if the responses on individual instances were benign. However, more likely, aggregation should have a smoothing effect on the parameter responses, so that a negative answer to RQ 2 might still be consistent with a positive answer to RQ 1.

The answers to these research questions matter, because existing configurators make only weak assumptions about the landscapes they search. Indeed, the only assumption made by well-known, high-performance configurators, such as SMAC [9], ParamILS [12] and GGA [2], is that the performance of one configuration is likely to be correlated with the performance of nearby configurations. Of the configurators we consider here, only irace [14] (by the nature of the generative probabilistic model it uses to sample promising configurations) assumes a smooth response for numerical parameters. Additional structural insights, such as uni-modality or convexity of individual parameter responses, could at least in theory be exploited to substantially improve configurator performance.

In the literature on combinatorial optimization and meta-heuristics, landscape analysis is a well-established topic. Two particularly prominent approaches are based on the analysis of fitness-distance correlation and of landscape correlation functions (see, *e.g.*, Chap. 5 of [8] and the references therein). When used in empirical work for the characterization of landscapes, both approaches assess global landscape properties, while our work is focused on local properties. Furthermore, to yield reasonably accurate results, they both require large sets of

samples from a given landscape, which, in the context of algorithm configuration, are expensive to obtain (every sample involves many runs of the given target algorithm, one for each problem instance in the given training set.) Finally, both approaches require normalization to deal with parameters whose domains show large differences in range or number of values (for discrete parameters), as frequently encountered in typical algorithm configuration scenarios, and have difficulties dealing with integer-valued parameters with small ranges (such as `BACKBONE_TRIALS` for LKH [7] considered in our study).

Somewhat related to our work, in the algorithm configuration literature, there has been a recent focus on analyzing parameter importance, *i.e.*, the degree to which individual parameters affect the performance of a given algorithm. Importance analysis approaches such as ablation analysis [6], fANOVA [11] and forward selection based on empirical performance models [10] are orthogonal to our work, since they quantify the impact of parameters on algorithm performance, but do not provide direct insights into the shape of the parameter responses or structural aspects of the configuration landscapes arising from these shapes.

The remainder of this paper is structured as follows. In Sect. 2, we explain our methods for investigating the structure of configuration landscapes. Then, in Sect. 3, we introduce the setup used for our experimental investigation. The results of this investigation are presented in Sect. 4, and finally, in Sect. 5, we summarize our findings and outline several avenues for future work.

2 Methods

Our approach to analyzing configuration landscapes is severely constrained by the fact that for typical configuration scenarios, obtaining performance measurements for all configurations or even sampling a substantial fraction of the landscape would be prohibitively expensive. For example, the smallest configuration scenario we study (which involves two integer-valued parameters for the TSP solver, EAX [17]), would take over 500 CPU years on our reference machines to obtain complete evaluation of the corresponding configuration landscape. Configuration spaces grow exponentially with the number of parameters, so even relatively sparse samples quickly become unaffordable. Therefore, to perform our analysis of configuration landscapes, we restricted ourselves to a small number of configurations. In the light of this, and consistent with the aims of our investigation, we focused on individual slices of the parameter responses.

2.1 Parameter Response Slices

Given a target algorithm A, a *response slice for parameter* p is obtained by fixing all other parameters of A to some value and measuring the performance of A as a function of p. Intuitively, this corresponds to a slice through the configuration landscape of A, and technically, it can be seen as a conditional response, subject to all other parameters taking a specific value. Since we are primarily interested in the parameter responses near high-quality configurations, we first performed

25 independent runs of SMAC [9] for each scenario (configuring both numerical and categorical parameters), and subsequently evaluated the resulting configurations on the entire training set. We then used the best-performing configuration on the training set as the centre point for each parameter slice.

Even evaluating every possible value for each parameter response slice in our EAX scenario (described in more detail in Sect. 3) would take 6 CPU years, so we further reduced our slices to only 15 parameter values each. If a parameter had less than 15 possible values, then we used all of them; otherwise, to obtain high resolution near the best-known parameter setting, we increased the spacing between adjacent values exponentially with increasing distance from the best-known value. We added an additional constraint to ensure that we obtained some coverage of the parameter response on either side of the best-known parameter setting: we restricted at most 75% of the points to be on one side of the best-known value (note that if the best-known value took the maximum or minimum allowed value, we could not enforce this constraint). Since there were still many possible locations for the points satisfying these constraints, we multiplied the grid of points by a randomly chosen weight to choose their exact location. For any integer-valued parameter, we first determined a set of values as for real-valued parameters, and then rounded each setting thus obtained to the nearest valid and previously unused value. Finally, to obtain robust performance estimates for each value in a given parameter slice, we performed 10 independent runs per instance and determined a median performance value from these.

2.2 Bootstrap Analysis and Confidence Intervals

To account for the variance between independent target algorithm runs and problem instances, we used a nested bootstrap procedure similar to the one by Mu *et al.* [15] with 1 001 outer and inner bootstrap samples. To be precise, for each parameter value and problem instance, we created 1 001 (inner) bootstrap samples of the 10 independent runs to obtain a distribution of median performance values; from these, we determined 95% bootstrap confidence intervals. Next, we created 1 001 bootstrap samples of the instance set, and for each occurrence of an instance in a sample we sampled at random from the corresponding distribution of median performance values. Finally, we calculated medians and 95% confidence intervals for the performance observed at each parameter value. We used Bonferroni multiple testing correction when calculating confidence intervals, since each slice had up to 15 parameter values, and linear interpolation to estimate confidence intervals between adjacent parameter values.

2.3 Tests for Convexity and Uni-modality

We designed two testing procedures to determine if there is sufficient evidence to reject the hypotheses of convexity and uni-modality of a given parameter response slice with 95% confidence. These tests attempt to fit a piece-wise linear curve that is constrained to be uni-modal or convex, respectively, through the previously described bootstrap confidence intervals. We say that a test rejects

uni-modality or convexity if no such line exists. If the upper bound of a parameter value was censored, we excluded it from the test, since there is insufficient information to properly reason about it. When considering individual instance response slices (RQ 2), if there were too few uncensored data points to perform a test, we considered it to be insufficient data to reject the hypotheses of convexity or uni-modality. For aggregate response slices (RQ 1), we used PAR10 scores, counting censored runs at 10 times our running time cutoff).

2.4 Identifying "Interesting" Parameter Response Slices

Parameters with almost flat responses (*i.e.*, robust ones, whose settings have little or no effect on the performance of the algorithm) are of limited interest to our investigation. We therefore used a simple heuristic procedure to identify parameters with interesting (*i.e.*, non-flat or sensitive) responses, based on the sizes of and overlap between the bootstrap confidence intervals for each value in the respective parameter response slice. To be precise, we define a parameter's response to be *interesting* if the size (in terms of the log of the performance measure) of the overlap between the two confidence intervals with the least amount of overlap is at most half of the average size of the confidence intervals.

2.5 Counting the Number of Modes (Local Minima)

One commonly used feature to describe the ruggedness of a search landscape is the number and density of local optima (see, *e.g.*, Chap. 5 of [8]). We partially capture this notion of ruggedness by counting the number of modes that occur along a parameter response slice. To do this, we use a very similar procedure to our tests for uni-modality and convexity: we fit the flattest possible piece-wise linear curve within the region defined by the 95% confidence intervals along a given parameter response slice and then count the number of modes in that line.

2.6 Fitness Distance Analysis

Since traditional fitness distance analysis would have been too expensive, considering the constraints on our computational budget, we applied it locally to the sets of data points belonging to each parameter response slice. We also calculated the FDC for each bootstrap sample of a parameter response slice to obtain medians and confidence intervals for each slice.

3 Experimental Setup

We studied 10 different algorithm configuration scenarios, spanning three widely studied, $\mathcal{NP}$-hard problems (SAT, MIP and TSP), 6 prominent algorithms for these and 5 well-known instance sets. All of these scenarios involve the minimization of running time, measured in terms of PAR10, *i.e.*, mean running time with timed-out runs counted at 10 times the running time cutoff. In Table 1,

Table 1. The instance sets we studied from ACLib scenarios and the configuration budgets and training/testing running time cutoffs we used for their scenarios.

Problem	Instance set	Configuration budget [CPU days]	Training running time cutoff [CPU sec]	Test running time cutoff [CPU sec]
SAT	Circuit-fuzz	2	300	600
	BMC08	2	300	600
MIP	CLS	2	10000	10000
	Regions200	2	10000	10000
TSP	tsp-rue-1000-3000	1	86	3600

we summarize the configuration budgets and running time cutoffs used for our scenarios. All instance sets are readily available online in ACLib scenarios that have been identified as interesting and challenging benchmarks for algorithm configurators [13]. For the SAT and MIP instance sets, we used the same budgets and running time cutoffs as specified in the corresponding ACLib scenarios. We increased the running time cutoff for the test set (and parameter slices) for the SAT and TSP scenarios, in order to better assess poorly performing configurations.

Table 2. The 6 algorithms we studied. *We configured all 185 numerical parameters, but only studied slices for the 10 most important (see text for details).

Problem	Algorithm	Version	Numerical parameters	Categorical parameters
SAT	CaDiCaL	sc17	40	22
	Lingeling	azf	185*	137
	Cryptominisat	4.1	22	36
MIP	CPLEX	12.6	22	52
TSP	EAX + restart	JDL	2	0
	LKH + restart	2.0.7	12	9

Table 2 provides an overview of the 6 algorithms we studied. We introduce a few new, state-of-the-art algorithms not found in existing ACLib scenarios.

For SAT, we studied CaDiCaL [3], because it was one of the top-performing, configurable solvers in the application track of the 2017 SAT competition; lingeling [3], because it was the winner of the 2014 Configurable SAT Solver challenge on the circuit-fuzz and BMC08 instances; and cryptominisat [18], because it is a variant of the well-known and commonly used minisat algorithm. Reference implementations of lingeling and cryptominisat were directly obtained from ACLib 2.0, whereas that of CaDiCaL was taken from the 2017 SAT competition.

For TSP, we chose two extensively studied [4,16], state-of-the-art, inexact solvers: EAX [17] and LKH [7]. We used the same implementations as Mu *et al.* [16] and Dubois-Lacoste *et al.* [4], which were modified to use a restarting mechanism and terminate upon reaching optimal solution quality values (known from long runs of an exact solver). The TSP scenarios in ACLib configure for solution quality, so we chose these solvers to focus on running time minimization.

For MIP, we studied the high-performance commercial solver IBM ILOG CPLEX [1], version 12.6 (featured in several ACLib scenarios), which terminates upon finding an optimal solution to a given MIP instance and completing a proof of optimality. We slightly modified the CPLEX scenarios from ACLib, by treating CPLEX as a randomized algorithm. Earlier versions of CPLEX used a fixed random seed that was not exposed to the user; however, CPLEX is in fact a randomized solver, and treating it as such avoids potential problems arising from bias due to the use of a specific random seed.

For every algorithm except lingeling we were able to evaluate parameter slices for all of their numerical parameters; however, since lingeling had so many parameters, we restricted our analysis to a subset of them. Falkner *et al.* [5] reported the 10 most important parameters according to fANOVA [11] (all of which were numerical) for lingeling on the circuit-fuzz instance set, so we only used these 10. We also slightly modified the ranges for a few parameters for LKH and CPLEX. Some of the numerical parameters use values 0 or -1 to encode special behaviour, *e.g.*, the automatic setting of the parameter value or deactivation of the mechanism controlled by the parameter. In cases where the documentation was unclear, we erred on the side of caution and removed a parameter value or treated the special value as a categorical parameter.

In Table 3, we show the results from configuring our 10 scenarios, using 25 runs of SMAC [9] per scenario. These results are consistent with the literature. We note that in some cases, configuration did not result in significant performance improvements over the default parameter settings of a given algorithm; this is unproblematic, since our goal in performing automated configuration was not to obtain improved performance, but rather to ensure we used high-performance configurations as reference points for the parameter response slices that formed the basis for our configuration landscape analysis.

We ran all of our experiments on Ada, a cluster of 20 nodes, equipped with 32 2.10 GHz Intel Xeon E5-2683 v4 CPUs with 40960 KB cache and 96 GB RAM each, running openSUSE Leap 42.1 (x86_64). To minimize detrimental cache effects and memory contention, in all experiments, we used a single core per CPU and limited RAM use to 3 GB. In total, we used 43.5 CPU years for automated configuration and collection of parameter response slice data.

4 Results

We collected and analyzed 193 parameter slices for instance sets and individual problem instances, as motivated in Sect. 1 and outlined in Sects. 2 and 3.

Table 3. PAR10 values on the test sets for the default configuration versus the configuration with the best training PAR10. All times are in CPU seconds.

Problem	Algorithm	Instance set	Default PAR10	Configured PAR10	Speedup
SAT	CaDiCaL	Circuit-fuzz	468.71	252.34	1.86
		BMC08	638.57	637.93	1.00
	Lingeling	Circuit-fuzz	382.23	279.29	1.37
		BMC08	692.80	691.51	1.00
	Cryptominisat	Circuit-fuzz	444.68	276.83	1.61
		BMC08	938.61	970.07	0.97
MIP	CPLEX	CLS	40.39	3.39	11.91
		Regions200	106.77	6.40	16.68
TSP	EAX	tsp-rue-1000-3000	65.99	56.84	1.16
	LKH	tsp-rue-1000-3000	428.60	228.62	1.87

4.1 RQ 1. Uni-modality and Convexity on Instance Sets

Overall, the parameter response slices for instance sets appear to be more benign than one might expect. Our tests for uni-modality and convexity failed to reject for all but 1 of the 193 parameter slices. That is, 99.48% of the slices we measured appear to be both uni-modal and convex. Somewhat surprisingly, our heuristic method outlined in Sect. 2.4 identified only 18 of the slices as interesting. In Fig. 1, we show 4 parameter response slices that are representative of the qualities we observed in this set of 18 (the remaining 14 are available in our online, supplementary material, available at http://ada.liacs.nl/projects/ac-landscapes). To our surprise, neither lingeling nor cryptominisat had any interesting parameter response slices. The parameter that fANOVA rated to be the most important for lingeling [5] shows a slight dip at the smallest parameter value in the slice for the circuit-fuzz instance set. To investigate further, we evaluated an additional 15 parameter values, but still found it to be un-interesting according to our criterion.

The only parameter we found having a non-unimodal and non-convex response was LKH's `BACKBONE_TRIALS` parameter (see Fig. 1), which specifies the number of backbone trials in each run. Apart from `BACKBONE_TRIALS` $= 0$, even this response slice appears to be convex and uni-modal. To the best of our knowledge, a value of 0 does not have special meaning, apart from the obvious semantic difference of *some* versus *no* backbone trials, which may alone account for this difference, since it likely corresponds to a (poorly performing) heuristic component of the algorithm that is turned on or off.

Some of the parameter responses (*e.g.*, `keepglue` in Fig. 1), appear to be flat for poorly performing parameter values, and hence non-convex overall. Our tests were unable to reject convexity, despite this visual evidence, because of the relatively wide bootstrap confidence intervals. However, we believe that these

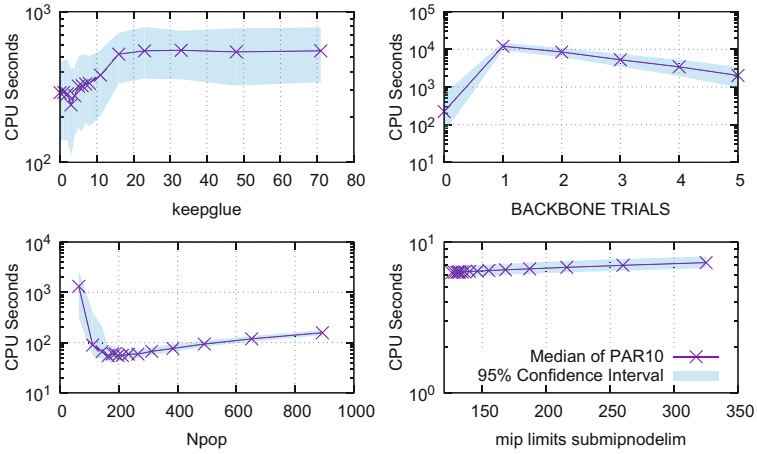


Fig. 1. Four parameter response slices. From left to right top to bottom: CaDiCaL's `keepglue` on the circuit-fuzz instance set, LKH's `BACKBONE_TRIALS` on the tsp-rue-1000-3000 instance set, EAX's `Npop` on the tsp-rue-1000-3000 instance set and CPLEX's `mip_limits_submipnodelim` on the Regions200 instance set.

flat regions are an artifact of how PAR10 scores treat censored runs, and that a sufficiently large running time cutoff would yield convex responses.

Interestingly, in the three SAT scenarios involving the BMC08 instance set, we found only one parameter response slice considered interesting according to our criterion: CaDiCaL's `restartmargin`. This is consistent with the fact that SMAC was unable to achieve significant performance improvements for these scenarios. We further note that the default value for `restartmargin` is very near to the best-known value. Hence, it appears that better configurations may not exist rather than being hard to find due to highly irregular or rugged landscapes.

4.2 RQ 2. Uni-modality and Convexity on Individual Instances

To study our second research question, we ran our tests for convexity and uni-modality on the parameter response slices for each individual problem instance. We consider the parameters independently by looking at statistics of their responses on each instance. For example, on the left pane of Fig. 2 we plot a cumulative distribution function (CDF) showing on the y-axis the percentage of parameters that had convex responses on a percentage of instances less than or equal to the value specified on the x-axis. Surprisingly, there is a large percentage of parameters with convex responses slices for most instances. However, nearly half of the parameters with interesting response slices on the entire instance set tend to have much fewer convex parameter responses on the individual instances. Our procedure (outlined in Sect. 2.3), sometimes assumes uni-modality or convexity when there is insufficient data to perform a test. On average, over all parameters, this happened for only 6.2% of the instances we considered, and at

most, on 16.9% of the instances. Hence, even if all of these cases were instead assumed to be non-unimodal or non-convex, our overall results would not be substantially different.

Furthermore, looking at the CDF of the average numbers of modes for each instance parameter slice on the right pane of Fig. 2, we see that just under 50% of the interesting parameters have an average of more than one mode for their individual instance responses. On the other hand, most of the parameters have an average of only one mode per instance, which is consistent with the fraction of parameters with primarily convex instance response slices.

Overall, there are a surprisingly large number of parameter response slices that are both uni-modal and convex on most or all of their individual instances. In Table 4, we show a summary of these results, in addition to the corresponding results for the parameter responses on entire instance sets. Note that for the aggregate instance set parameter responses, we show the percentage of uni-modal and convex parameter response slices observed on different instance sets, whereas for the individual instances, we first computed the percentage of instances with uni-modal and convex responses for each parameter, and then report the average percentages over the set of all parameters.

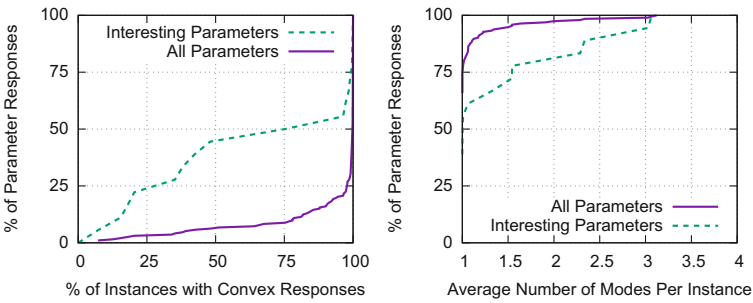


Fig. 2. CDFs summarizing our findings for individual instances. Left: for each parameter we computed the percentage of instances on which it had a convex response, and then we plot the CDF of these percentages; right: the CDF of the average number of modes observed in the responses for a parameter on each instance.

Our analysis of the fitness distance correlation coefficient (FDC) for the parameter response slices supports our hypothesis that parameter responses on individual instances are more rugged than the aggregate responses on entire instance sets. In particular, we found that 80% of the parameters have an average instance response slice FDC less than 0.25, compared to 0.4 for the instance set responses. However, through manual inspection of the instance parameter slices, we found that some responses obtained low FDC scores simply because they are relatively flat (hence deviations in parameter value have low correlation with deviations in algorithm performance). Still, the high average numbers of modes observed for some of the parameters indicate that these responses are truly rugged.

Table 4. Left: the percentages of uni-modal and convex parameter response slices on entire instance sets; right: we computed the percentage of instances with convex or uni-modal responses for each parameter, and then show the average percentages over the parameters, i.e., we show the percentage of convex and uni-modal instance responses for the "average parameter".

	Instance set		Individual instances	
	% Uni-modal parameters	% Convex parameters	Average % uni-modal instances	Average % convex instances
All parameters	99.5	99.5	95.3	92.6
Interesting parameters	94.4	94.4	76.1	66.1

To check that these were not spurious results, we performed exact replicates for three scenarios that were near the Pareto front of the largest average number of modes and the smallest average FDC: CaDiCaL's `posize` and `elimint` on circuit-fuzz instances, and CPLEX's `mip_limits_cutpasses` on CLS instances. Then, for each parameter, we chose three instances near to their respective Pareto fronts. In all cases, the replicates were qualitatively identical to the original ones. Additional details on these experiments and our FDC analysis can be found at http://ada.liacs.nl/project/ac-landscapes.

5 Conclusions and Future Work

Overall, we found strong support for our hypothesis that parameter responses on instance sets tend to be uni-modal and convex. We also found evidence that many parameters have convex and uni-modal responses on individual problem instances; however, these responses tend to be (in some cases substantially) more rugged than their aggregate counterparts. We were surprised to find that a small percentage of parameters appear to have highly rugged responses on most of the instances. However, even though these parameters have rugged response slices on individual instances, their aggregate responses still tend to be uni-modal and convex on the entire instance set, after performing bootstrap sampling to account for the variability over instances. This may be why the simple Gaussian model used for generating promising configurations in irace [14], which inherently exploits smoothness in individual parameter responses, works rather well.

Our results do not preclude the possibility of complex parameter interactions that result in configuration landscapes with many local optima. Future work could study parameter interactions and investigate whether or not local minima are rare, or at least easy to escape. Categorical parameters also play an important role in many algorithm configuration scenarios. Here, we set categorical parameters to values found in high-quality solutions; however, it would be

interesting to explore whether similar results to those we reported here hold for other settings of the categorical parameters of a given target algorithm. Moreover, it would be very interesting to investigate to which extent our findings interact with parameter importance, as assessed by fANOVA and ablation analysis.

We note that our method for collecting data is focussed on high-performance regions of the configuration spaces we considered, as is the search process of algorithm configurators. Therefore, our results may only hold in these regions; this would be at least of theoretical interest and could be investigated in future work. Another direction is to extend our analysis to scenarios that involve optimization of solution quality, such as loss scenarios involving hyperparameter optimization of machine learning algorithms. Finally, and perhaps most interestingly, we strongly believe that our results open the door to designing new algorithm configuration procedures that exploit the relatively benign characteristics of typical configuration landscapes discovered in this work.

Acknowledgements. YP was supported by an NSERC Vanier Scholarship. HH acknowledges funding through an NSERC Discovery Grant, CFI JLEF funding and startup funding from Universiteit Leiden.

References

1. IBM Corp: IBM ILOG CPLEX Optimizer (2018). https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer. Accessed 30 Mar 2018
2. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04244-7_14
3. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT Competition 2017. In: Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions, pp. 14–15 (2017)
4. Dubois-Lacoste, J., Hoos, H., Stützle, T.: On the empirical scaling behaviour of state-of-the-art local search algorithms for the Euclidean TSP. In: Proceedings of GECCO, pp. 377–384 (2015)
5. Falkner, S., Lindauer, M., Hutter, F.: `SpySMAC`: automated configuration and performance analysis of SAT solvers. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 215–222. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_16
6. Fawcett, C., Hoos, H.: Analysing differences between algorithm configurations through ablation. JOH **22**(4), 431–458 (2016)
7. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. EJOR **126**, 106–130 (2000)
8. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Morgan Kaufmann Publishers Inc., San Francisco (2005)
9. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40

10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Identifying key algorithm parameters and instance features using forward selection. In: Nicosia, G., Pardalos, P. (eds.) LION 2013. LNCS, vol. 7997, pp. 364–381. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-44973-4_40
11. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Proceedings of ICML, pp. 754–762 (2014)
12. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR **36**, 267–306 (2009)
13. Hutter, F., et al.: AClib: a benchmark library for algorithm configuration. In: Proceedings of LION, pp. 36–40 (2014)
14. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L., Stützle, T., Birattari, M.: The irace package: iterated racing for automatic algorithm configuration. ORP **3**, 43–58 (2016)
15. Mu, Z., Hoos, H.: Empirical scaling analyser: an automated system for empirical analysis of performance scaling. In: Proceedings of GECCO, pp. 771–772 (2015)
16. Mu, Z., Hoos, H.H., Stützle, T.: The impact of automated algorithm configuration on the scaling behaviour of state-of-the-art inexact TSP solvers. In: Festa, P., Sellmann, M., Vanschoren, J. (eds.) LION 2016. LNCS, vol. 10079, pp. 157–172. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50349-3_11
17. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS JOC **25**(2), 346–363 (2013)
18. Soos, M.: CryptoMiniSat v4. In: Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, p. 23 (2014)

A Model-Based Framework for Black-Box Problem Comparison Using Gaussian Processes

Sobia Saleem[1], Marcus Gallagher[1(✉)], and Ian Wood[2]

[1] School of Information Technology and Electrical Engineering, University of Queensland, Brisbane 4702, Australia
{s.saleem,marcusg}@uq.edu.au
[2] School of Mathematics and Physics, University of Queensland, Brisbane 4702, Australia
i.wood1@uq.edu.au

Abstract. An important challenge in black-box optimization is to be able to understand the relative performance of different algorithms on problem instances. This has motivated research in exploratory landscape analysis and algorithm selection, leading to a number of frameworks for analysis. However, these procedures often involve significant assumptions, or rely on information not typically available. In this paper we propose a new, model-based framework for the characterization of black-box optimization problems using Gaussian Process regression. The framework allows problem instances to be compared in a relatively simple way. The model-based approach also allows us to assess the goodness of fit and Gaussian Processes lead to an efficient means of model comparison. The implementation of the framework is described and validated on several test sets.

1 Introduction

A continuous black box optimization problem is defined as:

$$\min f(\mathbf{x}),\ \mathbf{x} \in S \subseteq I\!R^n \tag{1}$$

where $f()$ is the objective or fitness function and S is the feasible search space. It is assumed that the form of $f()$ is unknown but can be evaluated at any feasible candidate solution. Many real world problems can be formulated in this way and metaheuristic algorithms are specifically developed for this class of problems. The performance of an algorithm instance on a problem instance depends on how well the assumptions made by the algorithm suit the structure of the problem fitness landscape. Exploratory/fitness landscape analysis [10] aims to develop landscape features for understanding black-box problems, based on a sample of candidate solutions. A variety of different features have been proposed to measure different properties of problem landscapes [14]. If we are able to effectively characterize and compare problem instances, it should enable a

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 284–295, 2018.
https://doi.org/10.1007/978-3-319-99259-4_23

better understanding of algorithms and facilitate automated algorithm selection and configuration.

Evaluating the effectiveness of exploratory landscape analysis features and their role in the algorithm selection techniques is a nontrivial experimental challenge. While a number of procedures have been explored in the literature, they can be complex, involve the calculation of a set of landscape features and multiple stages of analysis that may require information not generally available for new problems.

In this paper we propose a model-based framework for continuous black-box problem comparison using Gaussian process (GP) regression. The advantages of this framework are that a flexible model is used, the accuracy of which can be measured, together with an appropriate way of comparing problems via their GP models. We describe our specific implementation of the model-based framework and evaluate the approach on a number of pre-designed test problem sets.

The paper is organized as follows: Sect. 2 summarizes existing frameworks for problem characterization and their limitations. Our proposed model-based framework for problem comparison is described in Sect. 3, highlighting its main elements. In Sect. 4 we describe the experiments, including the problem sets used based on controlled transformations. The experimental results are presented and discussed in Sect. 5 followed by the concluding Sect. 6.

2 Existing Frameworks for Problem Characterization and Algorithm Selection

An early framework for algorithm selection based on problem features was proposed by Rice [20]. This framework is based on extracting problem characteristics $c \in C$ for the given problem $f \in F$ and selecting an algorithm $\alpha \in A$ such that the output (e.g. performance) $y \in Y$ is maximized. The relationship between Rice's framework and more recent research in landscape analysis, meta-learning models and algorithm portfolios is discussed in [14].

Many different features have been proposed in the literature for problem characterization. But capturing and summarizing the structure of an arbitrary landscape is a difficult task [9]. Most features make very strong modelling assumptions (e.g. the R^2 coefficient for a linear or quadratic model of the landscape) or only use part of the information available in the sample (e.g. the sample skewness or kurtosis of the f values in the sample). Using a set of features in combination is a possibility [22], but the features are heterogeneous, making it difficult to select and utilize features in a principled way [16].

A regression model based on landscape features and algorithm (CMA-ES) hyperparameters is built in [13] to predict algorithm performance for a given problem. A framework to analyze the performance of algorithms using problem features is given in [15]. It uses a set of nine selected features and applies principal component analysis to reduce the feature space to two dimensions. An algorithm footprint is estimated on the feature space to relate which feature

Framework 1. Model-based continuous problem comparison

Given: set of problem instances, sample size N, regression model.
for all problems **do**
  Draw a sample of size N from the search space S.
  Evaluate f over the sample.
  Fit a regression model using the sample and f values.
  Evaluate the goodness of fit of the model.
end for
Calculate pairwise (dis)similarities between models.
Output: Problem similarity values.
Results Analysis: Dimensionality reduction or other techniques.

values correspond to particular algorithm performance. Another framework suggested in [11] uses algorithm rankings from the BBOB competition [5] to predict the best algorithm for a set of benchmark problems. The results are related to problem features to find some rules about the algorithm problem relationship. This approach requires a carefully chosen set of test functions and performance measures of the list of algorithms selected.

Most existing techniques that use problem features for algorithm selection or performance prediction are retrospective, using information not generally available for new problem instances, such as reported algorithm performance data or labelled categories of problems. Fundamentally, a way of comparing a set of problems in terms of their relative distances to each other might be simpler and yet still allow us to better understand the problem-algorithm mapping. In the next Section we describe a framework aimed at this, using GP regression models.

3 A Model-Based Framework for Problem Comparison

The framework proposed here essentially involves fitting a regression model to a sample of candidate solutions and fitness function values for a set of black-box optimization problem instances. A goodness of fit or error measure is then used to evaluate the regression model. Our framework compares problems by a comparison of the regression models built for each problem. This provides a set of pairwise distance or similarity values which characterize a problem set in terms of their relative distances or similarities. Algorithm selection could then follow based on the assumption that an algorithm that is effective for a given problem instance is likely to also be effective for nearby problem instances. Calculating features embeds a problem set in a somewhat arbitrary space, where selecting a suitable similarity measure may be difficult. Assuming the sample contains some information about the important landscape structure, a clear advantage of using a flexible regression model is that increasing the sample size will typically result in an improved model fit (i.e. a better representation of the problem landscape) whereas in the case of estimating problem features, increasing the sample size simply makes the feature estimate more robust.

3.1 Problem Comparison Using Gaussian Processes

In principle any regression model could be used in the above framework, however GPs are particularly well-suited to this task, as discussed below. Briefly, a GP is defined as a collection of random variables such that their joint distribution is a multivariate Gaussian, $\mathcal{N}(\mu, \Sigma)$ [19]. A GP is completely specified by its mean ($\mathbf{E}[f(\mathbf{x})]$) and covariance ($k(\mathbf{x}_i, \mathbf{x}_j)$) functions. This defines a prior distribution over the function space. Given a set of training data consisting of input vectors, $\mathbf{x}$ and target values $\mathbf{y}$, the posterior predictive distribution of the GP at a test point, $\mathbf{x}_*$, is Gaussian with mean and variance given by:

$$\bar{f}_* = \mathbf{k}_*^T (K + \sigma_n^2 I)^{-1} \mathbf{y} \tag{2}$$

$$\text{Var}(f_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T (K + \sigma_n^2 I)^{-1} \mathbf{k}_* \tag{3}$$

where K is the $(N \times N)$ covariance matrix between all pairs of points in the training set, $\mathbf{k}_*$ is a vector of covariance values between the test point and the training set, I is the identity matrix and σ_n^2 is an additive noise parameter (see below).

In general, calculating a distance or difference between regression models can be a complex task. However given two GPs, the distance calculation becomes a difference between two multivariate Gaussian distributions. The difference between two continuous probability density functions, $p(\mathbf{x})$ and $q(\mathbf{x})$, is commonly measured using the Kullback Leibler (KL) divergence:

$$d_{KL}(p||q) = \int_{-\infty}^{\infty} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} dx \tag{4}$$

If we have two multivariate Gaussian distributions, $\mathcal{N}_a(\mu_a, K_a)$ and $\mathcal{N}_b(\mu_b, K_b)$, then the KL divergence can be written in closed form:

$$d_{KL}(\mathcal{N}_a||\mathcal{N}_b) = \frac{1}{2} \log |K_b K_a^{-1}| + \frac{1}{2} K_b^{-1}((\mu_a - \mu_b)(\mu_a - \mu_b)^T + K_a - K_b) \tag{5}$$

We use the Jeffreys divergence [7], d_J which is a symmetric version of the KL divergence:

$$d_J(\mathcal{N}_a||\mathcal{N}_b) = \frac{1}{2}(d_{KL}(\mathcal{N}_a||\mathcal{N}_b) + d_{KL}(\mathcal{N}_b||\mathcal{N}_a)) \tag{6}$$

This gives us an efficient and direct way to calculate the difference between two problems via the GP models of their landscapes. Similar problems are expected to have relatively small divergence values and large divergence values will imply that the problems are quite different from each other.

3.2 Gaussian Process Implementation Details

Building a GP regression model requires the selection of mean and covariance functions. The mean function is assumed to be zero (the sample data can be

centred by subtracting the sample mean prior to fitting the model). There are many possible covariance functions that can be used to characterize the degree of similarity between data points in the input space [18]. The squared exponential covariance function is the most common choice [19]. The squared exponential covariance function has the form:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(\frac{1}{2l^2}||\mathbf{x}_i - \mathbf{x}_j||_2\right) + \sigma_n^2 \delta_{ij} \tag{7}$$

where δ_{ij} is the Kronecker delta function. The characteristic length scale l is indicative of the smoothness of the function. This hyperparameter captures the distance needed to move in any direction for the function values to become uncorrelated. σ_f^2 is the signal variance. In addition, the noise variance σ_n^2 is a hyperparameter of the GP which specifies the trade-off between the strength of the prior and fitting observed data [19]. In this paper we use the squared exponential covariance function as a spherical model across all dimensions of our problem sample data (i.e. a single l parameter is used). The optimization of these hyperparameters is important for fitting a GP. We use a standard approach to this: conjugate gradient is used to maximize the log-likelihood [19]. The hyperparameter optimization problem is not convex, so we use trial and error to set hyperparameters in the ranges indicated by the problem data. Many values worked well; for the results presented here the initialization used was $l = 0.5$, $\sigma_f^2 = 1000$ and $\sigma_n^2 = 0.1$.

In case of very smooth functions, the correlation between the observations is very high. This results in very similar rows/columns in the covariance matrix which can lead to making it poorly conditioned. This is an important issue in the implementation of GP's. The inversion of the covariance matrix is done using the Cholesky decomposition as it increases the tolerance towards the conditioning problem [17].

3.3 Related Work

Surrogate models have been widely used in optimization, particularly for problems where evaluating f is expensive. The model is used in place of the actual objective function. In using surrogate models it is important to balance the number of samples used (which should be minimum) with the improvement in approximation [3]. GPs as well as other models such as Random Forests and Support Vector Machines have been used [1]. Bayesian optimization algorithms use GPs to find a better solution using a minimum number of function evaluations [4,17]. A survey on the use of surrogate models in evolutionary computation is given in [6]. A framework based on using GP to find the promising individuals in the PSO population during the search is presented in [21].

As far as we are aware, there has only been one recent paper where surrogate regression models have been directly used to characterize problem instances [2]. The authors' main focus is on reducing the sample size required for feature-based algorithm selection. Therefore, a GP model is built from a small sample

as a surrogate, with further sampling carried out on the surrogate. In contrast, we calculate the difference between models fitted to these samples rather than computing features based on the surrogate model.

Finally, the Jeffrey's divergence has previously been used to compare problems in the context of length-scale feature analysis [12]. Length scale distributions were obtained using kernel density estimation and the divergence values calculated numerically over the sample. As shown above, the GP model-based framework requires no density estimation and the divergence values are calculated in closed-form.

4 Experimental Methodology

In this section we validate our framework on several test problems. We have generated 4 sets of 11 problem instances by gradually transforming a standard test problem into a different problem in a controlled way. The transformations determine a possible intuitive ordering of the problems, which we then try and recover in the black-box scenario, using the GP model-based framework. Existing problem sets such as the BBOB functions [5] do not have such an ordering, making it less straightforward to evaluate our results. The problem transformations are:

- Sphere to Rastrigin: the amplitude of the periodic term in the Rastrigin function is increased in 10 equal steps.
- Rastrigin to Flat: piece-wise linear combination, flat region grows from the center and expands equally in each intermediate problem.
- Sphere to Ellipse: increase in condition number across problem dimensions.
- Linear to Sphere: piece-wise linear combination, linear slope function is replaced by sphere function starting from the center.

Figure 1 illustrates the transformations in 1-D. We used 2-D and 5-D versions of the problems, with $S = [-5, 5]^n$. The sample size used was $N = 1000$ for the 2-D problems and $N = 2500$ for the 5-D problems. To visualize the results, dimensionality reduction techniques based on the similarity matrices can be utilized. We applied the t-Stochastic Neighbor Embedding (t-SNE) algorithm [8] as well as heatmaps and dendrograms to visualize the problem comparisons. t-SNE is a state of the art dimensionality reduction technique in machine learning. The algorithm is based on similar principles to our framework in that it calculates the KL divergence between a distribution of distances in a high dimensional data space and a lower dimensional mapping.

5 Results and Discussion

To measure the goodness of fit of the GP model, we have used the normalized mean square error (NMSE) for the test set. The NMSE calculates the deviation between the GP model predicted ($f_*(\mathbf{x}_i)$) and actual fitness values at sample points. It is defined as:

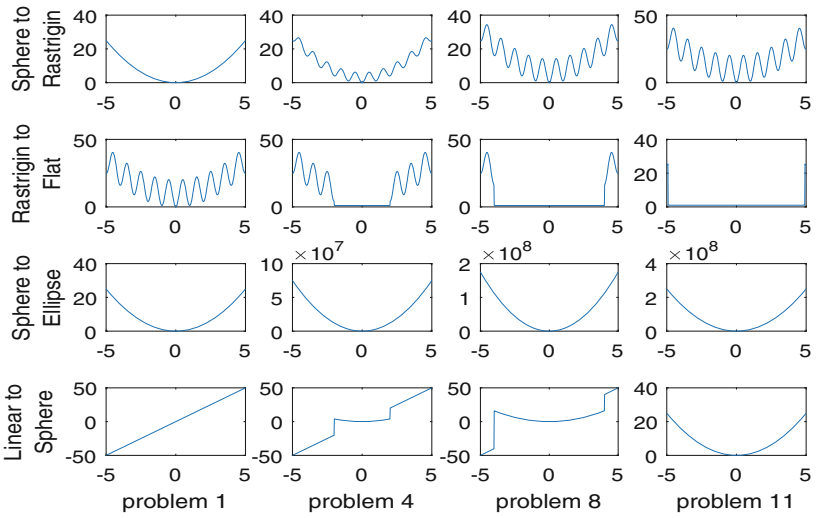


Fig. 1. Problem transformations used to generate the test problem sets. Shown are 1-D versions of problems 1,4,8 and 11 in each set.

$$NMSE = \frac{1}{N} \sum_{i} \frac{(f_*(\mathbf{x}_i) - f(\mathbf{x}_i))^2}{\overline{f_*()} \cdot \overline{f()}} \tag{8}$$

where $\overline{f()}$ is the sample mean of the function values. The NMSE values for the models of all the problems in each transformation in 2D are shown in Table 1. The NMSE values for 5D problem models are shown in Table 2.

Table 1. The estimated NMSE values in 2D.

Prob ID	Sphere2Ellipse	Rastrigin2Flat	Sphere2Rastrigin	Linear2Sphere
1	8.59E-06	0.001599	9.44E-06	3.66E-06
2	1.20E-08	0.0067396	0.00019719	3.72E-06
3	3.91E-09	0.014997	0.0002686	0.001875
4	9.51E-10	0.01169	0.00026819	0.0034078
5	5.38E-10	0.031776	0.00052105	0.0076807
6	7.59E-10	0.077764	0.00028147	0.014644
7	3.50E-10	0.085257	0.00071756	0.07977
8	3.17E-10	0.31847	0.00066255	0.092218
9	2.21E-10	4.1164	0.00019552	0.015076
10	2.36E-10	2.7546	0.00029032	0.11195
11	2.89E-10	0.002448	0.00036012	8.79E-06

Table 2. The estimated NMSE values in 5D.

Prob ID	Sphere2Ellipse	Rastrigin2Flat	Sphere2Rastrigin	Linear2Sphere
1	1.20E-06	0.030001	1.63E-06	2.97E-06
2	1.15E-08	0.030058	0.0013048	2.45E-06
3	4.76E-09	0.032689	0.004254	3.53E-06
4	3.05E-09	0.038068	0.0080946	0.0014831
5	9.84E-10	0.057157	0.011884	0.0264
6	8.69E-10	0.11271	0.01589	0.1312
7	4.85E-10	0.40765	0.019802	0.23115
8	2.03E-10	2.5472	0.023586	0.22068
9	1.56E-10	7.4724	0.027164	0.014305
10	3.71E-11	0.79306	0.030465	1.47E-06
11	1.70E-10	0.0024149	0.033622	1.54E-06

The t-SNE plots for the Sphere to Ellipse transformations in 2D and 5D are shown in Fig. 2. The results show that the order of the problems in the transformation is very strongly recovered from our framework (i.e. problem i tends to be closest to $i-1$ and/or $i+1$). The model error values for all the problems in this set are very small, indicating that the GP is an accurate model of these landscapes. The dimensionality reduction attempts to preserve the pairwise distances between the problem instances, so the orientation of the points in the plot is not meaningful.

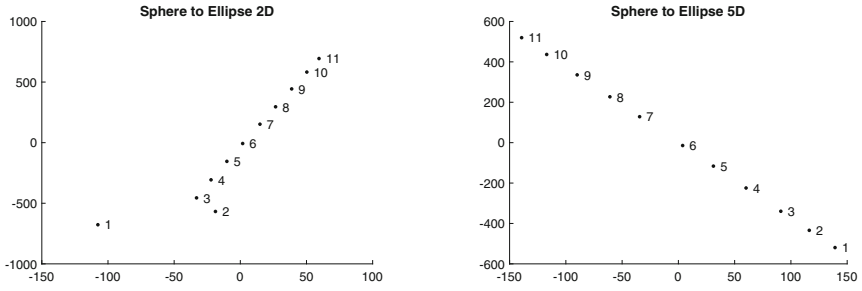


Fig. 2. t-SNE visualisations of the Sphere to Ellipse problem set. Left: 2D, Right: 5D

Figure 3 shows the t-SNE plot for the Rastrigin to Flat problem set in 2D and 5D. The visualisation shows that the problem order from the transformation is strongly recovered. This problem transformation is rather complex as it combines multiple local minima and a perfectly flat surface. The error values for the models (Tables 1 and 2) show relatively high values for problems 8,9 and 10, suggesting that the model lacks some accuracy at modelling the function. This

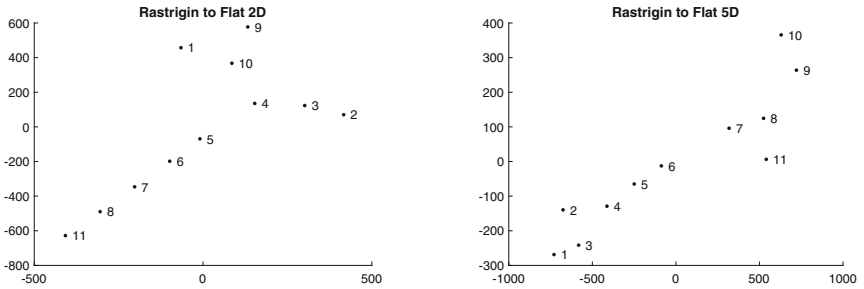


Fig. 3. t-SNE visualisations of the Rastrigin to Flat problem set. Left: 2D, Right: 5D

may partly explain the distribution of problems in Fig. 3 (e.g. problems 9 and 10 in the 2D set are well separated from 8 and 11). No model can be expected to provide a highly accurate model of arbitrary fitness landscapes based on a modest sample of data. A strength of our approach is that rather than simply accepting calculated feature values, the error of the model gives us a measure of how reliable our results are. For the Sphere to Rastrigin transformations, Fig. 4(a) shows excellent recovery of the problem ordering in 2D. The Sphere function (problem 1) appears somewhat separate, perhaps because it is the only unimodal problem in the set. In Fig. 4(b) the trend is not as visually obvious across the entire problem set, however most problem instances have, as their nearest neighbours, the neighbouring problems in the transformation (e.g. 8 is closest to 9 and 10, 7 is closest to 6 and 5). The model error values for this problem transformation, (Tables 1 and 2), are all relatively low indicating a good fit to the data. Error values increase a little from 2D to 5D, with these sample sizes. Note that a model can still be capturing some important landscape properties and have a large error value.

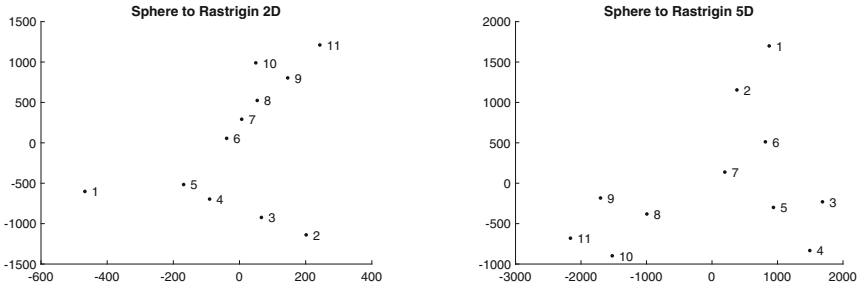


Fig. 4. t-SNE visualisations of the Sphere to Rastrigin set. Left: 2D, Right: 5D.

The Linear to Sphere transformation is done in a piece-wise way which may contribute to high NMSE values in the middle functions of the transformation.

Function 1 is the Linear function and Function 11 is the Sphere function and both have a very good model fit. Both of these functions are smoothly structured functions and we can see in the Fig. 5 that both are rather close to each other. The remaining problems also form a cluster which show their similarity.

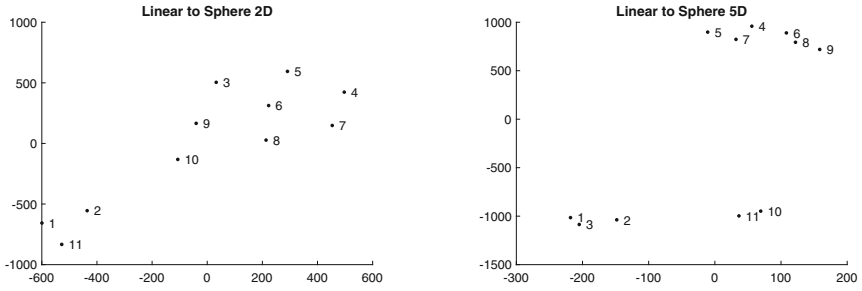


Fig. 5. t-SNE visualisations of the Linear to Sphere problem set. Left: 2D, Right: 5D

A more direct way of examining the results is to look at a pairwise distance matrix. Figure 6 shows two examples for the 5D Sphere to Ellipse and 5D Linear to Sphere problem sets visualized as heat maps. The pattern across the Sphere to Ellipse transformation reflects almost perfectly the definition of the transformation: nearby problems are close to the diagonal and have a low distance value, which smoothly increases as we move away from the diagonal. For the Linear to Sphere transformation we can see that the model distances between problems 1–3 and 5–7 are greater than expected, as are the distances between 5–8 and 10–11. This agrees with the t-SNE plot for this problem set (Fig. 5, right). Finally, dendrograms offer another popular way of displaying distance data. Figure 7 shows examples for the 2D Linear to Sphere and 5D Sphere to Rastrigin problem sets. For some problems (Figs. 5, left and 4, right), the relative magnitude of the distances between problems is more accurately represented in dendrograms as compared to the t-SNE plots for these problem sets.

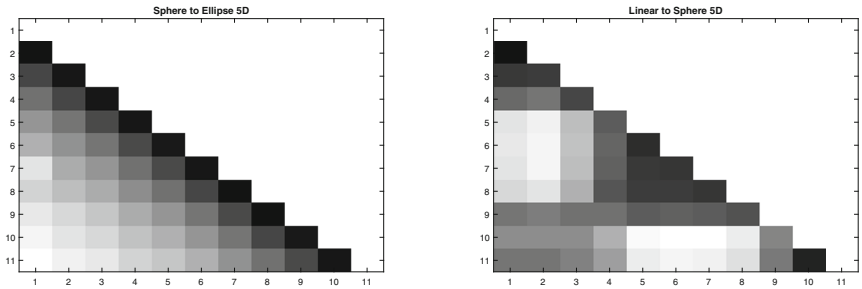


Fig. 6. Heat maps for two of the problem sets. Left: 5D Sphere to Ellipse, Right: 5D Linear to Sphere. Greyscale shows the log of the distance values for better contrast.

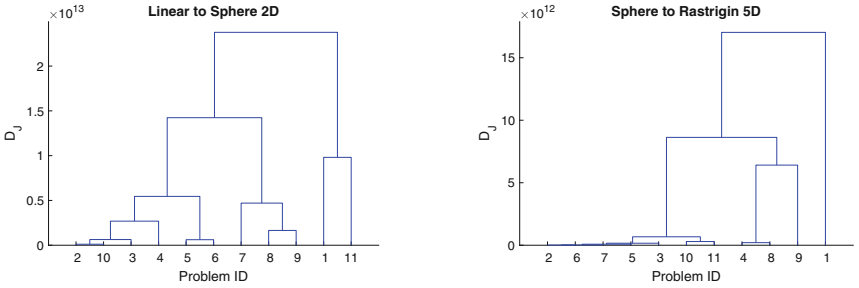


Fig. 7. Dendrograms for two of the problem sets. The relative distances between problems is given by the y-axis. (Left) 2D Linear to Sphere transformation, (Right) 5D Sphere to Rastrigin transformation.

6 Conclusion

Exploratory landscape analysis and algorithm selection frameworks have motivated the research in understanding problems. Here, we have proposed a model based framework for understanding problems using Gaussian Processes. Being a regression model, we have a measure of goodness of fit which provides a source of verification of model. To get the distance between models of problems, GP provide a closed form expression for measuring KL divergence which avoids any numerical approximations. We tested the methodology on a set of problem transformations with pre-defined similarity ranking. The framework is tested on its ability to identify the distance between problems in each transformation. GPs are known to provide a surrogate model of the function using a small set of samples. In future we will extend our methodology on problems with smaller sample sizes. The experiments in this paper are limited to 2D and 5D problems, but higher dimensional problems also need to be explored. The experimental results presented here indicate that measures based on these models can detect small differences between problems and recover much of an specified problem ordering.

References

1. Bajer, L., Pitra, Z., Holeňa, M.: Benchmarking Gaussian processes and random forests surrogate models on the BBOB noiseless testbed. In: Proceedings of the Companion Publication of the 2015 Conference on Genetic and Evolutionary Computation, pp. 1143–1150. ACM (2015)
2. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Surrogate assisted feature computation for continuous problems. In: Festa, P., Sellmann, M., Vanschoren, J. (eds.) LION 2016. LNCS, vol. 10079, pp. 17–31. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50349-3_2
3. Forrester, A., Keane, A., et al.: Engineering Design via Surrogate Modelling: A Practical Guide. Wiley, Hoboken (2008)
4. Frean, M., Boyle, P.: Using Gaussian processes to optimize expensive functions. In: Wobcke, W., Zhang, M. (eds.) AI 2008. LNCS (LNAI), vol. 5360, pp. 258–267. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89378-3_25

5. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking: noiseless functions definitions. Technical report (2009)
6. Jin, Y.: Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evol. Comput. **1**(2), 61–70 (2011)
7. Legrand, L., Grivel, E., Giremus, A.: Jeffrey's divergence between autoregressive moving-average processes. In: 2017 25th European Signal Processing Conference (EUSIPCO), pp. 1085–1089. IEEE (2017)
8. Maaten, L.V.D., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**(Nov), 2579–2605 (2008)
9. Malan, K.M., Engelbrecht, A.P.: Fitness landscape analysis for metaheuristic performance prediction. In: Richter, H., Engelbrecht, A. (eds.) Recent Advances in the Theory and Application of Fitness Landscapes. ECC, vol. 6, pp. 103–132. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-41888-4_4
10. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, pp. 829–836. ACM (2011)
11. Mersmann, O., Preuss, M., Trautmann, H., Bischl, B., Weihs, C.: Analyzing the bbob results by means of benchmarking concepts. Evol. Comput. **23**(1), 161–185 (2015)
12. Morgan, R., Gallagher, M.: Analysing and characterising optimization problems using length scale. Soft Comput. **21**(7), 1735–1752 (2017)
13. Muñoz, M.A., Kirley, M., Halgamuge, S.K.: A meta-learning prediction model of algorithm performance for continuous optimization problems. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012. LNCS, vol. 7491, pp. 226–235. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32937-1_23
14. Munoz, M.A., Kirley, M., Halgamuge, S.K.: The algorithm selection problem on the continuous optimization domain. In: Moewes, C., Nürnberger, A. (eds.) Computational Intelligence in Intelligent Data Analysis, pp. 75–89. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-32378-2_6
15. Muñoz, M.A., Smith-Miles, K.A.: Performance analysis of continuous black-box optimization algorithms via footprints in instance space. Evol. Comput. **25**(4), 529–554 (2017)
16. Muñoz, M.A., Sun, Y., Kirley, M., Halgamuge, S.K.: Algorithm selection for black-box continuous optimization problems: a survey on methods and challenges. Inf. Sci. **317**, 224–245 (2015)
17. Osborne, M.A., Garnett, R., Roberts, S.J.: Gaussian processes for global optimization. In: 3rd International Conference on Learning and Intelligent Optimization (LION3), pp. 1–15. Citeseer (2009)
18. Rasmussen, C.E., Nickisch, H.: Gaussian processes for machine learning (GPML) toolbox. J. Mach. Learn. Res. **11**(Nov), 3011–3015 (2010)
19. Rasmussen, C.E., Williams, C.K.: Gaussian Process for Machine Learning. MIT press, Cambridge (2006)
20. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
21. Su, G.: Accelerating particle swarm optimization algorithms using Gaussian process machine learning. In: 2009 International Conference on Computational Intelligence and Natural Computing, CINC 2009, vol. 2, pp. 174–177. IEEE (2009)
22. Sun, Y., Halgamuge, S.K., Kirley, M., Munoz, M.A.: On the selection of fitness landscape analysis metrics for continuous optimization problems. In: 2014 7th International Conference on Information and Automation for Sustainability (ICIAfS), pp. 1–6. IEEE (2014)

A Suite of Computationally Expensive Shape Optimisation Problems Using Computational Fluid Dynamics

Steven J. Daniels(✉), Alma A. M. Rahat(✉), Richard M. Everson, Gavin R. Tabor, and Jonathan E. Fieldsend

University of Exeter, Exeter, UK
{S.Daniels,A.A.M.Rahat,R.M.Everson,G.R.Tabor,J.E.Fieldsend}@exeter.ac.uk

Abstract. In many product design and development applications, Computational Fluid Dynamics (CFD) has become a useful tool for analysis. This is particularly because of the accuracy of CFD simulations in predicting the important flow attributes for a given design. On occasions when design optimisation is applied to real-world engineering problems using CFD, the implementation may not be available for examination. As such, in both the CFD and optimisation communities, there is a need for a set of computationally expensive benchmark test problems for design optimisation using CFD. In this paper, we present a suite of *three* computationally expensive real-world problems observed in different fields of engineering. We have developed Python software capable of automatically constructing geometries from a given decision vector, running appropriate simulations using the CFD code OpenFOAM, and returning the computed objective values. Thus, users may easily evaluate a decision vector and perform optimisation of these design problems using their optimisation methods without developing custom CFD code. For comparison, we provide the objective values for the base geometries and typical computation times for the test cases presented here.

1 Introduction

Many real-world engineering design optimisation problems are computationally expensive. For instance, optimising the shape of an aircraft wing may require evaluating the performance of candidate designs in flight using Computational Fluid Dynamics (CFD). A high-fidelity simulation may take hours to converge, imposing a practical limit to how many designs may be considered during optimisation.

In recent years, interest in computationally expensive optimisation problems has grown rapidly. It was first popularised by Jones *et al.* [1]. They presented two real-world problems: minimising a voltage spike in an integrated circuit, and exploring the trade-off between viscosity and yield stress in a proprietary automotive application; both of which required expensive computer simulations. Since then many example problems have been published. For instance, Naujoks

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 296–307, 2018.
https://doi.org/10.1007/978-3-319-99259-4_24

et al. presented a multi-objective shape optimisation problem for aerofoils, optimising high-lift and low-drag simultaneously using CFD simulations [2]. Similar problems with a particular attention to drag coefficients and uncertainty are available from [3,4]. Leary *et al.* presented a CFD-based shape optimisation problem of minimising the volume of beams subject to stress and stiffness constraints [5]. In [6], a multi-objective optimisation problem of minimising pressure drop and maximising heat flux for the design of a heat exchanger using CFD was discussed. Another example is the rocket simulator in [7], which has been used for simultaneously optimising the time to return to earth and the angular distance travelled [8]. More recently, Daniels *et al.* presented several CFD based geometry optimisation problems: minimising pressure difference in a pipe [9] and a duct [10]. Beyond engineering design problems, further examples of computationally expensive problems exist in the literature from the machine learning community, mostly for optimising model parameters to reduce training errors; see for examples [11–13].

Note that authors often develop custom codes for their problems, and they are generally reluctant to release code, primarily because many problems are proprietary in nature. Therefore, despite numerous example problems, it is often difficult to acquire the simulators for exact comparison of methods. Moreover, there is no complete test suite of benchmark problems similar to inexpensive test suites such as the DTLZ test problems [14]. The optimisation community is actively investigating benchmark computationally expensive problems. However, most previous attempts, for example [15,16], use *pseudo-expensive* problems, i.e. inexpensive functions are used with delays to mimic expensive problems.

Addressing these issues, the aim of this paper is to present a test problem suite[1] for computationally expensive problems with the following features:

- We focus on real-world problems of designing apparatus using CFD to evaluate the performances of a geometry in a fluid environment. As such these are functions that are *truly* computationally expensive to evaluate, and optimisation has real implications for engineers.
- All problems use open source software suitable for popular platforms and machines, and thus enable comparison between different methods without requiring substantial hardware.
- This flexible suite offers the opportunity to create different instances of a problem with a configurable number of dimensions for the decision space. In practice increasing the number of dimensions increases the difficulty of the associated problem.
- We provide the base geometry performance and the computation time so that they may be used as a yardstick for comparison.
- Some decision vectors may result in an unphysical geometry, and consequently a CFD simulation will fail. We therefore constrained the problems to only evaluate feasible geometries. We provide a callable function encapsulating the

[1] Python code for these test problems and relevant instructions are available at: https://bitbucket.org/arahat/cfd-test-problem-suite.

constraint checks to inform the users whether a decision vector is feasible. As such users may treat these constraints as black-box functions.
– Some design optimisation scenarios are naturally phrased as single objective problems, while others are inherently multi-objective. Adhering to such genuine objectivity of design optimisation, we present *three* distinct design problems: *two* single-objective and *one* multi-objective.

The rest of the paper is organised as follows. Section 2 provides a background discussion on geometry optimisation using CFD, and in Sect. 3 we present the necessary background regarding geometry representation. The problems in this test suite are detailed in Sects. 4 and 5. Finally, we draw conclusions in Sect. 6 with the base geometry performance and relevant computation time.

2 Computational Fluid Dynamics (CFD)

Design performance in a fluid environment cannot usually be evaluated analytically. It is therefore necessary to resort to a numerical approximation using CFD. CFD undoubtedly represents the more computationally costly end of engineering simulation, requiring fast processing speed and making serious demands on memory, multi-processor intercommunication speeds (for parallelisation) and even graphical visualisation. CFD requires the solution of a set of Partial Differential Equations (PDEs) which describe the physics of fluid flow (principally the Navier-Stokes equations, obtained independently by M. Navier and G. Stokes in 1822). This is typically achieved using the Finite Volume Method, in which the fluid continuum is discretised into a grid and the PDEs are solved algebraically for each cell. There are many software packages available to perform these calculations. Over the last few years the open-source C++ code OpenFOAM [17] has emerged as one of the most popular CFD codes in the community, partly boosted by the financial issues of using a commercial code with a 'per core' license cost.

For this test suite, we have developed a Python-based optimisation framework to operate with OpenFOAM. The communication of the Python libraries with OpenFOAM was achieved using PyFoam as an interface to control the OpenFOAM case set-ups and runs, and to post-process the data generated after each CFD simulation. An appealing aspect of this framework is that the simulations are run in an automated procedure based on a decision vector prescribed by the user. The parametric geometries generated from the decision vector are converted into sterolithography (STL) files and imported into OpenFOAM. Interested readers should refer to [18] for details of importing a STL file into OpenFOAM environment. Following CFD simulation with the imported STL file, the problem specific objective value(s) are computed from the flow fields. In the next section, we describe the geometry representation methods used in this paper.

3 Geometry Representation Methods

Generally, the standard procedure to create a geometry is to use a Computer Aided Design (CAD) software. However, it is difficult to automatically alter designs using CAD. Consequently, we resort to various parametric

representations for parts of the original geometry created in CAD; varying the parameters of the representation allows the generation of new geometries. The new geometries may then be considered as candidate designs in the optimisation. Below we briefly describe the representation methods used in this paper. To keep computation times manageable we formulate the test problems in terms of two-dimensional geometries (although one is a fully three-dimensional flow).

3.1 Catmull-Clark Subdivision Curves

To alter the boundary wall of a geometry, we use Catmull-Clark subdivision curves [19]. In this method, a curve $\mathcal{C}$ is parametrised with a sequence of n vertices $S^0 = \langle \mathbf{p}_1, \ldots, \mathbf{p}_n \rangle$. We refer to each vertex vector $\mathbf{p}_i \in \mathbb{R}^2$ in the control polygon S^0 as a control point. We then insert a mid-point between adjacent vertices, and adjust each of the vertices' position iteratively. Thus, at the jth iteration with previous vertex sequence S^{j-1}, we generate a larger sequence S^j. The vectorial subdivision operation in each iteration j may be expressed as:

$$S^j[1] = S^{j-1}[1], \tag{1}$$

$$S^j[|S^j|] = S^{j-1}[|S^{j-1}|], \tag{2}$$

$$S^j[2i] = \frac{S^{j-1}[i-1] + 6S^{j-1}[i] + S^{j-1}[i+1]}{8}, \tag{3}$$

$$S^j[2i+1] = \frac{4S^{j-1}[i-1] + 4S^{j-1}[i+1]}{8}, \tag{4}$$

where $|S^j| = 2|S^{j-1}| - 1$ is the total number of elements in the sequence S^j, $S^j[k]$ is the kth element in the sequence, and indices $i \in \{2, \ldots, |S^{j-1}| - 1\}$.

The curve $\mathcal{C} = \lim_{j\to\infty} S^j$ is a result of the infinite iterative process of subdivision starting from the original sequence of vertices S^0. Thus a sequence S^0 with a small number of control points is sufficient to represent a curve with infinitely many points. We construct a decision vector from S^0 by sequentially arranging the control points in a vector.

It should be noted that from practical perspective only a few iterations of subdivision usually results in a visually smooth curve that may be exported in STL format. We set the iteration limit to *five* in this paper. In Fig. 1, we provide an illustration of a Catmull-Clark subdivision curve.

3.2 Chebyshev Polynomials

Often multiple geometric variables may be spatially related, but the nature of this relationship may not be known *a priori*. Rather than representing these independently, it may be useful to encode their relationship with a parametrised function such that altering the parameters changes all geometric variables simultaneously. An additional benefit is that a small number of parameters may then represent a large number of variables, and consequently reduce the search space size. In this paper, we used Chebyshev polynomials for encoding spatial relationships for one dimensional variables [20].

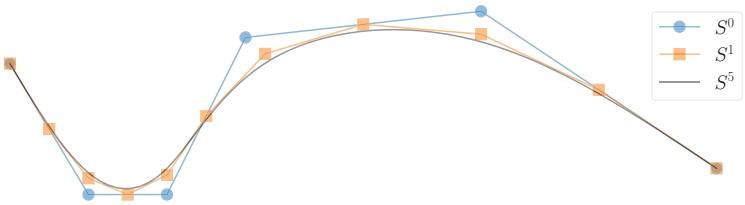


Fig. 1. Illustration of a Catmull-Clark subdivision curve. The blue line shows the original control polygon S^0 with the control points depicted in blue dots. After one iteration, the resulting approximation S^1 is shown in orange with the new control points shown in solid squares. The visually smooth curve S^5 after five iterations is drawn in black. From practical perspective, further iterations are unnecessary as the curve is already smooth enough for STL file generation. (Color figure online)

A function based on Chebyshev polynomials (type I) may be defined as:

$$f(t, \mathbf{c}) = \sum_{i=0}^{n} c_i T_i(t), \tag{5}$$

where, $t \in [0, 1]$ is a location variable, $T_i(t) = \frac{(-2)^i i!}{(2i)!}\sqrt{1-t^2}\,\frac{d^i}{dt^i}(1-t^2)^{\frac{i-1}{2}}$ is the ith Chebyshev basis function, which is orthogonal to all other Chebyshev functions, and the associated coefficient vector is $\mathbf{c} = (c_1, \ldots, c_n)^\top$ with $c_i \in [-1, 1]$. With this parameterised function, if there are k variables at locations $t_1, \ldots, t_k$, and a vector of n coefficients (or parameters) $\mathbf{c}$, then the jth variable of interest takes the value:

$$v_j = f(t_j, \mathbf{c}). \tag{6}$$

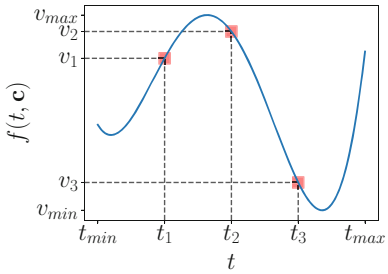
The coefficient vector may be directly considered as the decision vector here, and as it is varied, we may achieve a distinct value for v_j at a fixed location t_j.

Note that it is straightforward to scale the variables v_j and locations t_j between specified lower and upper bounds. Furthermore, so long as $n < k$, a smaller number of coefficients (or parameters) in this representation may directly encapsulate and control the relationships between the k variables $v_1, \ldots, v_k$. Thus we may effectively reduce the search space. An illustration of the scheme is presented in Fig. 2a.

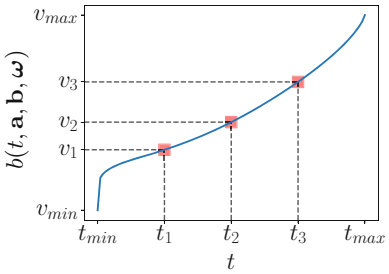
3.3 Monotonic Beta Cumulative Distribution Functions

It can be envisaged that some geometric variables should be monotonically increasing with respect to location. Again, we may use parametric monotonic functions to encode such relationships. In this paper, we use a weighted sum of cumulative distribution functions (CDFs) of Beta distributions for this purpose.

Let the function $F(t, \alpha, \beta)$ be the CDF of a Beta distribution. The CDF monotonically increases from zero to one as location t is changed from zero to one for a specified set of shape parameters $\alpha > 0$ and $\beta > 0$. If the shape parameters



(a) Chebyshev polynomials from (5).



(b) Monotonic Beta CDF from (7).

Fig. 2. Illustration of parametric functions: Chebyshev polynomials in (a) and monotonic Beta CDFs in (b). Function responses are depicted in blue. Red squares show the selected function values v_1, v_2 and v_3 at locations t_1, t_2 and t_3. For demonstration we chose arbitrary parameter vectors $\mathbf{c}, \mathbf{a}, \mathbf{b}$ and $\boldsymbol{\omega}$. Clearly, changing the parameters will result in a different function response. Thus, a variable of interest v_j at a fixed location t_j may be varied by changing the relevant parameters, but the basis function representation ensures that changes to a single coefficient yields correlated changes to all the variables. (Color figure online)

are altered to α' and β', this will result in a distinct monotonic relationship between t and $F(t, \alpha', \beta')$ in comparison to t and $F(t, \alpha, \beta)$. To further increase the flexibility of such a representation, we may consider the weighted sum of multiple Beta CDFs, and as a convex combination of monotonic functions this will preserve the monotonicity. Such a combination of n Beta CDFs may be expressed as:

$$b(t, \mathbf{a}, \mathbf{b}, \boldsymbol{\omega}) = \sum_{i=1}^{n} \omega_i F(t, \alpha_i, \beta_i), \tag{7}$$

where $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n)^\top$ is the weight vector with $\omega_i > 0$ for all i, $\sum_i \omega_i = 1$, and $\mathbf{a} = (\alpha_1, \ldots, \alpha_n)^\top$ and $\mathbf{b} = (\beta_1, \ldots, \beta_n)^\top$ are the vectors of parameters for the Beta distribution. As in Sect. 3.2, we can now compute a monotonic variable of interest using Eq. (6) by replacing $f(t_j, \mathbf{c})$ with $b(t, \mathbf{a}, \mathbf{b}, \boldsymbol{\omega})$. Again, if we choose a small n number of Beta functions to represent a large k number of monotonic geometric variables, then we efficiently reduce the size of the search space. The scheme is depicted in Fig. 2b.

4 Single Objective Problems

In this paper, we present *two* single objective problems: PitzDaily and Kaplan draft tube. These are detailed in the next sections.

4.1 PitzDaily

Flow separation, recirculation, and reattachment are common phenomena observed in many engineering applications, and are usually undesirable features

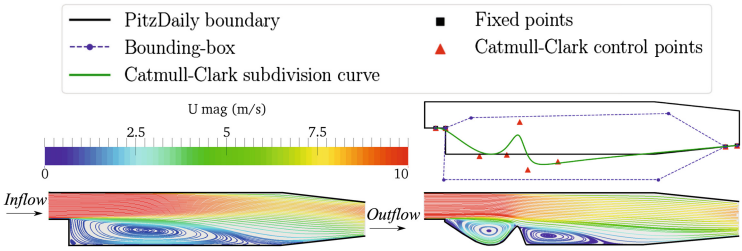


Fig. 3. (Left) streamlines of the flowfield coloured by velocity magnitude for the original (base) PitzDaily case. (Top-right) a schematic of the Catmull-Clark subdivision curve setup for the PitzDaily test case, and a randomly generated subdivision curve. (Bottom-right) streamlines and contour of the resulting flowfield from the random design.

within a product's design. Based on the experimental set up by Pitz and Daily [21], this first case features a so-called 'backward-facing step', which serves as a simple prototype for simulating the above flow phenomena. In this geometry, the flow separates at the edge of the step, creating a recirculation zone, the flow then reattaches at some distance beyond the step. The flow structure for the base case can be seen in Fig. 3 (bottom-left). Traditionally, this case has featured as a benchmark case for testing the accuracy of CFD methodologies and thus has been the focus of much experimental and computational investigation. Furthermore, this has also been used as a test case for adjoint (gradient descent) methods of optimisation (see for example [22]).

Head losses within a flow are an undesirable characteristic for engineering design. To quantify this, the mechanical energy loss factor, ζ, describes the energy that is converted to a form that cannot be used during the operation of an energy producing, consuming, or conducting system (i.e. due to frictional losses, or dissipation due to turbulence). In one mathematical form, ζ is defined as the total pressure difference between the inflow and outflow of the apparatus (relative to the kinetic energy at the inflow), i.e.

$$\zeta = \frac{1}{\frac{1}{2}\rho U_{in}^2}\left[\frac{1}{A_{in}}\int_{in} P_{t,in}(\mathbf{u}\cdot\mathbf{n})dA_{in} - \frac{1}{A_{out}}\int_{out} P_{t,out}(\mathbf{u}\cdot\mathbf{n})dA_{out}\right], \tag{8}$$

where P_t is the total pressure, and $\mathbf{u}\cdot\mathbf{n}$ indicates the velocity component normal to the boundary, U_{in} is the inflow velocity, ρ is the density of the fluid, A is the cross-sectional area, and subscripts *in* and *out* indicate the inflow and outflow boundaries. The primary objective of this case is to minimise this energy loss, i.e. $\min \zeta$. Using adjoint (gradient descent) optimisation, [22] identified a local minimum, with $\zeta = 0.0903$[2]; this was achieved by removing the backward-facing step, and by gently increasing and decreasing the cross-sectional area along the domain. Based on this, the design optimisation for this first case focuses on altering the geometry across the lower portion of the domain.

[2] Solution repeated using our framework.

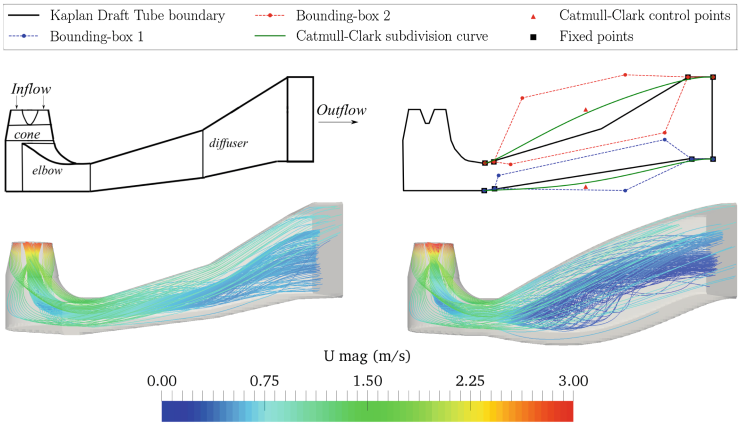


Fig. 4. Top-left: a schematic of the Hölleforsen-Kaplan sharp-heeled draft tube. Bottom-left: streamlines of the flow through the base (original) design. Top-right: a schematic of the Catmull-Clark subdivision curve setup for the Kaplan Draft tube, and a randomly generated subdivision curve. Bottom-right: streamlines of the resulting flowfield from the new (random) design.

Figure 3 (top-right) shows the case setup for optimisation. The fixed bounding box indicates the limits which the Catmull-Clark control points cannot exceed. Fixed points are applied to enforce a smooth transition between the Catmull-Clark subdivision curve and the adjacent wall. The bottom-right panel of Fig. 3 also shows an example geometry created from a randomly generated set of Catmull-Clark control points. Note that the number of control points can be defined by the user. For this configuration, the energy loss $\zeta = 0.2037$, which is an improvement on the cost function for the base design (see Table 1).

4.2 Sharp-Heeled Kaplan Draft Tube

A hydropower plant converts the gravitational potential energy of water from an upstream reservoir into electrical energy, by means of a turbine coupled to a generator. The flow leaving the turbine loses its velocity in the draft (exhaust) tube, where kinetic energy of the flow is transformed into pressure. This energy conversion has a significant impact on the efficiency and power of the turbines, thus, the draft tube design is of great interest to the industry. The two most common draft tubes considered in the literature are the *sharp-heeled* and *underground* designs; the former represents a large group that were installed in Swedish hydropower plants in the 1950s. Elbow-draft tubes are widely used for vertical Kaplan and Francis turbines, due to their low height, lesser excavation cost, and greater potential for pressure recovery. This type of draft tube consists of three parts: a cylindrical cone, an elbow, and an end diffuser. The draft tube geometry considered for the second case of this test suite is a 1:11 scaled model from

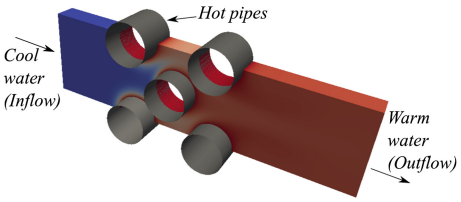


Fig. 5. CFD case set up for the staggered 'quincunx' formation of the cross-flow tube-bundle heat exchanger.

the Hölleforsen Kaplan turbine, built in 1949. A schematic of this draft tube geometry is shown in Fig. 4 (top-left).

The flowfield through the base design of the draft tube is shown in Fig. 4 (bottom-left). An unusual characteristic of this test case is that a swirl-flow is imposed at the inflow, which simulates the exit flow from the turbine. This makes the flowfield complex, and, unlike the other test cases in this suite, inherently three-dimensional. As discussed earlier, the main purpose of the draft tube is to recover the kinetic energy leaving the turbine by increasing the pressure energy. A performance indicator of this is given by the pressure recovery factor,

$$C_p = \frac{1}{\frac{1}{2}\rho U_{in}^2}\left[\frac{1}{A_{out}}\int_{A_{out}} p_{out} dA_{out} - \frac{1}{A_{in}}\int_{A_{in}} p_{in} dA_{in}\right], \tag{9}$$

where p is the static pressure. A higher value of C_p indicates a higher conversion of kinetic energy to pressure energy. Thus, the single objective of this problem is to maximise the pressure recovery factor, i.e. $\max C_p$. The region of interest for this case is the end-diffuser; changing its shape dramatically alters the structure of the swirl-flow, and the resulting kinetic-pressure energy conversion. Two Catmull-Clark subdivision curves define the shape of the top and bottom of this section, as indicated in Fig. 4 (top-right). Once again, a randomly generated subdivsion curve using one free control point is used as a demonstration for altering the diffuser shape. As this is a three-dimensional flow, this test case requires the highest computational effort out of all the cases in this test suite. Thus, it is likely that the user will run this case in parallel, something for which the OpenFOAM code is already equipped. The method of parallel computing used by OpenFOAM is known as 'domain decomposition', in which the geometry and associated fields are divided into sections allocated to separate cores. For the example case in Fig. 4, the pressure recovery factor $C_p = 0.955$, which is an improvement on the cost function for the base design (see Table 1).

5 Multi-objective Problem: Heat Exchanger

A cross-flow tube-bundle heat exchanger has a wide range of applications in many fields, such as the chemical, food and nuclear industries, and HVAC (Heating, Ventilation and Air Conditioning) sectors, to name a few. Generally, this type of

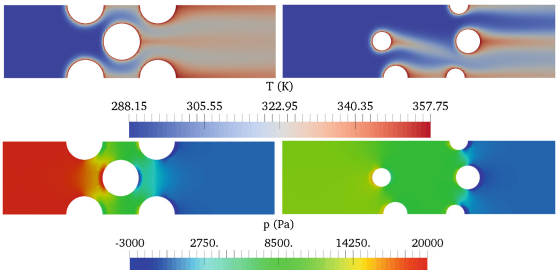


Fig. 6. (Left) tube arrangement of the base case for the heat exchanger. (Right) tube arrangement of a randomly generated decision vector. (Top row) contours of the temperature distribution. (Bottom row) contours of static pressure across the CFD domain.

heat exchanger contains many rows of tubes oriented in a direction perpendicular to the flow, as shown in Fig. 5. The tubes may be arranged in many configurations in order to obtain the greatest heat transfer between the two media. The transfer of heat between the tubes and the main flow occurs through the tube walls and will be maximised by increasing the surface area of contact between the flow and the walls. A detrimental effect of this may be that the static pressure across the tube configuration increases, requiring greater energy to push the flow through the heat exchanger. Overall, this potentially results in a conflicting pair of objectives for heat exchanger design.

For this final test case, we have constructed a simple heat exchanger with *three* rows of tubes. The design variables include altering the diameter of the tubes, the position of the tubes in the streamwise direction, and the number of tubes per row. To alter these variables, the decision vector contains the parameters of Chebyshev polynomials (for number of tubes in a row, and radii of the tubes), and monotonic Beta functions (for the position of the tubes) as described in Sects. 3.2 and 3.3. The cost functions describing the heat transfer and pressure drop across the heat exchanger are defined as follows:

$$\max \ |\Delta T| = |T_{in} - T_{out}|, \tag{10}$$

$$\min \ |\Delta p| = |p_{in} - p_{out}|, \tag{11}$$

where p is the static pressure (units in Pascal, Pa), and T is the temperature (units in Kelvin, K). Figure 6 (right) shows the result of a random decision vector on the configuration of the pipes and the flowfield. The cost functions from this random configuration are $|\Delta T| = 26.8733$K and $|\Delta p| = 12035.9$Pa; this shows that this configuration has improved on the pressure drop across the heat exchanger but the heat transfer has worsened when compared to the staggered 'quincunx' tube formation of the base case (see Table 1).

6 Conclusion

In this work we describe a Python-based software framework for the automated optimisation of a suite of computationally expensive design problems. These

include two single objective, and one multi-objective cases to act as benchmark test problems for the CFD and optimisation communities. An appealing aspect of this code is its flexibility, allowing the user to test their optimisation methodology under a number of dimensions for the decision space, and the application of these methods to real-world engineering problems. The schematics and contour diagrams shown in this paper were generated using utilities in the proposed framework, and ParaView – a visualisation utility – was used to visualise the flowfield data. In summary, Table 1 below shows the cost function values and typical simulation runtimes for the base case of each test problem, which may be used as a yardstick to compare the fitness of optimal designs found using different optimisation algorithms.

Table 1. Base geometry evaluation results for each test case. The calculations were performed using an Intel Xeon(R)-3.60 GHz desktop.

Test Problem	Execution time (sec)	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$
PitzDaily	40.35	0.312	(-)
Kaplan Draft Tube[a]	947.37	0.939	(-)
Heat Exchanger	34.55	40.933K	19,577Pa

[a]Simulation was performed in parallel using four-cores.

Acknowledgements. This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant (reference number: EP/M017915/1).

References

1. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optim. **13**(4), 455–492 (1998)
2. Naujoks, B., Willmes, L., Bäck, T., Haase, W.: Evaluating multi-criteria evolutionary algorithms for airfoil optimisation. In: Guervós, J.J.M., Adamidis, P., Beyer, H.-G., Schwefel, H.-P., Fernández-Villacañas, J.-L. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 841–850. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45712-7_81
3. Keane, A.J.: Statistical improvement criteria for use in multiobjective design optimization. AIAA J. **44**(4), 879–891 (2006)
4. Forrester, A.I.J., Bressloff, N.W., Keane, A.J.: Optimization using surrogate models and partially converged computational fluid dynamics simulations. In: Proceedings of Mathematical, Physical and Engineering Sciences, vol. 462, no. 2071, pp. 2177–2204 (2006)
5. Leary, S.J., Bhaskar, A., Keane, A.J.: A derivative based surrogate model for approximating and optimizing the output of an expensive computer simulation. J. Global Optim. **30**(1), 39–58 (2004)
6. Foli, K., Okabe, T., Olhofer, M., Jin, Y., Sendhoff, B.: Optimization of micro heat exchanger: CFD, analytical approach and multi-objective evolutionary algorithms. Int. J. Heat Mass Transf. **49**(5), 1090–1099 (2006)

7. Hasbun, J.E.: Classical Mechanics with MATLAB Applications. Jones & Bartlett Publishers, Burlington (2012)
8. Shah, A., Ghahramani, Z.: Pareto frontier learning with expensive correlated objectives. In: International Conference on Machine Learning, pp. 1919–1927 (2016)
9. Daniels, S.J., Rahat, A.A.M., Tabor, G., Fieldsend, J., Everson, R.: Shape optimisation using computational fluid dynamics and evolutionary algorithms. In: 11th OpenFOAM Workshop, Portugal (2016)
10. Daniels, S.J., Rahat, A.A.M., Tabor, G., Fieldsend, J., Everson, R.: Automatic shape optimisation of the turbine-99 draft tube. In: 12th OpenFOAM Workshop, Exeter (2017)
11. Brochu, E., Cora, V.M., De Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599 (2010)
12. González, J., Dai, Z., Hennig, P., Lawrence, N.: Batch Bayesian optimization via local penalization. In: Artificial Intelligence and Statistics, pp. 648–657 (2016)
13. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: a review of Bayesian optimization. Proc. IEEE **104**(1), 148–175 (2016)
14. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multiobjective optimization. In: Abraham, A., Jain, L., Goldberg, R. (eds.) Evolutionary Multiobjective Optimization, pp. 105–145. Springer, London (2005). https://doi.org/10.1007/1-84628-137-7_6
15. Liang, J., Qu, B., Suganthan, P.: Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. Technical report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore (2014)
16. Chen, Q., Liu, B., Zhang, Q., Liang, J., Suganthan, P., Qu, B.: Problem definitions and evaluation criteria for CEC 2015 special session on bound constrained single-objective computationally expensive numerical optimization. Technical report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University (2015)
17. Weller, H., Tabor, G., Jasak, H., Fureby, C.: A tensorial approach to computational continuum mechanics using object orientated techniques. Comput. Phys. **12**(6), 620–631 (1998)
18. Daniels, S.J., Rahat, A.A.M., Tabor, G., Fieldsend, J., Everson, R.: A review of shape distortion methods available in the OpenFoam framework for automated design optimisation. In: Nóbrega, J., Jasak, H. (eds.) OpenFOAM: Selected Papers of the 11th Workshop. Springer, Heidelberg (2018, in Press)
19. Catmull, E., Clark, J.: Recursively generated B-spline surfaces on arbitrary topological meshes. Comput.-Aided Des. **10**(6), 350–355 (1978)
20. Arfken, G.B., Weber, H.J., Harris, F.E.: Mathematical Methods for Physicists: A Comprehensive Guide. Academic Press, Cambridge (2011)
21. Pitz, R., Daily, J.: An experimental study of combustion the turbulent structure of a reacting shear layer formed at a rearward-facing step. Technical report, University of California, Berkeley, California, USA, NASA Contractor Report 165427, August 1981
22. Nilsson, U.: Description of adjointShapeOptimizationFoam and how to implement new objective functions. Technical report, Chalmers University of Technology (2014)

Automated Selection and Configuration of Multi-Label Classification Algorithms with Grammar-Based Genetic Programming

Alex G. C. de Sá[1(✉)], Alex A. Freitas[2], and Gisele L. Pappa[1]

[1] Computer Science Department, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
{alexgcsa,glpappa}@dcc.ufmg.br
[2] School of Computing, University of Kent, Canterbury, UK
A.A.Freitas@kent.ac.uk

Abstract. This paper proposes Auto-MEKA$_{GGP}$, an Automated Machine Learning (Auto-ML) method for Multi-Label Classification (MLC) based on the MEKA tool, which offers a number of MLC algorithms. In MLC, each example can be associated with one or more class labels, making MLC problems harder than conventional (single-label) classification problems. Hence, it is essential to select an MLC algorithm and its configuration tailored (optimized) for the input dataset. Auto-MEKA$_{GGP}$ addresses this problem with two key ideas. First, a large number of choices of MLC algorithms and configurations from MEKA are represented into a grammar. Second, our proposed Grammar-based Genetic Programming (GGP) method uses that grammar to search for the best MLC algorithm and configuration for the input dataset. Auto-MEKA$_{GGP}$ was tested in 10 datasets and compared to two well-known MLC methods, namely Binary Relevance and Classifier Chain, and also compared to GA-Auto-MLC, a genetic algorithm we recently proposed for the same task. Two versions of Auto-MEKA$_{GGP}$ were tested: a full version with the proposed grammar, and a simplified version where the grammar includes only the algorithmic components used by GA-Auto-MLC. Overall, the full version of Auto-MEKA$_{GGP}$ achieved the best predictive accuracy among all five evaluated methods, being the winner in six out of the 10 datasets.

Keywords: Automated machine learning (Auto-ML)
Multi-label classification · Grammar-based genetic programming

1 Introduction

The outgrowing popularity of machine learning algorithms and its indiscriminate use by practitioners who do not necessarily know the peculiarities of these methods have made the area of automated machine learning (Auto-ML) [3,5,6,8,14]

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 308–320, 2018.
https://doi.org/10.1007/978-3-319-99259-4_25

more relevant than ever. The area of Auto-ML emerged to deal with the problem of how to select learning algorithms and their hyper-parameters to successfully solve a given ML problem. This problem is a hard one even for experts, which usually follow ad-hoc approaches to choose learning algorithms. In the majority of cases, such decisions are based on trial and error when testing different methods from the literature or on the recommendation of other experienced data scientists. Additionally, the algorithm's hyper-parameters are rarely deeply explored to achieve the best algorithm's performance for the given problem.

This scenario makes many ML solutions biased, incomplete and inefficient. Auto-ML proposes to deal with these problems by customizing solutions (in terms of algorithms and configurations) to ML problems. Most Auto-ML systems proposed to date focus on generating sequences of steps to solve single label classification (SLC) problems [3,5,6,8,14]. The objective of classification is to learn models from data capable of expressing the relationships between a set of predictive attributes and a predefined set of class labels. In the case of SLC, each instance is associated to a single class label.

However, there is an increasing number of applications that require associating an example to more than one class label, including image and video annotation, gene function prediction, medical diagnosis and tag suggestion for text mining. For example, in the context of medical diagnosis, a patient can be associated to one or more diseases (e.g., diabetes, pancreatic cancer and high blood pressure) at the same time. This classification scenario is better known as multi-label classification (MLC) [15]. MLC is considered a more challenging problem than SLC. First, the algorithm needs to consider the label correlations (i.e., detecting if they exist or not) in order to learn a model that produces accurate classification results. Second, the limited number of examples for each class label in the dataset makes generalization harder, as the algorithm needs more examples to create a good model from such complex data.

In the same way that MLC is harder than SLC, we consider the Auto-ML task for MLC data more challenging than the Auto-ML task for SLC data. This is because of the higher difficulty to learn from multi-label data, the strain to evaluate the produced MLC models [15], and the computational cost involved. Despite these problems, we have recently proposed the first Auto-ML method to tackle MLC [2], here referred to as GA-Auto-MLC. The method is a simple real-coded genetic algorithm (GA) that performs a search in a very large (hierarchical) search space of many different types of MLC algorithms from the MEKA framework [12]. Although GA-Auto-MLC was effective in the experiments reported in [2] (with only three datasets), its solution encoding approach has two major drawbacks: it is cumbersome and it allows individuals representing impractical MLC algorithm configurations (in the sense that the MLC algorithms could have invalid configurations or take too long to run).

GA-Auto-MLC encodes solutions using a real-valued array to code a complex hierarchical structure representing the MLC algorithms and their hyper-parameters. Although the genotype is represented by a vector of a fixed predefined size, each position of the array can map to distinct components (essential functional parts) of MLC algorithms. In other words, the genes do not have any

semantic meaning regarding the mapping to the phenotype. Because of that, when performing genetic operations (such as crossover and mutation), some operations are highly conservative (e.g., no changes occur in the phenotype after a mutation operation) while others highly destructive (e.g., abrupt changes occur in the phenotype after a mutation operation).

Aiming to address the aforementioned problems, this paper proposes a new evolutionary Auto-ML for MLC (based on the MEKA tool), namely Automated MEKA (Auto-MEKA$_{GGP}$). Auto-MEKA$_{GGP}$ is a grammar-based genetic programming method [7] capable of handling the complex hierarchical nature of the MLC search space while avoiding the generation of invalid solutions. The method was conceived to explore a larger set of MLC algorithms and components when compared to GA-Auto-MLC. Auto-MEKA$_{GGP}$ optimizes the choice of an MLC algorithm and hyper-parameter settings to the target problem.

In order to evaluate its effectiveness, Auto-MEKA$_{GGP}$ was tested in 10 datasets and compared to two well-known MLC algorithms: Binary Relevance (BR) [15] and Classifier Chain (CC) [11]. Auto-MEKA$_{GGP}$ was also compared to GA-Auto-MLC, and all comparisons were based on a combination of several multi-label predictive accuracy measures [2,15]. We run two versions of Auto-MEKA$_{GGP}$: a full version with our proposed grammar, and a simplified grammar version including only the components of GA-Auto-MLC. The results showed that Auto-MEKA$_{GGP}$ was the best method in terms of average rank, followed by its simplified version, and then GA-Auto-ML, BR and CC.

The remainder of this paper is organized as follows. Section 2 reviews related work on Auto-ML and MLC. Section 3 details the proposed method, while Sect. 4 presents and discusses the results obtained. Finally, Sect. 5 draws some conclusions and discusses directions of future work.

2 Related Work

Currently, Auto-ML methods [3,5,6,8,14] have been dealing with the optimization of complete ML pipelines. This means that, instead of just focusing on ML algorithms and their hyper-parameters, these methods are also concerned with other aspects of ML, such as data preprocessing (e.g., feature normalization or feature selection) and post-processing (e.g., classification probability calibration). Most methods proposed so far in the literature use as their search method either Bayesian optimization or evolutionary approaches.

Auto-WEKA [14] automates the process of selecting the best ML pipeline in WEKA [16], whereas Auto-SKLearn [5] optimizes the pipelines in Scikit-Learn [10]. Both methods implemented a random forest based version of a Bayesian optimization approach (i.e., Sequential Model-based Algorithm Configuration).

Evolutionary methods are also commonly used to perform this task. The Tree-Based Pipeline Optimization Tool (TPOT) [8], for instance, applies a canonical genetic programming (GP) algorithm to search for the most appropriate ML pipeline in the Scikit-Learn library. Considering a different evolutionary approach, the Genetic Programming for Machine Learning method (GP-ML) [6]

uses a strongly typed genetic programming (STGP) method to restrict the Scikit-Learn pipelines in such a way that they are always meaningful from the machine learning point of view. Finally, the REsilient ClassifIcation Pipeline Evolution method (RECIPE) [3] adopts a grammar-based genetic programming (GGP) method to search for Scikit-Learn pipelines. It uses a grammar to organizes the knowledge acquired from the literature on how successful ML pipelines look like. The grammar avoids the generation of invalid pipelines, and can also speed up the search.

All Auto-ML methods previously discussed were designed to solve the conventional *single-label* classification task. By contrast, we propose Auto-MEKA$_{GGP}$, a grammar-based genetic programming method to solve the Auto-ML task for *multi-label* data. Auto-MEKA$_{GGP}$ overcomes the major drawbacks of our previously proposed GA-Auto-MLC method [2], being able to properly handle the complex hierarchical nature of the MLC search space. It is important to point out that in this paper we focus only on algorithms and hyper-parameters (not pipelines), as the MLC search space is much bigger than the SLC search space.

Most works in the MLC literature fall into one of two approaches [15]: problem transformation (PT) and algorithm adaptation (AA). While PT creates algorithms that transform the multi-label dataset (task) into one or more single-label classification tasks (making it possible to use any SLC algorithm), AA adapts traditional single-label classification algorithms to handle multi-label data.

Among the many MLC algorithms in the literature, it is worth mentioning: Binary Relevance (BR), which learns $Q = |L|$ independent binary classifiers, one for each label in the label set L; Label Powerset (LP), which creates a single class for each unique set of labels that exists in a multi-label training set; and Classifier Chain (CC), which extends the BR method by chaining the Q binary classifiers (also one for each label), where the attribute space of each link in the chain is increased with the classification outputs of all previous links. For more details about MLC algorithms, see [1,15].

Given the very large variety of MLC algorithms in the literature—each one having its own assumptions or biases—it is clear that selecting the best MLC algorithm for a dataset is a hard task, and the use of Auto-ML is fully justified. This is because different algorithms' assumptions can lead to different predictive performances, depending on the characteristics of the dataset and the algorithms. For instance, when the BR method is selected, the label correlations are disregarded, which is beneficial for some types of datasets. However, considering the label correlations is essential for some other datasets, which makes LP and CC methods better choices. Hence, it is important to identify these patterns and map specific algorithms (with hyper-parameters) to specific datasets.

3 Automatically Selecting Algorithms and Hyper-Parameters for Multi-Label Classification

This section presents Automated MEKA (Auto-MEKA$_{GGP}$), a method conceived to automatically select and configure MLC algorithms in the MEKA

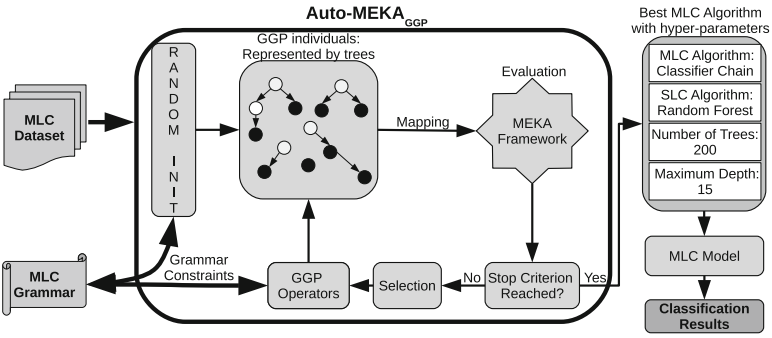


Fig. 1. The proposed method to select and configure MLC algorithms.

tool [12]. Auto-MEKA$_{\text{GGP}}$ relies on a grammar-based genetic programming (GGP) search to select the best MLC algorithm and its associated hyper-parameters to a given dataset. The GGP search naturally explores the hierarchical nature of the problem, a missing feature of our previous method [2].

As shown in Fig. 1, Auto-MEKA$_{\text{GGP}}$ receives as input an MLC dataset (with the attribute space X_F with F features and the Q class labels, L_1 to L_Q) and a grammar describing the (hierarchical) search space of MLC algorithms and their hyper-parameters. The grammar directly influences the search, as each individual created by the GGP is based on its production rules, which guarantees that all individuals are valid. In other words, the MLC grammar defines the search space and how the individuals are created and modified (see Sect. 3.1).

Auto-MEKA$_{\text{GGP}}$ works as follows. First, it creates an initial population of individuals (trees representing MLC algorithms) by choosing at random valid rules from the grammar (see Sect. 3.1), generating a derivation tree. Next, an iterative process starts. First, a mapping of each derivation tree to a specific MLC algorithm is performed. The individuals are evaluated by running the algorithm they represent within the MEKA tool on the input (see Sect. 3.2). Different MLC measures are taken into account to assess the individuals' quality, i.e., the fitness function. Next, Auto-MEKA$_{\text{GGP}}$ checks if a search stopping criterion is satisfied (e.g., a fixed number of iterations or a quality criterion). If this criterion is not satisfied, Auto-MEKA$_{\text{GGP}}$ selects individuals by using tournament selection. Next, the GGP operators (i.e., Whigham's crossover and mutation [7]) are applied on the selected individuals to create a new population. These operators also respect the grammar constraints, ensuring that the produced individuals represent valid solutions. This process goes on until the stopping criterion is satisfied. At the end of the evolution, the best individual (an MLC algorithm with its hyper-parameters) is returned, and its model is built from the full training set and evaluated in the test set (which was not accessed during the evolution), in order to measure the predictive performance of the returned individual.

It is worth noting that Auto-MEKA$_{GGP}$ was implemented using EpochX [9], an open source genetic programming framework, and is available for download[1].

3.1 Grammar: A Formal Description of the MLC Search Space

This section describes the grammar used to specify the search space of our proposed Auto-MEKA$_{GGP}$ method. The Auto-MEKA$_{GGP}$'s grammar was created based on MEKA, which is a multi-label extension to WEKA [16], and hence includes most of its algorithms. MEKA has a large variety of algorithms, focusing mainly on problem transformation methods.

We first performed a deep study of the MLC search space in MEKA: the algorithms and their hyper-parameters, the constraints associated with different hyper-parameter settings, the hierarchical nature of operations performed by problem transformation algorithms and meta-algorithms, and other issues. The grammar includes 30 MLC algorithms, exploring most algorithms in MEKA. We let some algorithms aside because of their poor performance to solve the MLC task or because of errors when testing the algorithm for different types of data. The MLC algorithms were divided into three types: problem transformation (PT), algorithm adaptation (AA) and meta-algorithms (Meta).

PT algorithms usually call the SLC algorithms to solve an MLC problem, transforming the given problem into one or various SLC problems. For these algorithms, we choose 30 SLC algorithms based on a robust method to select and configure algorithms in WEKA, i.e., Auto-WEKA [14]. On the other hand, AA methods do not need to transform the data in a preprocessing step, applying their learning process in a direct way. Finally, meta algorithms have the aforementioned MLC algorithms (PT or AA) as base algorithms, using the base classifiers' outputs in different ways to try to improve MLC performance. Considering these learning algorithms, their hyper-parameters, their dependencies and constraints, the search space of MLC algorithms has $(8.420 \times 10^{128}) + [(5.642 \times 10^{124}) \times Q] + [(1.755 \times 10^{113}) \times Q^2]$ possible MLC algorithm configurations, where Q is the number of labels of the input dataset. For more details about these possible algorithm configurations, see [1].

After studying this search space, we defined a grammar that encompasses the knowledge about MLC in MEKA, i.e., all algorithms, hyper-parameters and constraints. Formally, a grammar G is represented by a four-tuple $<N, T, P, S>$, where N represents a set of non-terminals, T a set of terminals, P a set of production rules and S (a member of N) the start symbol.

Figure 2 presents a sample of our proposed grammar. The complete version of the MLC grammar is specified in [1] and the implemented grammar (i.e., for EpochX) is also available online[2]. The proposed grammar has 138 production rules, in a total of 137 non-terminals and 230 terminals. It uses the Backus Naur Form (BNF), where each production rule has, for instance, the form $< Start >$

[1] Code and documentation are available at: https://github.com/laic-ufmg/automlc/.
[2] The implementation of the grammar(s) for EpochX is available at: https://github.com/laic-ufmg/automlc/tree/master/PPSN.

```
<Start> ::= (<MLC-PT> | <MLC-AA> | <META-MLC-LEVEL> ) <Pred-Tshd>
<MLC-PT> ::= <ALGS-PT> <ALGS-SLC>
<ALGS-SLC> ::= <ALG-TYPE> | <META-SLC1> <ALG-WEIGHTED-TYPE> | <META-SLC2> <ALG-RANDOM-TYPE> |
               (<META-SLC3> | <META-SLC4> [<ALG-TYPE>] [<ALG-TYPE>] [<ALG-TYPE>] <ALG-TYPE>) <ALG-TYPE>
                                        #META-SLC 1-4='Meta SLC algorithms with different constraints'
                                        #ALG-WEIGHTED-TYPE='Defining SLC weighted algorithms'
                                        #ALG-RANDOM-TYPE='Defining randomizable SLC algorithms'
<ALG-TYPE> ::= [<ASC>] (<Trees> | <Rules> | <Lazy> | <Functions> | <Bayes> | <Exceptions>)
...                                     #ALG-TYPE='Types of SLC algorithms'
                                        #ALGS-PT 1-4='PT methods with different constraints'
<ALGS-PT> ::= <ALGS-PT1> | <ALGS-PT2> | <ALGS-PT3> | <ALGS-PT4>
<ALGS-PT1> ::= BR | CC | LP          #BR='Binary Relevance', CC='Classifier Chain', #LP='Label Powerset'
<MLC-AA> ::= <ML-DBPNN> <ML-BPNN> | <ML-BPNN>    #ML-BPNN='Multi-Label Back Propagation Neural Network'
...                                     #ML-DBPNN= Deep ML-BPNN
<META-MLC-LEVEL> ::= <META-MLC1> | <META-MLC2> | <META-MLC3> | <META-MLC4> | MBR BR <ALGS-SLC>
                                        #MBR='BR method stacked with feature outputs'
...                                     #META-MLC 1-4='Meta MLC algorithms with different constraints'
<Pred-Tshd> ::= PCut1 | PCutL | RANDOM-REAL(0.001, 0.999)   #Pred-Tshd='prediction threshold'
```

Fig. 2. An excerpt of the proposed grammar for multi-label classification.

$::= <Meta{-}Algorithm><AlgorithmA> | <AlgorithmB> ParamA$. Symbols wrapped in "<>" represent non-terminals, whereas terminals (such as ParamA) are not bounded by "<>". The special symbols "|", "[]" and "()" represent, respectively, a choice, an optional element and a set of grouped elements that should be used together. Additionally, the symbol "#" represents a comment in the grammar, i.e., it is ignored by the grammar's parser. The choice of one among all elements connected by "|" is made using a uniform probability distribution (i.e., all elements are equally likely to occur in an individual).

3.2 From Individual Representation to Individual Evaluation

Each individual is represented by a tree, derivated from the expansion of the production rules of the MLC grammar. The mapping process takes the terminals from the tree and constructs a valid MLC algorithm from them. Given the mapped MLC algorithm in MEKA (and WEKA), the fitness function measures how effective each algorithm is for the input dataset To do this, the training set is split into two parts: a learning set (80%) and a validation set (20%). We use a stratified sampling method [13] to split the training set. Each MLC algorithm creates an MLC model from the learning set and evaluates its predictive accuracy on the validation set, using the fitness function.

MLC algorithms are usually evaluated considering several measures [15]. Hence, we set the fitness function as the average of four of these MLC measures [2,15]: Exact Match (EM), Hamming Loss (HL), F_1 Macro averaged by label (FM) and Ranking Loss (RL), as indicated in Eq. 1:

$$Fitness = \frac{EM + (1 - HL) + FM + (1 - RL)}{4} \tag{1}$$

EM is a very strict evaluation metric, as it only takes the value one when the predicted label set is an exact match to the true label set for an example, and takes the value zero otherwise. HL counts how many times a label not belonging

to the example is predicted, or a label belonging to the example is not predicted. FM is the harmonic mean between precision and recall, and its average is first calculated per label and, after that, across all the labels in the dataset. This metric is interesting because it accounts for different levels of class imbalance of the data. Finally, RL measures the number of times that irrelevant labels are ranked higher than relevant labels, i.e., it penalizes the label pairs that are reversely ordered in the ranking for a given example. All four metrics are within the $[0, 1]$ interval. However, the EM and FM measures should be maximized, whereas HL and RL should be minimized. Hence, HL and RL are subtracted from one in Eq. 1 to make the search maximize the fitness function.

4 Experimental Results

This section presents the experimental results of the proposed method in 10 datasets from the KDIS (Knowledge and Discovery Systems) repository[3]. The datasets are presented in the first two columns of Table 1, where name of the dataset is followed by a three-tuple (M, F, Q), where M is the number of examples, F is the number of features, and Q is the number of labels.

Tests are performed with two different grammar versions: a simplified version[4] that matches the search space of GA-Auto-MLC [2] and a full version[5] corresponding to the complete set of MLC components defined in this paper. The simplified version (i.e., Auto-MEKA$_{\text{GGP(S)}}$) allows us to directly compare our results to those obtained by GA-Auto-MLC.

The two versions of Auto-MEKA$_{\text{GGP}}$ and GA-Auto-MLC were executed with the following parameters: 100 individuals evolved for at most 100 generations, tournament selection of size two, elitism of five individuals, and crossover and mutation probabilities of 0.9 and 0.1, respectively. If the best individual remains the same for over five generations and the algorithm has run for at least 20 generations, we stop the evolutionary process and return that best individual. The learning and validation sets are resampled from the training set every five generations in order to avoid overfitting. Additionally, we use time and memory budgets for each MLC algorithm (generated by the evolutionary Auto-MLC methods) of 450 s (7.5 min) and 2 GB of RAM, respectively.

The algorithms produced are also compared to Binary Relevance (BR) and Classifier Chain (CC) methods. These two methods do not have hyper-parameters at the MLC level, but can use different SLC algorithms. We test them with 11 candidate algorithms [16]: Naïve Bayes (NB), Tree Augmented Naïve Bayes (TAN), Bayesian Network Classifier algorithm with a K2 search method (BNC-K2), Logistic Model Trees (LMT), Random Forest (RF), C4.5 (J48), Sequential Minimal Optimization (SMO), Multi-Layer Perceptron (MLP),

[3] The datasets are available at: http://www.uco.es/kdis/mllresources/.

[4] Available at: https://github.com/laic-ufmg/automlc/tree/master/PPSN/AutoMEKAS.bnf.

[5] Available at: https://github.com/laic-ufmg/automlc/tree/master/PPSN/AutoMEKA.bnf.

Table 1. The characteristics of the datasets, and the comparison for the versions of Auto-MEKA$_{GGP}$ and the baseline methods in the test set as to the fitness function.

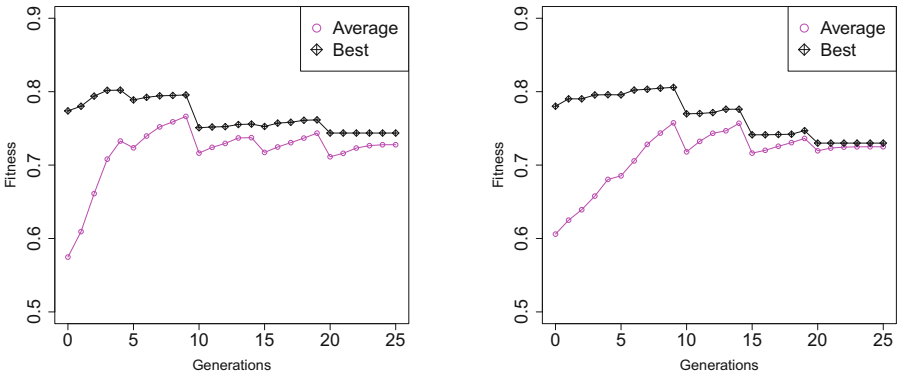
Dataset	(M, F, Q)	Auto-MEKA $_{GGP}$	Auto-MEKA $_{GGP(S)}$	GA-Auto-MLC	BR	CC
Flags	(194, 18, 7)	**0.606 (0.02)**	0.598 (0.02)	0.603 (0.03)	0.582 (0.02)	0.590 (0.04)
Scene	(2407, 294, 6)	**0.837 (0.01)**	0.830 (0.01)	0.826 (0.01)	0.824 (0.01)	0.787 (0.02)
Birds	(645, 260, 19)	**0.724 (0.02)**	0.718 (0.02)	0.722 (0.01)	0.715 (0.03)	0.657 (0.02)
Yeast	(2417, 103, 14)	0.567 (0.01)	**0.568 (0.01)**	0.565 (0.01)	0.566 (0.00)	0.552 (0.01)
GPosPse	(519, 440, 4)	**0.734 (0.04)**	0.729 (0.04)	0.721 (0.03)	0.700 (0.04)	0.697 (0.04)
CHD_49	(555, 49, 6)	**0.554 (0.02)**	0.549 (0.02)	0.550 (0.02)	0.540 (0.02)	0.524 (0.02)
WTQlty	(1060, 16, 14)	0.521 (0.01)	0.522 (0.01)	**0.524 (0.01)**	0.523 (0.02)	0.483 (0.01)
Emotions	(593, 72, 6)	0.668 (0.02)	**0.676 (0.02)**	0.674 (0.01)	0.666 (0.02)	0.627 (0.02)
Reuters	(294, 1000, 6)	0.473 (0.04)	0.475 (0.04)	**0.476 (0.05)**	0.469 (0.04)	0.457 (0.04)
Genbase	(662, 1186, 27)	**0.941 (0.01)**	0.938 (0.01)	0.938 (0.01)	0.887 (0.10)	0.934 (0.01)
Average values		0.663	0.660	0.660	0.647	0.631
Average ranks		1.800	2.250	2.250	3.900	4.800

K-Nearest Neighbors (KNN), PART and Logistic Regression (LR). All SLC algorithms use the default hyper-parameters, except for SMO which uses a Gaussian Kernel (with default hyper-parameters), and for KNN which searches for the best K value in the interval [1, 20] by performing a leave-one-out procedure based on the learning and validation sets. Note that identical time and memory budgets were applied in this local search to provide a fair comparison.

We perform the experiments using a stratified five-fold cross-validation [13] with six repetitions varying Auto-MEKA$_{GGP}$'s random seed, resulting in 30 runs per dataset for each method. Table 1 presents the (average) results of fitness function (see Eq. 1) in the test set followed by their standard deviations. For each dataset, the best average result is displayed in bold.

We use the well-known statistical approach proposed by Demšar [4] to compare different methods, using an adapted Friedman test followed by a Nemenyi post hoc test with significance level of 0.05. The last two rows of Table 1 show the average value and the average rank for each method.

As shown in Table 1, Auto-MEKA$_{GGP}$ has achieved the best (lowest) average rank (based on the fitness measure), followed by Auto-MEKA$_{GGP(S)}$ and GA-Auto-MLC, which had the same rank. BR and CC presented the worst performances. There was no statistically significant difference between the performances of the two versions of Auto-MEKA$_{GGP}$ and GA-Auto-MLC. However, Auto-MEKA$_{GGP}$ was the best method in six out of the 10 datasets in Table 1, whilst Auto-MEKA$_{GGP(S)}$ and GA-Auto-MLC were each the best in only two datasets. Finally, both versions of Auto-MEKA$_{GGP}$ (and, also, GA-Auto-MLC) performed statistically better than BR and CC, well-known MLC methods.



(a) Auto-MEKA$_{GGP}$'s behavior. (b) GA-Auto-MLC's behavior.

Fig. 3. Evolution of fitness values for the dataset GPosPse.

4.1 Evolutionary Behavior

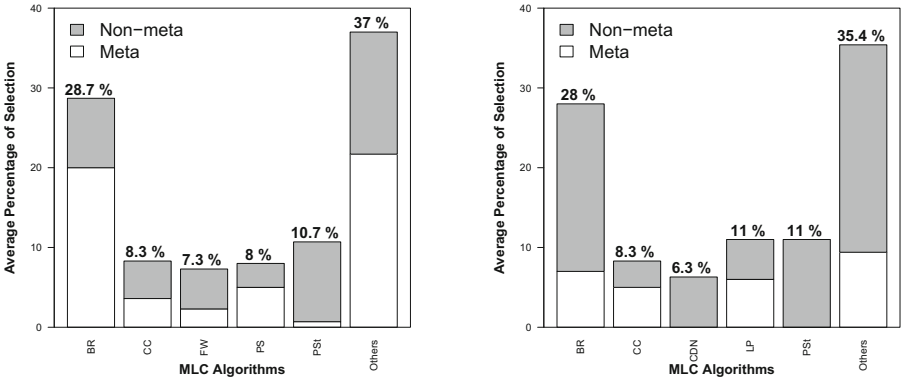
This section compares the evolutionary behaviors of Auto-MEKA$_{GGP}$ and GA-Auto-MLC. We did not include Auto-MEKA$_{GGP(S)}$ in this analysis because its results were not significantly better than those achieved by GA-Auto-MLC.

Figures 3(a) and (b) illustrate the fitness evolution of the best individuals of the population and the average fitness of the population of individuals of Auto-MEKA$_{GGP}$ and GA-Auto-MLC for the dataset GPosPse. All curves consider the mean of the only 10 runs (out of 30) with the same final number of generations (25). This dataset was chosen because it shows a situation where Auto-MEKA$_{GGP}$ is clearly better than GA-Auto-MLC, but the evolution curves are similar for other datasets. Note that the fitness values of the individuals can decrease or increase from one generation to another due to training data resampling.

Observe that Auto-MEKA$_{GGP}$'s population converges faster than GA-Auto-MLC's one. This may be due to the lack of semantic meaning of the genes in GA-Auto-ML's individuals, so a GA-Auto-MLC's individual can change severely from one generation to another by performing crossover and mutation. This is less likely to happen in Auto-MEKA$_{GGP}$ as the grammar restricts the GGP operations, which explains why the produced individuals converge quickly.

4.2 The Diversity of the Selected MLC Algorithms

This section analyzes the diversity of the MLC algorithms selected by two evolutionary Auto-ML methods: Auto-MEKA$_{GGP}$ and GA-Auto-ML. We focus only on the selected MLC algorithms (the "macro-components" of the Auto-ML methods), and not on their selected parameter settings (the "micro-components"), to simplify the analysis. We do not report results for Auto-MEKA$_{GGP(S)}$ because again the full version of this method, Auto-MEKA$_{GGP}$, obtained better results,



(a) Auto-MEKA$_{GGP}$. (b) GA-Auto-MLC.

Fig. 4. Barplots for the MLC algorithms' selection over all the 300 runs.

as discussed earlier. Analyzing the MLC algorithms selected by Auto-MEKA$_{GGP}$ and GA-Auto-ML can help us to better understand the results of Table 1, giving an idea of how the choice of an MLC algorithm influences the performance of these two Auto-ML methods.

Figures 4(a) and (b) present the bar plots to analyze the percentage of selection of MLC algorithms for the Auto-ML methods. For the full details about each MLC algorithm, see [1]. In these figures, we have for each MLC algorithm a (gray/white) bar, representing the average percentage of selection over all the 300 runs: 10 datasets times 30 independent runs per dataset (5 cross-validation folds times 6 single runs with different random seeds). These percentages rely on two cases: (i) when the traditional MLC algorithm is solely selected; (ii) when the traditional MLC algorithm is selected together with a MLC meta-algorithm. To emphasize these two cases, the bar for each traditional MLC algorithm is divided into two parts, with sizes proportional to the percentage of selection as a standalone algorithm (in gray color) and the percentage of selection as part of a meta-algorithm (in white color).

BR was the traditional MLC algorithm most frequently selected (in about 30% of all runs) by both Auto-ML methods. Besides, Classifier Chain (CC), Four-Class Pairwise Classification (FW), and Pruned Sets with and without threshold (PS and PSt) were selected in total in 34.3% of all runs by Auto-MEKA$_{GGP}$; whilst CC, Conditional Dependency Networks (CDN), LP and PSt were selected in total in 36.6% of all runs by GA-Auto-MLC. Note that the MLC algorithms most frequently selected by Auto-MEKA$_{GGP}$ and GA-Auto-ML are broadly similar, which suggests that Auto-MEKA$_{GGP}$'s superior performance is partly due to a better exploration of the space of hyper-parameter settings of those most successful MLC algorithms. Finally, we observed that Auto-MEKA$_{GGP}$ selected meta MLC algorithms in 53.3% of all runs, whilst GA-Auto-MLC selected meta MLC algorithms in only 27.4% of all runs.

5 Conclusions and Future Work

This paper introduced Auto-MEKA$_{GGP}$, a new grammar-based genetic programming method for automatically selecting the best Multi-Label Classification (MLC) algorithm and its hyper-parameter settings for an input dataset. Auto-MEKA$_{GGP}$ uses a grammar representing prior knowledge about MLC algorithms, restricting its search space to valid solutions.

Auto-MEKA$_{GGP}$ was compared to two well-known MLC algorithms – Binary Relevance (BR) and Classifier Chain (CC) – and to GA-Auto-ML, a GA we recently proposed to solve this task [2]. We tested Auto-MEKA$_{GGP}$ with the full version of the proposed grammar and with a simplified grammar version which has the same search space (candidate MLC algorithms and their hyper-parameters) as GA-Auto-ML. Overall, the full version of Auto-MEKA$_{GGP}$ obtained the highest predictive accuracy among all five tested methods, being the winner in six out of the 10 datasets. Also, both versions of Auto-MEKA$_{GGP}$, as well as GA-Auto-ML, obtained statistically significantly higher accuracies than BR and CC.

In future work we plan to extend Auto-MEKA$_{GGP}$ to search for MLC pipelines too. This means to search for the best combination of MLC algorithms, data preprocessing and post-processing methods, and their respective hyper-parameters.

Acknowledgments. This work has been partially funded by the EUBra-BIGSEA project by the European Commission under the Cooperation Programme (MCTI/RNP 3rd Coordinated Call), Horizon 2020 grant agreement 690116. In addition, this work has been partially supported by the following Brazilian Research Support Agencies: CNPq, CAPES and FAPEMIG.

References

1. de Sá, A.G.C., Freitas, A.A., Pappa, G.L.: Multi-label classification search space in the MEKA software. Technical report, UFMG (2018). https://github.com/laic-ufmg/automlc/tree/master/PPSN/MLC-SearchSpace.pdf
2. de Sá, A.G.C., Pappa, G.L., Freitas, A.A.: Towards a method for automatically selecting and configuring multi-label classification algorithms. In: Proceedings of GECCO Companion, pp. 1125–1132 (2017)
3. de Sá, A.G.C., Pinto, W.J.G.S., Oliveira, L.O.V.B., Pappa, G.L.: RECIPE: a grammar-based framework for automatically evolving classification pipelines. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 246–261. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_16
4. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
5. Feurer, M., Klein, A., Eggensperger, K., et al.: Efficient and robust automated machine learning. In: Proceedings of the International Conference on Neural Information Processing Systems, pp. 2755–2763 (2015)

6. Křen, T., Pilát, M., Neruda, R.: Automatic creation of machine learning workflows with strongly typed genetic programming. Int. J. Artif. Intell. Tools **26**(5), 1–24 (2017)
7. Mckay, R., Hoai, N., Whigham, P., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. Genet. Program Evolvable Mach. **11**(3), 365–396 (2010)
8. Olson, R., Bartley, N., Urbanowicz, R., Moore, J.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of GECCO, pp. 485–492 (2016)
9. Otero, F., Castle, T., Johnson, C.: EpochX: genetic programming in Java with statistics and event monitoring. In: Proceedings of GECCO Companion, pp. 93–100 (2012)
10. Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
11. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. Mach. Learn. **85**(3), 333–359 (2011)
12. Read, J., Reutemann, P., Pfahringer, B., Holmes, G.: MEKA: a multi-label/multi-target extension to WEKA. J. Mach. Learn. Res. **17**(21), 1–5 (2016)
13. Sechidis, K., Tsoumakas, G., Vlahavas, I.: On the stratification of multi-label data. In: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (eds.) ECML PKDD 2011. LNCS (LNAI), vol. 6913, pp. 145–158. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23808-6_10
14. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the ACM SIGKDD Conference, pp. 847–855 (2013)
15. Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining multi-label data. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 667–685. Springer, Boston (2010). https://doi.org/10.1007/978-0-387-09823-4_34
16. Witten, I., Frank, E., Hall, M.A., Pal, C.: Data Mining: Practical Machine Learning Tools and Techniques, 4th edn. Morgan Kaufmann, Burlington (2016)

Performance Assessment of Recursive Probability Matching for Adaptive Operator Selection in Differential Evolution

Mudita Sharma[1(✉)], Manuel López-Ibáñez[2], and Dimitar Kazakov[1]

[1] University of York, York, UK
{ms1938,dimitar.kazakov}@york.ac.uk
[2] University of Manchester, Manchester, UK
manuel.lopez-ibanez@manchester.ac.uk

Abstract. Probability Matching is one of the most successful methods for adaptive operator selection (AOS), that is, online parameter control, in evolutionary algorithms. In this paper, we propose a variant of Probability Matching, called *Recursive Probability Matching* (*RecPM-AOS*), that estimates reward based on progress in past generations and estimates quality based on expected quality of possible selection of operators in the past. We apply *RecPM-AOS* to the online selection of mutation strategies in differential evolution (DE) on the BBOB benchmark functions. The new method is compared with two AOS methods, namely, *PM-AdapSS*, which utilises probability matching with relative fitness improvement, and *F-AUC*, which combines the concept of area under the curve with a multi-arm bandit algorithm. Experimental results show that the new tuned *RecPM-AOS* method is the most effective at identifying the best mutation strategy to be used by DE in solving most functions in BBOB among the AOS methods.

Keywords: Parameter control · Probability matching
Differential evolution · Black-box optimisation

1 Introduction

In many optimisation algorithms, there are operations, such as crossover, mutation, and neighbourhood exploration, for which a discrete number of operators or strategies exist. Choosing the right operator is often key for improving the performance of the algorithm. Adaptive Operator Selection (AOS) is a framework that dynamically selects an operator at run-time from a finite set of choices. AOS methods are a subset of online tuning or parameter control methods [11]. Examples of AOS methods include *PM-AdapSS* [7] and *F-AUC* [5], both of which were introduced in the context of selecting a mutation strategy in differential evolution (DE) [13]. In particular, *PM-AdapSS* uses probability matching (PM)

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 321–333, 2018.
https://doi.org/10.1007/978-3-319-99259-4_26

as the method for operator selector, whereas *F-AUC* uses a method inspired by multi-armed bandits. PM was initially proposed in the context of classifier systems [6] and it was later adapted as a component of AOS methods [7].

In this work, we propose a variant of PM called Recursive Probability Matching (*RecPM*). PM probabilistically selects an operator to apply according to its estimated *quality*. The quality of each operator is calculated as the weighted sum of a reward value, which measures the impact of the most recent application of the operator on solution fitness, and its historical quality. Instead, our proposed *RecPM* estimates the quality of each operator according to a method inspired by the Bellman equation from reinforcement learning [16], which takes into account not only the reward values but also the selection probabilities of other operators. By combining *RecPM* with a credit assignment method based on offspring survival rate, we obtain the *RecPM-AOS* method.

We follow previous works [4], and apply *RecPM-AOS* to adaptively select a mutation strategy in DE for continuous optimisation, and compare our results with both *PM-AdapSS* and *F-AUC*. For completeness, we also include two state-of-the-art algorithms: a variant of DE, *JADE* [17], and an evolution strategy with covariance matrix adaptation, *CMAES* [10]. As a benchmark, we use the noiseless functions from the black-box optimisation benchmark (BBOB) [8]. Our results show that RecPM-AOS is competitive with other AOS methods.

2 Background

2.1 Adaptive Operator Selection

Adaptive Operator Selection (AOS) methods dynamically select, at each iteration t of an algorithm, one operator k out of a discrete set of K operators. The selection is based on (*1*) a credit or *reward* value $r_{k,t}$ that rewards recent performance improvements attributed to the application of the operator and (*2*) an estimated *quality* of the operator $q_{k,t}$ that accumulates historical performance or takes into account the performance of other operators. We identify two components of AOS methods: the credit assignment and the operator selection.

Credit Assignment (CA) defines the performance statistics that measure the impact of the application of an operator and assigns a reward value $r_{k,t}$ according to this impact. For example, the reward of a mutation operator may be defined in terms of the fitness of the solutions generated by its application. CA is applied after each application of the operator, possibly taking into account its past performance. For example, the CA of *F-AUC* uses a sliding window of size W to store the rank-transformed fitness obtained by the last W selected operators that generated an improved solution. A decay factor is applied to the ranks so that top-ranks are rewarded more strongly. The ranks in the window are used to compute a curve of the contribution of each operator and the area under the curve (AUC) is taken as the reward value of the operator. More details are given in the original paper [5]. On the other hand, *PM-AdapSS* only considers

the immediate performance of the operators and calculates the reward of selected operator k at iteration t as:

$$r_{k,t} = \frac{1}{N^{\text{surv}}} \sum_{i=1}^{N^{\text{surv}}} \frac{f(\boldsymbol{x}_{\text{best}}) \cdot |f(\boldsymbol{x}_i^{\text{parent}}) - f(\boldsymbol{x}_i)|}{f(\boldsymbol{x}_{\text{i}})} \tag{1}$$

where N^{surv} is the number of offspring that improved over its parent, and $f(\boldsymbol{x}_{\text{i}})$, $f(\boldsymbol{x}_i^{\text{parent}})$, and $f(\boldsymbol{x}_{\text{best}})$ are the fitness of an offspring solution generated by selected operator k, of its parent solution and of the best solution found so far, respectively. If there is no improvement or the operator was not selected at iteration t, the reward is zero.

The Operator Selector (OS) estimates the quality $q_{i,t+1}$ of each operator i, based on the reward assigned to it at iteration t, and chooses one operator to use in iteration $t+1$ among K operators according to its quality. For example, the OS in *F-AUC* uses a multi-arm bandit (MAB) technique called Upper Confidence Bound (UCB) [1] to calculate:

$$q_{i,t+1} = r_{i,t} + C \cdot \sqrt{\frac{2 \log \sum_{j=1}^{K} n_{j,t}}{n_{i,t}}} \tag{2}$$

where C is a scaling factor parameter, $n_{i,t}$ is the number of applications of operator i in the last W iterations that improved a solution, and $r_{i,t}$ is the reward assigned to operator i at iteration t. In the above equation, $r_{i,t}$ introduces exploitation whereas the second term introduces exploration. The operator selector greedily chooses the operator with the highest quality value. By comparison, *PM-AdapSS* uses probability matching (PM) to map the quality of each operator to a probability value and applies roulette-wheel selection to probabilistically choose the next operator. In particular, the quality of each operator is calculated as:

$$q_{i,t+1} = q_{i,t} + \alpha \cdot (r_{i,t} - q_{i,t}), \forall i \in K \tag{3}$$

where α is a parameter called adaptation rate. The selection probabilities for choosing an operator in iteration $t+1$ are calculated as:

$$p_{i,t+1} = p_{\min} + (1 - K \cdot p_{\min}) \left(\frac{q_{i,t+1}}{\sum_{j=1}^{K} q_{j,t+1}} \right) \tag{4}$$

where $p_{\min}$ is a minimum probability of selection. Initially, $q_{i,0} = 1$ and $p_i = 1/K$, $\forall i \in K$. Thus initially all operators have the same chance of getting selected.

Table 1 summarizes the components of the various AOS methods compared in this paper. The components of the proposed *RecPM-AOS* are described in the following Sect. 3.

Table 1. Comparison of AOS methods and their components.

		F-AUC	PM-AdapSS	RecPM-AOS
CA		Area under the curve	Average relative	Offspring survival rate (Eq. 10)
			Fitness improvement (Eq. 1)	
OS		Multi-armed bandit	Probability matching (Sect. 2.1)	Probability matching (Sect. 2.1)
OS	Selection	Greedy	Roulette wheel	Roulette wheel
	Quality	UCB (Eq. 2)	Weighted sum of quality and reward (Eq. 3)	Bellman equation (Eq. 9)
Parameters		Window size(W), scaling factor(C)	$p_{\min}$, α	$p_{\min}$, γ

2.2 Mutation Strategies in Differential Evolution

In order to evaluate different AOS methods, we apply them to the online selection of mutation strategies in differential evolution (DE) [13]. In DE, the mutation strategy creates an offspring solution $\boldsymbol{u}$ as a linear combination of three or more parent solutions $\boldsymbol{x}_i$, where i is the index of a solution in the current population. Different strategies show a different balance between exploration and exploitation in the search space and they may be applied to the current solution, a random one or the best one. Examples of such mutation strategies [2] are:

$$\begin{aligned}
\text{“DE/rand/1”:} \quad & \boldsymbol{u}_i = \boldsymbol{x}_{r_1} + F \cdot (\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}) \\
\text{“DE/rand/2”:} \quad & \boldsymbol{u}_i = \boldsymbol{x}_{r_1} + F \cdot (\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3} + \boldsymbol{x}_{r_4} - \boldsymbol{x}_{r_5}) \\
\text{“DE/rand-to-best/2”:} \quad & \boldsymbol{u}_i = \boldsymbol{x}_{r_1} + F \cdot (\boldsymbol{x}_{\text{best}} - \boldsymbol{x}_{r_1} + \boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3} + \boldsymbol{x}_{r_4} - \boldsymbol{x}_{r_5}) \\
\text{“DE/current-to-rand/1”:} \quad & \boldsymbol{u}_i = \boldsymbol{x}_i + F \cdot (\boldsymbol{x}_{r_1} - \boldsymbol{x}_i + \boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3})
\end{aligned}$$

where F is a parameter, and r_1, r_2, r_3, r_4, and r_5 are randomly generated indexes.

For a fair comparison, all AOS methods in this paper are integrated into the same DE algorithm and able to select from the set of mutation strategies shown above. The general framework of DE with AOS is shown in Algorithm 1.

3 Recursive PM (RecPM)

We propose here a novel *PM* variant called Recursive Probability Matching (*RecPM*). The main difference between *PM* and *RecPM* is that the latter estimates the quality of each operator by adapting the Bellman equation from Markov Decision Processes (MDPs) [14,16]. MDP is a framework from Reinforcement Learning for making decisions in a stochastic environment. MDP assumes that the current state is independent of the whole history given the previous state

Algorithm 1. DE with AOS

```
Initialise parameter values of DE (F, NP, CR) and AOS method
Initialise and evaluate fitness of each individual x_i in the population
t = 0 (generation number or time step)
while stopping condition is not satisfied do
  for each x_i, i = 1, ..., NP do
    if one or more operators not yet applied then
      k = Uniform selection among operator(s) not yet applied
    else
      k = Select mutation strategy based on selection method (AOS)
    Generate offspring using selected operator k
  Evaluate offspring population
  Perform credit assignment (AOS)
  Estimate quality for each operator (AOS)
  Update selection value (eg. probability) for each operator (AOS)
  t = t + 1
```

(Markov property). Bellman equation [14] is widely used in MDPs to calculate the expected return starting from a state. Although other AOS and parameter control methods have used techniques from MDP such as Q-learning and SARSA [11], our proposal is the first to be based on Bellman equation, to the best of our knowledge.

In the context of AOS, a state represents the selected operator k at a time step t and the corresponding reward is the future immediate reward assigned to the operator $r'_{k,t+1}$, which is based on the impact of the application of the operator on the performance of the algorithm. Since the next operator is chosen probabilistically, we consider only transitions between states and rewards, and not actions, thus we follow the Bellman equation for discrete decision processes [14, p. 3094], which is used to predict the next state according to the expected next reward given the current state. Our motivation for using the Bellman equation is to use the historical performance of operators to predict their quality in the next iteration, which is then mapped to their probability of selection.

We use the Bellman equation to estimate the quality $q_{k,t}$ of an operator k after its application in iteration t as the expected value ($\mathrm{E}[\cdot]$) of its total sum of discounted future rewards:

$$q_{k,t} = \mathrm{E}[r'_{k,t+1} + \gamma r'_{k,t+2} + \cdots \mid K_t = k] = \mathrm{E}[\sum_{z=0}^{\infty} \gamma^z r'_{k,t+z+1} \mid K_t = k] \tag{5}$$

$$= \mathrm{E}[r'_{k,t+1} + \gamma \sum_{z=0}^{\infty} \gamma^z r'_{k,t+z+2} \mid K_t = k] \quad \text{(using recursive property)} \tag{6}$$

$$= r_{k,t+1} + \sum_{j=1}^{K} P_{kj} \left[\gamma \mathrm{E} \left[\sum_{z=0}^{\infty} \gamma^z r'_{k,t+z+2} \mid K_{t+1} = j \right] \right] \quad \text{(assuming } \mathrm{E}[r'_k] = r_k) \tag{7}$$

$$= r_{k,t+1} + \gamma \sum_{j=1}^{K} P_{kj} q_{j,t+1} \quad \text{(using definition of } q_{k,t} \text{ in Eq. 5)} \tag{8}$$

where $r_{k,t+1}$ is the accumulated reward that stores all the past achievements (accumulated reward) for operator k and γ is the discount rate. In the context of AOS, we do not know the probability matrix P of size $K \times K$, thus we decided to calculate each entry as $P_{k,j} = p_k + p_j$, that is, as the sum of the selection probabilities of operators k and j.

The rationale behind the formula above is as follows: When estimating $q_{k,t}$, operator k competes with all other operators $j \in K$, including itself, since the selection of other operators in the past has impact on the current performance of the selected operator. Thus, their probabilities are added and multiplied by by the quality estimate of operator j. These values are then aggregated in the end to get an overall estimate for operator k. The quality is an estimate not because of the expected values, which are assumed to be completely provided by the method, but because $q_{j,t+1}$ is not known and the current estimate at t is used instead. When considering all operators, this forms a system of linear equations and can be re-written in the following vector form:

$$\boldsymbol{Q}_t = \boldsymbol{R}_{t+1} + \gamma P \boldsymbol{Q}_t \quad \text{or} \quad \boldsymbol{Q} = (1 - \gamma P)^{-1} \boldsymbol{R} \tag{9}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R} = [r_i]$ are the K-dimensional quality and reward vectors that are updated at the end of each iteration t. The system of linear equations can be solved efficiently by matrix inversion [14] when the number of operators is small. $\boldsymbol{Q}$ is then normalised using the softmax function, which "squashes" each real value to a K-dimensional vector in the range $(0, 1)$ using the exponential function. Once the quality is estimated for each operator, the probability vector $\boldsymbol{p} = [p_i]$ and probability matrix P are updated as in *PM-AdapSS* (Eq. 4) and used for the selection of an operator.

RecPM utilises the steps of Probability Matching as described in Sect. 2.1 except for the definition of operator quality, which is estimated using the Bellman equation as shown above. However, to obtain an AOS method, we still need to specify the credit assignment method that updates the reward values after the application of the selected operators at time step t. We propose to calculate the immediate reward $r'_{k,t+1}$ assigned to the selected operator as the number N_t^{surv} of offspring that survive to the next generation $t+1$ divided by the population size NP. We define the accumulated reward $r_{k,t}$ assigned to an operator as the ratio of offspring that survived plus half the last accumulated reward. The remainder unselected operators receive half of their accumulated reward. Thus, each operator gets a fraction of last reward value, that stores its historical performance, and the selected one gets extra reward. The value of 0.5 as weight assigned to $r_{k,t}$ was chosen by intuition.

$$r_{i,t+1} = \begin{cases} r'_{k,t+1} + 0.5 \cdot r_{k,t}, & \text{if } k \text{ is selected} \\ 0.5 \cdot r_{i,t}, & \forall i \neq k \end{cases}, \text{ where } r'_{k,t+1} = \frac{N_t^{\text{surv}}}{NP} \tag{10}$$

The rational behind this credit assignment is that, if the operator is unlucky and not getting selected for enough number of generations, it still receives some reward based on its past performance and it has a chance of being selected in

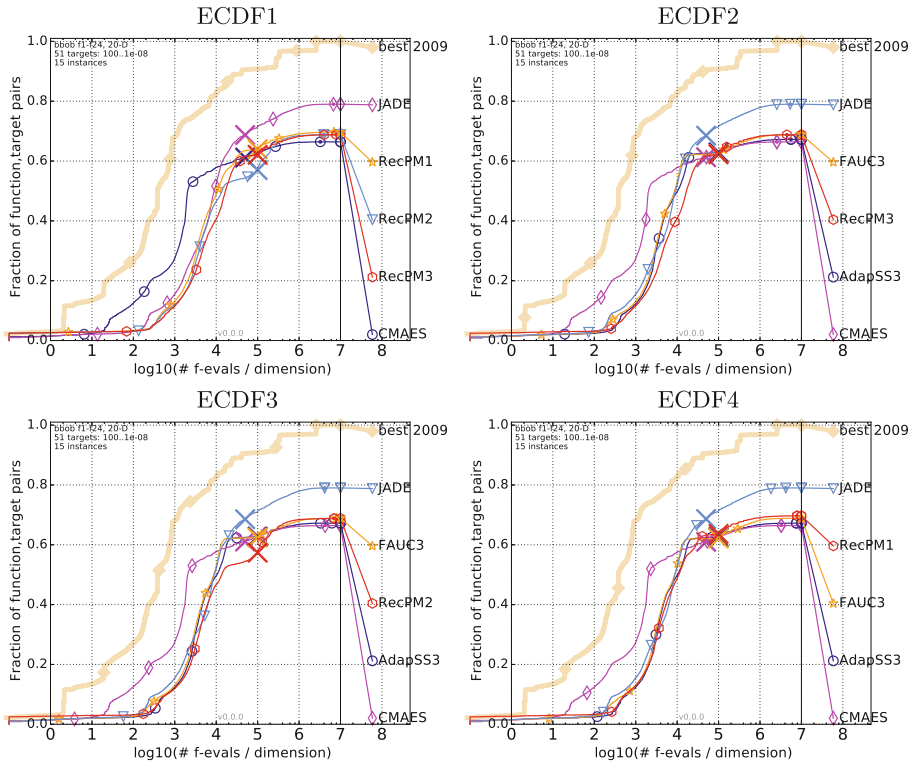


Fig. 1. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions in 20-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

the future. This ensures that such operator is not discarded completely and may be selected after a certain number of generations.

The combination of *RecPM* with the above credit assignment leads to a new AOS method named *RecPM-AOS* in the following. When comparing *PM-AdapSS* and *RecPM-AOS*, the former uses *PM* as an operator selector whereas the latter uses *RecPM*. Both AOS methods use a credit assignment based on the number of improvements from parent to offspring (N^{surv}), however, *PM-AdapSS* uses average relative fitness improvement (Eq. 1) as immediate reward without using accumulated reward, whereas *RecPM-AOS* uses offspring survival rate as immediate reward combined with a fraction of its previous accumulated reward.

RecPM-AOS is integrated within DE (Algorithm 1) to make DE more efficient by adaptively selecting, at run-time, a mutation strategy among the four mutation strategies shown in Sect. 2.2. DE combined with *RecPM-AOS* has five parameters: three belong to DE, namely, scaling factor (F), population size

(NP) and crossover rate (CR), while discount factor (γ) and minimum selection probability ($p_{\min}$) belong to *RecPM-AOS*.

4 Experimental Analysis

In the following, we compare the performance of proposed *RecPM-AOS* within DE with two other algorithms, namely *DE-F-AUC* [5] and *PM-AdapSS-DE* [7], for the online selection of mutation strategies in DE. More advanced DE variants are available in the literature, however, we want to understand and analyse the impact of the various AOS methods without possible interactions with other adaptive components of those variants. Nonetheless, for the sake of completeness, we also compare our results with two state-of-the-art algorithms *JADE* [17] and *CMAES* [10]. *JADE* is a DE variant that uses a mutation strategy called "current-to-pbest" and adapts the crossover probability CR and mutation factor F using the values which proved to be useful in recent generations. *CMAES* is an evolution strategy that samples new candidate solutions from a multivariate Gaussian distribution and adapts its mean and covariance matrix.

We use BBOB (Black-box optimisation Benchmarking) [8] test suite to test the algorithms. BBOB provides an easy to use tool-chain for benchmarking black-box optimisation algorithms for continuous domains and to compare the performance of numerical black-box optimisation algorithms. We evaluate all algorithms on the 24 noiseless continuous benchmark functions [9] provided by BBOB, each with 15 different instances, totalling to 360 function instances. Each instance of a function is a rotation and/or translation of the original function leading to a different global optimum. These 24 functions are grouped in five classes, namely, separable, moderate, ill-conditioned, multi-modal and weak-structure functions. Each algorithm with AOS method is run to a maximum number of $10^5 \cdot d$ function evaluations (FEvals), where d is the dimension of the benchmark function. In this paper, we focus on $d = 20$ for all functions. The solutions in the initial population for each function instance are generated with different seeds.

4.1 Parameter Tuning

We tune the parameters of the *DE-RecPM-AOS*, *DE-F-AUC* and *PM-AdapSS-DE* using the offline automatic configurator IRACE [12], which saves the hassle of manual tuning and allows for a fully specified and reproducible procedure. The input given to IRACE is the range of all parameters that need tuning (Table 2) and a set of training function instances; it then looks for good performing parameter configurations by executing the target algorithm on different training instances with a budget of 10^4 FEvals. In our case, the training set consist of only 10% of the function instances, randomly selected within each class, to avoid over-fitting.

In order to evaluate the impact of parameter tuning, we consider three parameter configurations of each algorithm. The first configuration is obtained by tuning all parameters of DE and the AOS methods. The second configuration is

Table 2. Optimal parameter configurations selected from the range shown below the parameter name. The following prefix abbreviations are used: RecPM for DE-RecPM-AOS, AdapSS for PM-AdapSS-DE and FAUC for DE-F-AUC. The symbol '-' in the table means that the parameter is not applicable to the AOS method.

	F $[0.1, 2.0]$	NP $[50, 400]$	CR $[0.1, 1.0]$	α $[0.0, 1.0]$	$p_{\min}$ $[0.0, 0.25]$	γ $[0.1, 1.0]$	W $[0, 200]$	C $[0.0, 1.0]$
RecPM1	0.47	168	0.98	-	0.17	0.75	-	-
RecPM2	0.5	200	1.0	-	0.11	0.46	-	-
RecPM3	0.5	200	1.0	-	0.0	0.6	-	-
AdapSS1	0.51	117	0.97	0.48	0.22		-	-
AdapSS2	0.5	200	1.0	0.86	0.04	-	-	-
AdapSS3	0.5	200	1.0	0.6	0.0	-	-	-
FAUC1	0.24	96	0.55	-	-	-	31	0.14
FAUC2	0.5	200	1.0	-	-	-	5	0.35
FAUC3	0.5	200	1.0	-	-	-	50	0.5

obtained by tuning only the parameters of the AOS methods, while the parameter values of DE are taken from [4]: $CR = 1.0$, $F = 0.5$ and $NP = 200$. The value $CR = 1.0$ means that a mutation strategy is applied to each dimension of all parents, which maximizes the impact of the mutation strategies. Finally, the third configuration (*default*) uses the settings suggested in [4] for *DE-F-AUC* and *PM-AdapSS-DE*, which uses the DE settings described earlier and AOS settings tuned with a different configurator. All parameter configurations are shown in Table 2.

4.2 Comparison of AOS Methods with Different Parameter Settings

After tuning, each obtained configuration is evaluated on the full BBOB benchmark set. We use plots of the Empirical Cumulative Distribution Function (ECDF) to assess their performance (Fig. 1). The ECDF displays the proportion of problems solved within a specified budget of function evaluations (FEvals) for different targets $f_{\text{target}} = f_{\text{opt}} + \Delta f$, where f_{opt} is an the optimum function value to reach with some precision $\Delta f \in [10^{-8}, 10^{2}]$. In the plots, FEvals is given on the x-axis and y-axis represents the fraction of problems solved. A large symbol '$\times$' shows the maximum number of function evaluations given to each algorithm, in our case, $10^5 \cdot d$ FEvals are given to each algorithm with AOS method. Results reported after this symbol use bootstrapping to estimate the number of evaluations to reach a specific target for a problem [3]. The results denoted with `best 2009` correspond to the artificial best algorithm from the BBOB-2009 workshop constructed from the data of the algorithm with the smallest aRT (average Run Time) for each set of problems with the same function, dimension and target. The aRT is calculated as the ratio of the number of function evaluations for

reaching the target value over successful runs (or trials), plus the maximum number of evaluations for unsuccessful runs, divided by the number of successful trials. Data to generate ECDF graphs for DE-F-AUC3, PM-AdapSS-DE3, CMAES and JADE is obtained directly from the COCO website.[1] The trials that reached f_{target} within specified budget are termed as successful trials, *#succ*. Table 3 shows the aRT (average Run Time), calculated as the ratio of the number of function evaluations for reaching the target value over successful runs, plus the maximum number of evaluations for unsuccessful trials, divided by the number of successful trials, on four out of 24 functions only due to limited space. The runtime for a function becomes undefined if there are no successful runs. The complete table can be seen in the supplementary material [15].

We expected to tune *DE-F-AUC* and *PM-AdapSS-DE* algorithms with the hope to replicate the original results for *DE-F-AUC3* and *PM-AdapSS-DE3* [5,7]. But we could not match the results shown in these papers. Thus, we decided to use the data available online at the COCO website and compare variants of proposed algorithm with *DE-F-AUC3* and *PM-AdapSS-DE3* only. The interested reader is referred to the supplementary material [15] to find the results of tuned *DE-F-AUC* and *PM-AdapSS-DE* algorithms. The ECDF graphs of variants of proposed algorithm with *DE-F-AUC3* and *PM-AdapSS3* are shown in four plots of Fig. 1 that show the performance of algorithms averaged over all 360 functions tested. From now on we only talk about the original results and not the replicated ones.

ECDF1 shows results obtained for three variants of *DE-RecPM-AOS*. As expected, proposed algorithm with all tuned parameters outperformed its all other variants both in terms of speed and percentage of problems solved. When all three AOS methods use the default settings (ECDF2), it is estimated that *F-AUC* and *RecPM-AOS* solves the same number of problems but within given budget all algorithms solved same number of problems with varied speed. The third graph (ECDF3) where only parameters of AOS method are tuned in proposed algorithm shows that *DE-F-AUC3* and *PM-AdapSS-DE3* solves maximum problems with almost same speed within given budget. But when given more FEvals, according to bootstrapping technique, *DE-RecPM-AOS2* shows the same performance as *DE-F-AUC3* by solving the same number of problems whereas *PM-AdapSS-DE3* could not match the performance of other two algorithms. ECDF4 compares results of *DE-RecPM-AOS1*, *DE-F-AUC3* and *PM-AdapSS-DE3*. The proposed method with all tuned parameters that is, parameters of DE algorithm and of *RecPM-AOS* method outperformed all other algorithms by solving 75% of the problems. This is clearly because of the properties the proposed AOS method has. The tuned configurations of replicated algorithms: *DE-F-AUC* and *PM-AdapSS-DE* are not better than the original results reported, which we cannot replicate.

Summing up the above discussion, it can be said that tuning all the parameters of the proposed algorithm (*DE-RecPM-AOS1*) outperformed all its variants, thus tuning on training set plays an important role. It also outperformed all other

[1] http://coco.gforge.inria.fr/doku.php?id=algorithms-bbob.

Table 3. Average runtime (aRT in number of function evaluations) divided by the respective best aRT measured during BBOB-2009 in dimension 20. RecPM: DE-RecPM-AOS, AdapSS: PM-AdapSS-DE, FAUC: DE-F-AUC. The different target Δf-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Best results are printed in bold.

20-D

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ	Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1	43	43	43	43	43	43	43	15/15	**f13**	652	2021	2751	3507	18749	24455	30201	15/15
AdapSS3	101	196	291	377	465	646	827	15/15	AdapSS3	28	13	12	12	2.6	2.6	2.6	15/15
CMAES	**7.5**	**13**	**20**	**26**	**33**	**45**	**58**	15/15	CMAES	**6.3**	**5.1**	**4.5**	**4.4**	**1.9**	4.6	8.4	12/15
FAUC3	93	180	265	350	431	597	763	15/15	FAUC3	25	12	11	11	2.4	**2.5**	**2.5**	15/15
JADE	47	94	143	191	240	340	437	15/15	JADE	17	14	15	14	3.6	4.8	9.0	15/15
RecPM1	96	187	278	369	459	636	819	15/15	RecPM1	29	14	14	14	3.2	3.6	4.2	15/15
RecPM2	114	219	317	410	510	707	932	15/15	RecPM2	36	17	16	16	3.6	4.1	7.2	15/15
RecPM3	132	263	432	932	1234	1621	2076	14/15	RecPM3	44	32	32	38	7.8	8.8	8.0	15/15
f2	385	386	387	388	390	391	393	15/15	**f14**	75	239	304	451	932	1648	15661	15/15
AdapSS3	52	63	73	83	93	112	132	15/15	AdapSS3	43	34	43	40	25	20	2.8	15/15
CMAES	36	43	45	47	**47**	**48**	**50**	15/15	CMAES	**4.2**	**3.0**	**3.7**	**4.3**	**4.2**	**6.2**	**1.2**	15/15
FAUC3	48	58	68	77	86	105	123	15/15	FAUC3	33	30	38	36	23	19	2.8	15/15
JADE	**28**	**34**	**39**	**44**	50	61	71	15/15	JADE	18	18	23	25	20	38	62	5/15
RecPM1	47	57	66	76	85	103	121	15/15	RecPM1	40	33	42	41	26	24	4.2	15/15
RecPM2	72	90	108	127	147	186	223	15/15	RecPM2	52	44	58	53	32	27	4.3	15/15
RecPM3	66	111	180	205	278	333	360	15/15	RecPM3	47	43	54	51	35	38	5.1	15/15

AOS methods within DE solving 75% of the total problems. Thus, historical information preserving property in the form of reward and using Bellman equation to estimate quality of operator led to efficient adaptability of operators. On the other hand both F-AUC and RecPM-AOS makes use of past performances of operators, we do that by defining reward of each operator capturing a fraction of its last reward which reduces the hassle of maintaining a window of certain size. However, *F-AUC* and *PM-AdapSS* shows similar speed in solving a fixed number of problems and *DE-RecPM-AOS1* has faster convergence speed and increased percentage of problems solved. The full table showing aRT for AOS methods within DE algorithm in supplementary material shows that no one algorithm has best converging speed for all functions and *DE-F-AUC3* and *DE-RecPM-AOS1* shows competitive results.

4.3 Comparison of *RecPM-AOS* with State-of-the-Art Algorithms

CMAES and JADE are given a budget of $5 \cdot 10^4$ FEvals. When comparing different versions of *DE-RecPM-AOS* with *CMAES* and *JADE*, proposed algorithm with all tuned parameters is able to solve most functions than *CMAES* as seen in ECDF4 in Fig. 1 that is, almost 10% more than best version of *DE-RecPM-AOS*: *DE-RecPM-AOS1*. However, *JADE* manages to solve majority of the problems than any AOS methods within DE, shown in ECDF1. In the initial runs, *CMAES* has faster convergence speed than any other algorithm.

5 Conclusion and Future Work

We proposed a variant of probability matching, *recursive-PM*, as a parameter control method that gives the quality as an aggregated estimate of future perfor-

mances of operators. The proposed adaptive operator selector adaptively selects a mutation strategy in Differential Evolution. The algorithm differs from classical PM in the way it assigns the quality to a strategy. The reward given to an operator depends on the short term success of that operator. It is compared with two AOS methods *DE-F-AUC*, *PM-AdapSS-DE* and two state-of-the-art algorithms *CMAES*, *JADE*. The overall performance of *Recursive-PM* within DE with tuned parameters shows that it outperforms other two AOS methods that is, *DE-F-AUC* and *PM-AdapSS-DE*, and *CMAES* by solving 75% of the problems. The proposed algorithm could not outperform *JADE*, but had similar convergence rate.

IRACE is used to find the good offline settings for the proposed AOS method, which illustrates the usefulness of offline procedures to successfully design new online adaptation methods. It is used to train the parameters on 10% of the total function instances. We plan to extend the proposed algorithm by integrating it with different definitions of credit assignment to compete with the state of the art algorithms. To make *RecPM-AOS* perform better, we plan to extend proposed algorithm by finding and tuning more parameters involved in the method such as the fraction of previous reward to take under consideration when designing credit assignment technique.

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**(2–3), 235–256 (2002)
2. Das, S., Mullick, S.S., Suganthan, P.N.: Recent advances in differential evolution-an updated survey. Swarm Evol. Comput. **27**, 1–30 (2016)
3. Efron, B., Tibshirani, R.J.: An Introduction to the Bootstrap. CRC Press, Boca Raton (1994)
4. Fialho, Á., Schoenauer, M., Sebag, M.: Fitness-AUC bandit adaptive strategy selection vs. the probability matching one within differential evolution: an empirical comparison on the BBOB-2010 noiseless testbed. In: Pelikan, M., et al. (eds.) GECCO (Companion), pp. 1535–1542 (2010)
5. Fialho, Á., Schoenauer, M., Sebag, M.: Toward comparison-based adaptive operator selection. In: Pelikan, M., et al. (eds.) 2010 Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Portland, Oregon, USA, pp. 767–774 (2010)
6. Goldberg, D.E.: Probability matching, the magnitude of reinforcement, and classifier system bidding. Mach. Learn. **5**(4), 407–425 (1990)
7. Gong, W., Fialho, Á., Cai, Z.: Adaptive strategy selection in differential evolution. In: Pelikan, M., et al. (eds.) GECCO, pp. 409–416 (2010)
8. Hansen, N., Auger, A., Mersmann, O., Tusar, T., Brockhoff, D.: COCO: a platform for comparing continuous optimizers in a black-box setting. Arxiv preprint arXiv:1603.08785 (2016)
9. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: noiseless functions definitions. Technical report, RR-6829, INRIA, France (2009). (updated February 2010)
10. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)

11. Karafotias, G., Hoogendoorn, M., Eiben, A.E.: Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans. Evol. Comput. **19**(2), 167–187 (2015)
12. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
13. Price, K., Storn, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization. Springer, New York (2005). https://doi.org/10.1007/3-540-31306-0
14. Rust, J.: Structural estimation of Markov decision processes. In: Handbook of Econometrics, vol. 4, pp. 3081–3143. Elsevier (1994)
15. Sharma, M., López-Ibáñez, M., Kazakov, D.: Performance assessment of recursive probability matching for adaptive operator selection in differential evolution: supplementary material (2018). https://doi.org/10.5281/zenodo.1257672. https://github.com/mudita11/AOS-comparisons
16. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
17. Zhang, J., Sanderson, A.C.: JADE: adaptive differential evolution with optional external archive. IEEE Trans. Evol. Comput. **13**(5), 945–958 (2009)

Program Trace Optimization

Alberto Moraglio[1(✉)] and James McDermott[2]

[1] University of Exeter, Exeter, UK
A.Moraglio@exeter.ac.uk
[2] University College Dublin, Dublin, Ireland
James.McDermott2@ucd.ie

Abstract. We introduce Program Trace Optimization (PTO), a system for 'universal heuristic optimization made easy'. This is achieved by strictly separating the problem from the search algorithm. New problem definitions and new generic search algorithms can be added to PTO easily and independently, and any algorithm can be used on any problem. PTO automatically extracts knowledge from the problem specification and designs search operators for the problem. The operators designed by PTO for standard representations coincide with existing ones, but PTO automatically designs operators for arbitrary representations.

Keywords: Universal optimisation · Operator design
Genotype-phenotype mappings

1 Introduction

In the 1960s and onwards, researchers in genetic algorithms proposed a vision of them as 'universal solvers', capable of addressing any search and optimization problem. Later, researchers found that this promise could not be delivered because each new problem required a significant investment of time and expertise in tailoring the algorithm to the problem for acceptable performance. This is common to all metaheuristics: solvers succeed only when using an encoding and search operators which are well-chosen for the problem at hand. This is hard work, to be done per problem, and a black art, which requires (often unwritten) expertise in both the problem and the solver.

The PTO vision is to make universal optimisation easy. This is achieved by (i) neatly separating problem specification and solver definition, and (ii) automatically tailoring the solver to the problem by analysing and using the structure of the problem specification. In PTO, the core task for the user is to write a *generator*, which implicitly defines the set of possible solutions, rather than design a *representation and search operators* as in most EAs. PTO will then induce these automatically. For some problems, a good representation and search operators are easy to create and PTO's will be equivalent. In others, a generator is easier to create: it is a more concrete task, perhaps more aligned with the thinking of domain experts, as opposed to metaheuristics experts. In yet other cases, a

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 334–346, 2018.
https://doi.org/10.1007/978-3-319-99259-4_27

generator may be an easy way to avoid constraint violations or express search bias difficult to express through operators (e.g. the Heuristic TSP generator of Sect. 4). PTO's automatic design of search operators deals seamlessly with arbitrarily complex representations. The PTO user does not have to think about how to optimise, while the researcher working on PTO solvers does not have to think about specific problems.

2 Overview of PTO

PTO has two key ingredients: (i) a *universal solution representation* – the program trace – that decouples problem and solver; (ii) a *naming scheme* on the trace reflecting the problem structure that automatically adapts generic search operators to the problem at hand. In the following, we briefly describe these.

2.1 Universal Solution Representation

Any search method requires an implicit or explicit representation for generation and manipulation of solutions. PTO uses a universal solution representation that applies to any problem. Metaheuristics defined on such a representation can be applied *unchanged* to any problem, thus becoming universal optimisers. The representation used by PTO is as follows. The user supplies a solution generator, which implicitly defines the set of possible solutions. PTO runs it and traces its execution, resulting in a sequential data structure called a program trace. The trace can be thought of as the sequence of outcomes of (random) decisions made by the generator in producing a particular solution. The trace can be manipulated: it can be 'played back' in the generator to redo the same sequence of decisions and produce the same solution; it can be edited and played back to produce a variant solution; two parent traces can be combined and the result played back to produce a child solution. That is, the trace is a genotype, the solution is the corresponding phenotype, and the playback mechanism in the generator is a developmental mapping; editing and recombination of traces are search operators. The trace is a universal representation that applies to any problem because it is implicit in the problem definition (in the generator) and can be extracted automatically by tracing. No other representation can be more general or more powerful, since the generator can use Turing-complete code.

2.2 Naming Scheme

The *trace* is a linear structure for any problem; the corresponding *solution* can be any arbitrarily complex structure as the generator can use any construct of the programming language, e.g. data structures, function calls and recursion, and can return any data structure. The linear genotype allows for easy adaptation of existing metaheuristics such as genetic algorithms or particle swarm optimization to work in PTO, while indirectly searching spaces of arbitrarily complex phenotypes. PTO annotates the trace so that it captures the problem structure

implicitly expressed by the user in the generator. The key observation is that the control structures used in the code of the generator reflect the structure of the solutions generated, e.g., a solution with a matrix structure may be reflected in the use of nested loops in the generator; a solution with a tree structure may be reflected in the use of multiple recursion in the generator. To make this information about solution structure available to the search operators acting on the linear genotype, the trace is annotated with the 'execution context' in which each random decision (a trace entry) took place. This execution context contains information about the state of the process (a picture in time of the running program), e.g., including the line of code from which the random number generator was invoked, the values held in the loop indices, and the current stack of (possibly recursive) calls. Search operators for annotated traces preserve the execution context as much as possible, to prevent the equivalent of the disruptive "ripple effect" [10] of Grammatical Evolution (GE). Generic search operators on the trace representation induce domain-specific search operators on the phenotype which are equivalent to known-good operators for standard representations. This mechanism of automatic implicit adaptation of the search operators to the structure of the problem at hand naturally extends to arbitrarily complex solution structures, as any solution structure can be described by a generator using a few fundamental control structures (i.e., sequence, condition, iteration, subroutine), which are handled by the annotation scheme.

2.3 PTO Software Architecture

PTO has a modular design in three parts. The tracer hooks into the user-supplied generator, and records and plays back program traces. The solver is pluggable: any metaheuristic solver can be plugged-in as the solver. The problem is also pluggable: the user supplies an objective function and a generator. Advances in any component do not require changes in the others.

User Interface: PTO automatically makes all design decisions and parameter configurations. This makes the user interface unobtrusive and the learning curve flat, and allows for reuse of existing code, e.g. EA initialisation methods or constructive heuristics as generators (as in PODI [6]).

Tracer: The tracer interfaces to the problem definition by tracing the generator when it is run, and playing back modified traces in the generator. This is achieved by overriding the calls to any function of the Python standard random library used by the generator, so that tracing and playing back are invisible to the user. The tracer annotates the trace entry by dynamic analysis of the running code of the generator as described in Sect. 3.1. The annotated trace is the interface between the tracer and the solvers. Search operators work on annotated traces but are only allowed to use the labels to align entries with the same label in different traces, and cannot use any other information stored in the labels. This decouples tracer and solvers, allowing the naming scheme (the information stored in the labels) to be modified without the need to alter solvers.

Search Algorithm: The requirement for a meta-heuristic to work with PTO is that it can work on the trace representation. This is a *dictionary*, i.e. a variable-length linear structure with entries identified by names rather than by indexes, and with heterogenous entries (real, binary, integer, and permutations – the output domains of random number generators in the Python library). Generalising meta-heuristics to work on the trace representation can be done in a principled way using the geometric framework [7] so as to retain the original dynamics of the meta-heuristic on the new representation. The meta-heuristics currently in PTO include Genetic Algorithm, Stochastic Hill-Climbers, Geometric Particle Swarm Optimisation [8] and Random Search.

2.4 Related Work

PTO builds on and combines previous ideas. The idea of using a sequence of decisions as a genotype was originally introduced in the Program Optimisation with Dependency Injection (PODI) system [6], with emphasis on using it with complex and smart generators, to evolve complex constrained structures. PODI is similar to the *decision chain encoding* [5]. Constructive heuristics and simulated annealing have also been hybridized for vehicle routing problems [1]. PODI can be thought of as a generalisation of GE [9] that, instead of using a grammar to map linear genotypes to complex phenotypes, uses an unrestricted program i.e., the generator, which is much more expressive. GE and by extension PODI can suffer from low locality [11] and the ripple effect [10] as a result of the mapping.

PTO goes beyond PODI and the other approaches by seeing the trace as a universal representation, allowing generic meta-heuristics to be used on any problem. The PTO philosophy of separating problem specification and solver mirrors that of Probabilistic Programming (PP) [4]. Connections between PTO and PP run deep. The PTO trace representation and naming scheme are the translation to an optimisation context of a successful PP approach [13] to inference over complex probabilistic models. Links between PP and EC have previously been made, including optimisation on a type of program trace [2]. The idea of automatic implicit design – that generic search operators on the trace representation induce domain-specific search operators at the "phenotype" level – is novel to PTO. This will be described in some detail in Sect. 3.

PTO differs from other works. There are approaches for the automatic design of search operators for specific problems based on hyper-heuristics e.g., [3]. PTO automatic design is different from these approaches, as it does not rely on searching the space of search operators but rather on extracting and using implicit problem-knowledge from the problem specification. Also PTO design differs from that of mathematical theories for the design of search operators such as the geometric framework [7], as in PTO the design is automatic and implicit rather than manual and explicit. PTO aims at shielding users from choosing parameters. This is not done by automatic off-line parameter tuning and algorithm configuration [12], but rather by choosing default robust parameter values in a principled manner that work well in many situations. PTO aims at being very

easily extendible and encompasses a large number of problems and search algorithms, and its design is highly modular. But unlike existing software libraries for metaheuristics, PTO is a self-configuring system which seamlessly adapts to any problem and any arbitrarily complex representation.

The No Free Lunch theorem [14] does not prevent PTO from good universal performance, because as will be described, PTO automatically tailors the operators to the problem. Thus, PTO is effectively not a black-box method.

3 Implicit Operator Design

Here, we define the naming scheme and generic search operators on traces. For illustrative purposes, we then give two examples using *simplified* naming schemes to show how the naming scheme implicitly adapts generic search operators to problem-specific search operators by using the control structure in the code of the generator. Finally, we discuss how problem knowledge is embedded in PTO.

3.1 Naming Scheme

We follow the approach of Wingate et al. [13] and name random choices according to their structural position in the execution trace, which we define in a way roughly analogous to a stack address: a random choice's name is defined as the list of the functions, their line numbers, and their loop indices, that precede it in the call stack. A trace annotated in this way is called a *structured trace*. The formal specification is inductive, as shown in Algorithm 1 (some details omitted).

Algorithm 1. Annotating a structured trace.

Begin executing the generator with empty function, line, and loop stacks.
When entering a function,
- push a unique function id on the function stack
- push a 0 on the line stack.

When moving to a new line, increment the last value on the line stack.
When starting a loop, push a 0 on the loop stack.
When iterating through a loop, increment the last value on the loop stack.
When exiting a loop, pop the loop stack.
When exiting a function, pop the function stack and the line stack.
When a random choice is made, name it with the entire contents of all stacks.

We also define a *linear trace*. It can be seen as a special case of the structured trace, in which each entry is named after its sequential position in the trace, thus more similar to PODI [6]. The search operators defined next work on both linear and structured traces.

3.2 Search Operators on Named Traces

Initialisation runs the generator and traces its execution. **Point mutation** picks a random entry of the trace and replaces its value with a value drawn from the same random call. **Uniform crossover** aligns parent traces on their names (i.e. dictionary keys). For names that appear in both parents, the offspring inherits the corresponding entries from either parent at random, i.e. using a random mask to select. For names that appear in only one parent, the offspring inherits all of them. **Repair** is applied after each alteration of the trace, i.e., after the application of any variation operator. The repair takes place when running the modified trace in the generator in playback mode to generate the corresponding solution. If there is a mismatch, i.e. the current value comes from a random call other than the one identified by the name, then a new random value is drawn from the correct random call. If the trace is used up before the generator finishes, the trace is extended with new random entries as needed. Excess entries in the trace, not used by the generator, are deleted.

3.3 Example: Loops and Matrices

The user writes a generator and objective function using standard Python:

```
1: def generator():
2:     s = random.randint(1,5)
3:     return [[random.randint(1,15) for i in range(s)]
4:                                   for j in range(s)]
5: def objective(matrix): return determinant(matrix)
```

In this simple example, the generator outputs a random square matrix, such as those shown below (next page). The objective function (to be maximised) is the determinant, calculated by recursively decomposing into sub-matrices.

Tracing. When the generator runs to generate a solution, a sequence of random events take place. The sequence of outcomes of the random events is the trace associated with the solution. In the example, the generator makes 1 call to `random.randint(1,5)` and then s^2 calls to `random.randint(1,15)`. The (unannotated) trace is a sequence of the outcomes of these random calls, e.g., `T=(3,9,13,5,1,11,7,3,7,4)`. In this example, the trace (genotype) is a list of integers, and the solution (phenotype) is a square matrix of integers, e.g., trace `T` corresponds to the matrix at bottom of page on left.

Playback. Given a trace, the corresponding solution is found by running the generator in 'playback mode' using the trace to override the source of randomness when random calls are made. This is the genotype-phenotype mapping.

Mutation and Linear Trace. Point mutation changes a single entry of the trace. E.g., in the trace `T` if 13 changes to 4, then `T` ⇒ `(3,9,4,5,1,11,7,3,7,4)`, corresponding to the change in solution illustrated below on the left. The change in the trace `T` ⇒ `2,9,13,5,1,11,7,3,7,4)` corresponds to the change illustrated

in the centre. When the trace is played back in the generator, excess entries are deleted to obtain `(2,9,13,5,1)`. The mutation has changed the size of the matrix from 3 to 2, and the original trace is scanned sequentially (i.e., played back) to fill in the smaller matrix (and the surplus elements are discarded).

$$\begin{pmatrix} 9 & 13 & 5 \\ 1 & 11 & 7 \\ 3 & 7 & 4 \end{pmatrix} \Rightarrow \begin{pmatrix} 9 & 4 & 5 \\ 1 & 11 & 7 \\ 3 & 7 & 4 \end{pmatrix}; \begin{pmatrix} 9 & 13 & 5 \\ 1 & 11 & 7 \\ 3 & 7 & 4 \end{pmatrix} \Rightarrow \begin{pmatrix} 9 & 13 \\ 5 & 1 \end{pmatrix}; \begin{pmatrix} 9 & 13 & 5 \\ 1 & 11 & 7 \\ 3 & 7 & 4 \end{pmatrix} \Rightarrow \begin{pmatrix} 9 & 13 \\ 1 & 11 \end{pmatrix}$$

This is however not satisfactory, as the mutation operator is treating these square matrices as sequential objects. A more satisfactory mutation operator would act as illustrated above on the right, extracting a square submatrix of size two from the original one.

Mutation and Structured Trace. The structured trace aims to fix this problem. For illustrative purposes, in this example we use the simplified naming scheme with line number of the call and loop indices `i` and `j`, i.e., of the form `[line number, i, j]`, instead of the general one in Sect. 3.1. The original trace `T`, i.e., `(3,9,13,5,1,11,7,3,7,4)`, is then annotated as follows:

```
([2,-,-]:3, [3,1,1]:9, [3,1,2]:13, [3,1,3]:5, [3,2,1]:1, [3,2,2]:11,
[3,2,3]:7, [3,3,1]:3, [3,3,2]:7, [3,3,3]:4).
```

As before, the first element of the trace is mutated from 3 to 2. When played back in the generator, elements of the structured trace are not accessed sequentially but by the context in their names, e.g., when line 3 is executing with `i = 1`, and `j = 2`, the decision returned from `random.randint(1,15)` is 13. Excess elements not used in play-back are deleted. Since now size = 2, when playing back the mutated trace in the generator the values of `i` and `j` that will be encountered are 1 and 2 (never 3), producing the modified trace:

```
([2,-,-]:2, [3,1,1]:9, [3,1,2]:13, [3,2,1]:1, [3,2,2]:11).
```

Induced Phenotypic Operators. The structured naming scheme and generic trace operators combine to give induced phenotypic operators tailored to matrices. In mutation, when size changes, the top left square matrix is retained. Crossover works by aligning parent matrices at the top left corner before recombination, as illustrated in Table 1.

3.4 Example: Recursion and Expressions

The example in this section illustrates that PTO with linear trace suffers from a disruptive "ripple effect" similar to GE's due to offspring genotype entries being used out of context. This is resolved by using the structured trace which results in implicit design of meaningful search operators for tree structures.

Below is a recursive generator in standard Python for random Boolean expressions on three variables contained in strings.

```
1: def gen():
2:   expr_type =  random.choice(['var','uop','biop'])
3:   if expr_type == 'var':
4:     return random.choice(['x1','x2','x3'])
5:   if expr_type == 'uop':
6:     return 'not ' + gen()
7:   if expr_type == 'biop':
8:     return '(' + gen() + random.choice([' and ',' or ']) + gen() + ')'
```

Tracing. Let us consider an example expression `E = (x2 or x1)`. The unannotated trace for this expression is `T = [biop, var, x2, or, var, x1]`.

Mutation and Linear Trace. The entries of the trace have types, so when we apply point mutation to an entry it can be replaced only with values of the same type e.g., the first entry of `T` with `biop` can be changed only to `var` or `uop` i.e., possible outputs of `random.choice(['var','uop','biop'])`. Point mutation on the *linear* trace can have a global effect. For example, changing `var` in the second entry of `T` to `uop` alters the context of all subsequent elements i.e., results in type-mismatch for all of subsequent elements when the mutated trace is played back in the generator to obtain the corresponding solution. The trace is then repaired by resolving type mismatch errors in the trace by replacing erroneous values by freshly generated values of the correct type, e.g. changing `E=(x2 or x1)` $\Rightarrow$ `(not x3 and x3)` with trace `[biop, uop, var, x3, and, var, x3]`.

Mutation and Structured Trace. Continuing the example, using the simplified naming scheme `[function, line number]*` instead of the general one, the original trace `T`, i.e., `[biop, var, x2, or, var, x1]`, is annotated as in Fig. 1(a). It can be rendered as a tree as in Fig. 1(b). This is analogous to a GE derivation tree, but distinct structurally: e.g. `x1` is not a child of `var` and one does not read the leaves to obtain the output (recall, it is *returned* by the generator).

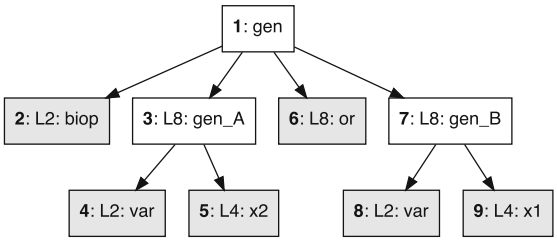
Table 1. Uniform crossover on structured traces for matrices: (right) recombine aligned traces $p1$ and $p2$ using random mask m to obtain the child trace uc, which when repaired becomes c; (left) phenotypic effect of crossover.

$$p1 = \begin{pmatrix} 9 & 13 & 5 \\ 1 & 11 & 7 \\ 3 & 7 & 4 \end{pmatrix} \quad p2 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad c = \begin{pmatrix} 9 & 2 \\ 1 & 4 \end{pmatrix}$$

name	p1	p2	m	uc	c
$[2,-,-]$	3	2	2	2	2
$[3,1,1]$	9	1	1	9	9
$[3,1,2]$	13	2	2	2	2
$[3,1,3]$	5			5	
$[3,2,1]$	1	3	1	1	1
$[3,2,2]$	11	4	2	4	4
$[3,2,3]$	7			7	
$[3,3,1]$	3			3	
$[3,3,2]$	7			7	
$[3,3,3]$	4			4	

```
[gen, 2] : biop
[gen, 8, gen-A, 2] : var
[gen, 8, gen-A, 4] : x2
[gen, 8] : or
[gen, 8, gen-B, 2] : var
[gen, 8, gen-B, 4] : x1
```

(a)



(b)

Fig. 1. (a) Annotated structured trace for `(x2 or x1)`. To read this we say, e.g., "`x2` is the result of a call on line 4 of the first call (unique ID created by addition of `-A`) to `gen` on line 8 of `gen`." (b) "Derivation" tree. Bold integers indicate execution order; 'L' indicates line number; node labels (e.g. `biop`) indicate results of random calls; edges indicate random calls from parent to child; shading indicates results of terminal random calls.

When we apply the same point mutation to `T` as before (`var` ⇒ `uop` at the second element, that is at the leaf labelled **4** in Fig. 1(b)), now with the structured trace, the mutation affects only the local context and it is much less disruptive, this time changing `E` e.g. `(x2 or x1)` ⇒ `(not x3 or x1)`.

Table 2. Uniform crossover on structured traces for expressions: recombine aligned traces $p1$ and $p2$ using random mask m to obtain the child trace uc, which when repaired becomes c; the phenotypic effect of crossover is: $p1$ = `(x2 or x1)` $p2$ = `(not x3 and x3)` c = `(not x3 or x1)`.

Name	p1	p2	m	uc	c
[gen, 2]	biop	biop	2	biop	biop
[gen, 8]	or	and	1	or	or
[gen, 8, gen-A, 2]	var	uop	2	uop	uop
[gen, 8, gen-A, 4]	x2			x2	
[gen, 8, gen-B, 2]	var	var	1	var	var
[gen, 8, gen-B, 4]	x1	x3	1	x1	x1
[gen, 8, gen-A, 6, gen, 2]		var		var	var
[gen, 8, gen-A, 6, gen, 4]		x3		x3	x3

Induced Phenotypic Operators. The structured naming scheme and generic trace operators combine to induce phenotypic operators tailored to nested expressions. The phenotypic operators have a *modularity* property.

Mutation: when a decision node C (a leaf in the tree view of the annotated trace) is mutated, no other change may be needed, so the effect is a point muta-

tion on the expression. In the worst case, mutation invalidates the contents of all subtrees, of parent node P, following C. This is because (i) P (a function call) is scoped and modular, and (ii) the execution order is mirrored in the ordering of siblings of C, so a change in C cannot have an effect on past execution.

Crossover (see Table 2): aligning parents on names corresponds to aligning their common tree structures at root (homologous crossover) with the effect on phenotypes that corresponding sub-expressions between parents are exchanged.

Analogously to these two examples (Sect. 3.3 on matrices and Sect. 3.4 on trees), the general naming scheme in Sect. 3.1 can induce phenotypic operators tailored to any solution structure because it encompasses all control structures i.e., sequential, conditional, nested and recursive function calls, nested loops, and any composition of these, that can describe any generator.

3.5 Implicit Problem Knowledge

Different generators for the same problem lead to different annotation on traces, different induced phenotypic operators, and so to different performance. PTO assumes that the control structures used by the user coding the generator implicitly reflect an understanding of the inherent structure of the problem and its underlying implicit representation. E.g., the user could have used a flat list representation for the matrix problem. This generator would not embed problem knowledge, and would lead to a structured trace coinciding with the linear trace, hence possibly worse performance. The assumption of PTO is that users will see the patterns and structures of the problem and use these in writing generators. We believe that this is a realistic assumption, and users will do this naturally and intuitively. PTO then automatically carries out design based on user's intuition on the problem. The resulting phenotypic operators will be effective to the extent that the generator captures the structure of the problem.

Perfecting the naming scheme based on experience of how users use the system in practice, and providing explicit guidelines to the user of how to effectively put structure in generators, is an important line of future research.

4 Experiments and Results

Is the structured trace better than the linear trace? Do "smart" generators boost performance? To seek preliminary answers, we test PTO on two very different domains: travelling salesman problems, and symbolic regression with grammatical evolution. We use three solvers: *Random search* (RS), simple *Hill-climbing* (HC), and an *Evolutionary Algorithm* (EA). The budget is set to 20,000 evaluations for all experiments. For the EA, PTO internally sets the number of generations = population size = $\sqrt{20000} = 141$.

For **symbolic regression by GE**, the grammar for n variables is:

```
<expr> ::= <op>(<expr>, <expr>) | <var> | <const>
<op> ::= add | sub | mul | aq
<var> ::= x1 | x2 | ... | xn
<const> ::= 0.0 | 0.1 | ... | 1.0
```

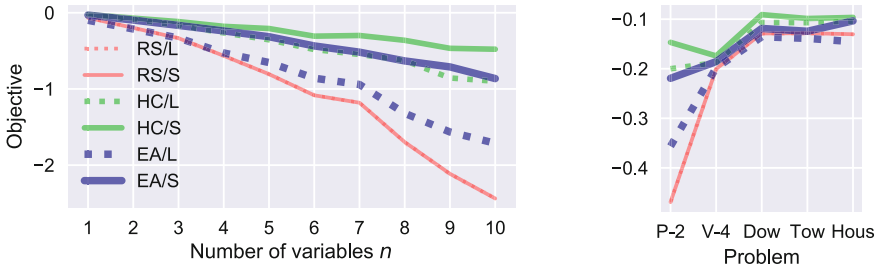


Fig. 2. Regression results (mean of 30) for polynomials (left) and dataset problems (right).

Here, `aq` is the analytic quotient $aq(x, y) = x/\sqrt{1.0 + y^2}$. We use the natural generator which creates a program by making uniform choices among all possible productions at each step. There is no maximum depth or weighting by tree depth. The objective is -RMSE. We report results on several problems. Synthetic instances are polynomials on n variables with degree $d = 4$, for $n = \{1, \ldots, 10\}$, with coefficients uniform in $[0, 1]$ for all possible terms. Well-known benchmarks are also used: Pagie-2D, Vladislavleva-4, Dow Chemical, Tower, and Housing[1]. We compare 3 solvers and 2 trace types, *Linear* (L) and *Structured* (S). We show results on training data only since the goal is to investigate search performance rather than generalisation. Results are shown in Fig. 2. These show that the best combinations use the structured trace: HC/S and EA/S. HC/L does very well, whereas EA/L is very weak. RS does very badly on larger synthetic problems, and for RS the trace type makes no difference. For synthetic problems, the differences are stronger for larger problems, demonstrating PTO scaling.

For **travelling salesman problems** we define two generators. In the *Unbiased* (U) generator, a random permutation is generated by starting with the integers $\{1, \ldots, n\}$, and for each index, swapping with a randomly-chosen later index. In the *Heuristic* (H) generator, a route is constructed by starting with a random starting city, and at each step, choosing the next city randomly from all those remaining, with their probabilities inversely weighted by the distance from the current city. Results on 6 TSPLIB[2] instances for 3 solvers and 2 generators are shown in Table 3. On larger problems in particular, EA/H is the best combination: it out-performs each of its components (EA/U and RS/H).

Overall, the structured trace gives consistently better performance than the linear trace, and "smart" generators can do very well. One surprising result is the good performance of hill-climbing with a linear trace in GE.

[1] Datasets taken from http://www.github.com/ponyge/ponyge2.
[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

Table 3. TSP results presented as mean (stddev). Lower is better. Integers in instance names (48–575) indicate problem size. Bold indicates best result per-instance.

Solver	Gen	att48	berlin52	eil101	u159	a280	rat575
RS	U	112397 (3521)	23108 (464)	2854 (43)	381521 (5157)	30203 (420)	104536 (935)
RS	H	71150 (2084)	15878 (448)	2029 (40)	222018 (3822)	18542 (304)	64691 (608)
HC	U	67573 (4875)	14699 (827)	**1628** (79)	198736 (8807)	16390 (520)	59456 (1385)
HC	H	**59976** (6092)	**13850** (1110)	1888 (100)	201490 (12426)	18286 (1084)	65858 (2172)
EA	U	68377 (4678)	14103 (913)	1747 (97)	224842 (9276)	18936 (499)	70865 (1183)
EA	H	67503 (5192)	14727 (1065)	1910 (65)	**112444** (6228)	**10086** (279)	**34949** (705)

5 Conclusions and Future Work

PTO provides a novel perspective on heuristic optimization by neatly separating problem specification and search algorithm, and automatically tailoring search operators to the problem at hand. We believe PTO has great potential and envisage an ambitious research plan, organised in three research strands.

(i) Problems: PTO will be extended to other optimisation paradigms such as multi-objective, co-evolution, dynamic and noisy objective functions. A broad range of complex real-world applications will be tested, from logistics to interactive art. Generators for many problems will be borrowed from throughout the fields of heuristics and EC and tested.

(ii) Trace: Variant naming schemes will be investigated for performance and for their effect on operator design. The naming scheme will be used in a GE variant which avoids the ripple effect in a principled way. Alternative styles of writing generators will be investigated, and the results provided to users as guidelines.

(iii) Algorithms: Many more metaheuristics will be plugged-in to PTO, including several already in the geometric framework [7], such as Surrogate-Based Optimisation and Estimation of Distribution Algorithms. Robust parameter settings and self-adaptive parameters will be investigated. Landscape analysis will be used to validate algorithm design and generator choices. The links between PTO and probabilistic programming will be used to import and export ideas.

Finally, PTO will be promoted as a community resource. Code is available at https://github.com/program-trace-optimisation.

References

1. de Armas, J., Keenan, P., Juan, A.A., McGarraghy, S.: Solving large-scale time capacitated arc routing problems: from real-time heuristics to metaheuristics. Ann. Oper. Res. 1–28 (2018)
2. Batishcheva, V., Potapov, A.: Genetic programming on program traces as an inference engine for probabilistic languages. In: Bieger, J., Goertzel, B., Potapov, A. (eds.) AGI 2015. LNCS (LNAI), vol. 9205, pp. 14–24. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21365-1_2

3. Burke, E.K., et al.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
4. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Future of Software Engineering, pp. 167–181. ACM (2014)
5. Janssen, P., Kaushik, V.: Decision chain encoding: evolutionary design optimization with complex constraints. In: Machado, P., McDermott, J., Carballal, A. (eds.) EvoMUSART 2013. LNCS, vol. 7834, pp. 157–167. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36955-1_14
6. McDermott, J., Carroll, P.: Program optimisation with dependency injection. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş., Hu, B. (eds.) EuroGP 2013. LNCS, vol. 7831, pp. 133–144. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37207-0_12
7. Moraglio, A.: Towards a geometric unification of evolutionary algorithms. Ph.D. thesis, University of Essex (2008)
8. Moraglio, A., Di Chio, C., Poli, R.: Geometric particle swarm optimisation. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 125–136. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71605-1_12
9. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, Norwell (2003)
10. O'Neill, M., Ryan, C., Keijzer, M., Cattolico, M.: Crossover in grammatical evolution. Genet. Program. Evolvable Mach. **4**(1), 67–93 (2003)
11. Rothlauf, F., Oetzel, M.: On the locality of grammatical evolution. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 320–330. Springer, Heidelberg (2006). https://doi.org/10.1007/11729976_29
12. Stützle, T., López-Ibáñez, M.: Automatic (offline) configuration of algorithms. In: Laredo, J.L.J. (ed.) GECCO (Companion), pp. 681–702. ACM, New York (2015)
13. Wingate, D., Stuhlmueller, A., Goodman, N.: Lightweight implementations of probabilistic programming languages via transformational compilation. In: Gordon, G., et al. (eds.) AISTATS. PMLR, vol. 15, pp. 770–778, 11–13 April 2011
14. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. Trans. Evol. Comput. **1**(1), 67–82 (1997)

Sampling Heuristics for Multi-objective Dynamic Job Shop Scheduling Using Island Based Parallel Genetic Programming

Deepak Karunakaran(✉), Yi Mei, Gang Chen, and Mengjie Zhang

School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
{deepak.karunakaran,yi.mei,aaron.chen,mengjie.zhang}@ecs.vuw.ac.nz

Abstract. Dynamic job shop scheduling is a complex problem in production systems. Automated design of dispatching rules for these systems, particularly using the genetic programming based hyper-heuristics (GPHH) has been a promising approach in recent years. However, GPHH is a computationally intensive and time consuming approach. Parallel evolutionary algorithms are one of the key approaches to tackle this drawback. Furthermore when scheduling is performed under uncertain manufacturing environments while considering multiple conflicting objectives, evolving good rules requires large and diverse training instances. Under limited time and computational budget training on all instances is not possible. Therefore, we need an efficient way to decide which training samples are more suitable for training. We propose a method to sample those problem instances which have the potential to promote the evolution of good rules. In particular, a sampling heuristic which successively rejects clusters of problem instances in favour of those problem instances which show potential in improving the Pareto front for a dynamic multi-objective scheduling problem is developed. We exploit the efficient island model-based approaches to simultaneously consider multiple training instances for GPHH.

Keywords: Scheduling · Genetic programming · Parallel algorithms

1 Introduction

Job shop scheduling is a complex problem in manufacturing industries. In general, researchers consider deterministic scheduling problems in which once the information of a new job is obtained it remains constant. But in practice, due to events like machine breakdown, variation in raw material quality, operator availability, etc. uncertainty is ubiquitous in manufacturing environments. In fact, with increasing uncertainty, scheduling becomes more difficult [8]. Dispatching rules have been widely used for generating schedules for different objectives

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 347–359, 2018.
https://doi.org/10.1007/978-3-319-99259-4_28

and have shown good success for DJSS problems [12], particularly for scheduling under uncertainty [9]. But designing them requires considerable expertise and rigorous experimentation. Genetic programming based hyper-heuristic (GPHH) approach has been very successful in automatically evolving dispatching rules [12]. Due to the flexible representation and powerful search ability of GP, existing GPHH methods can evolve very good rules for diverse shop scenarios [7] while handling multiple objectives for DJSS problems [10].

GPHH in general demands high computational cost and time. To address this issue, surrogate models [5] have been proposed as an alternative but they suffer from poor accuracy among other drawbacks [10]. Another key technique to reduce computation time is to use parallel evolutionary algorithms [14]. There are two main categories of parallelization, *parallelizing an independent run* and *island models*. Island models are particularly interesting because of their ability to deal with local optima [14]. The island model uses a spatially structured network of subpopulations (on different processors) to exchange promising individuals among each other is an effective approach. For example, [15] use a specialized island model for multi-objective optimization. [1] is a recent work which demonstrates the effectiveness of asynchronous parallel evolution for hyper-heuristics.

For GPHH to be effective, it is important to use a large training set containing instances with diverse characteristics. This is more important when we consider uncertain manufacturing environments as they present varying shop scenarios [7]. Moreover, when we consider multiple objectives the importance of using diverse training set is compounded. For example, makespan and total tardiness are two frequently considered conflicting objectives. Minimizing the makespan results in high throughput where as minimizing tardiness requires jobs to be not very late. A conflicting scenario arises when a set of jobs with long processing times but shorter deadline compete with a set of jobs with shorter processing times and longer deadlines. For higher throughput the shorter jobs must be completed first as against the longer jobs which adversely affects the tardiness. For evolving good dispatching rules it is important to present the evolutionary system with training instances which capture scenarios highlighting all such conflicts amongst the objectives, under different shop scenarios.

Therefore, for evolving dispatching rules for practical environments we need a large and diverse training set. But this in turn will increase the already high computational cost of GPHH. To address this problem, we need a method which can effectively sample training instances to significantly improve the effectiveness of GPHH without exceeding the computational budget. The parallel island model has already been shown to be both efficient and effective for GPHH when dealing with multiple objectives [6]. Motivated by this success, we address our issue with the idea of associating the multiple islands with different training instances. Furthermore, we utilize the inherent potential of migration policies [13] to introduce a cooperative behavior among islands and collectively evolve better rules.

In this work, we consider multi-objective DJSS problems with a focus on uncertainty in processing times which affects all our objectives. Our primary goal is to develop a sampling heuristic for GPHH which selects good training

instances toward evolving a significantly better Pareto front. To this end, our specific objectives are: (1) Develop a feature extraction method toward clustering the training dataset into DJSS problem instances with different characteristics. (2) Develop a sampling heuristic which iteratively rejects the clusters (*successive reject heuristic*) in favor of those which have the potential to improve the Pareto front. (3) Determine the migration policies of the island model toward improving the efficacy of the proposed sampling heuristic.

2 Background

DJSS problems are characterized by continuous arrival of jobs from a Poisson process [12]. n_j operations constitute a job j with the constraint that they must be processed in a predefined route say $(o_{j,1} \rightarrow o_{j,2} \rightarrow, \ldots, o_{j,n_j})$. Each operation can be processed only on one machine in its route. Other conditions/assumptions which are followed in this work are no preemption, no recirculation of jobs, no machine failure and zero transit times. Total tardiness, makespan, total flow time are some of the objectives considered for DJSS problems. In this work, we consider two potentially conflicting objectives [12]: (1) total weighted tardiness (TWT) and (2) makespan (C_{max}).

$$TWT = \Sigma w_j \times \max(C_j - d_j, 0),$$

where $C_j =$ completion time, $d_j =$ due date and w_j are weights.

$$C_{max} = \max(f_j),$$

where f_j is the flowtime of a job.

3 Proposed Method

3.1 Clustering of DJSS Problem Instances

We extract features from the DJSS problem instances based on the parameters: number of operations per job, processing time, due date factor and uncertainty in processing time. Firstly, the basic features for each job are extracted as described

Table 1. Job features

Feature	Description
#operations	Number of operations per job
p	Estimated processing time of the job
Δ^p	$\frac{p'}{p}$, p' is the actual processing time with uncertainty
Due date factor (ddf)	$\frac{(\delta_{duedate} - \delta_{reldate})}{p'}$; where $\delta_{duedate}$ is the due date and $\delta_{reldate}$ is the release date

in Table 1. Once the feature values are aggregated for all the jobs in an instance; the first, second and third quartiles of each aggregate are calculated to form a 12-dimensional feature vector characterizing each problem instance.

Consider an example of a DJSS problem instance with just 10 jobs

$$\{j_1, j_2, j_3, \ldots j_{10}\}$$

For each job the features described in Table 1 are calculated and aggregated e.g., for processing time the aggregated feature values are:

$$\{p_1, p_2, p_3, \ldots p_{10}\}$$

Then for each feature aggregate the quartiles are calculated. The feature vector of the DJSS instance is of the form

$$\{\#ops_{Q1}, \#ops_{Q2}, \#ops_{Q3}, p_{Q1}, p_{Q2}, p_{Q3}, \Delta^p_{Q1}, \Delta^p_{Q2}, \Delta^p_{Q3}, ddf_{Q1}, ddf_{Q2}, ddf_{Q3}\}$$

The objective of extraction of these features is to cluster the problem instances into classes with different characteristics. Considering our earlier example, assume we have one DJSS problem instance with high variability in number of operations per job and their processing times and another instance with low variability. It can be easily seen that our feature vectors corresponding to the two problem instances can be used to differentiate them.

$\mathcal{T}$ is the training set containing n DJSS problem instances. We extract features for all these problems and cluster them into $\mathcal{C}_1$ and $\mathcal{C}_2$ using K-means clustering. We apply K-means clustering again on each of these clusters to yield $\{\mathcal{C}_{11}, \mathcal{C}_{12}\}$ and $\{\mathcal{C}_{21}, \mathcal{C}_{22}\}$ respectively. This process can be repeated to obtain more sub-clusters $\{\{\mathcal{C}_{111}, \mathcal{C}_{112}\}, \{\mathcal{C}_{121}, \mathcal{C}_{122}\}\}$ and $\{\{\mathcal{C}_{211}, \mathcal{C}_{212}\}, \{\mathcal{C}_{221}, \mathcal{C}_{222}\}\}$ and so on.

3.2 Proposed Island Model

We use two classes of islands in our evolutionary system. The first class of islands, represented as G in Fig. 1(a) and (b), sample training instances from the set $\mathcal{T}$ throughout the evolutionary process. The second class of islands represented as A and B in Fig. 1(b) sample problem instances from the different clusters the choice of which varies with generations. The appropriate choice of the cluster is controlled by the *successive reject heuristic* (SRH).

We first describe the evolutionary process of island G in Algorithm 1. In every generation, a new training instance is sampled from $\mathcal{T}$. Unless otherwise mentioned, we use sample to denote a *simple random sample*. Due to our familiarity with NSGA-II [3] and the fact that we consider only two objectives in this work, we chose NSGA-II as our underlying evolutionary algorithm. In line 3, an iteration of NSGA-II is performed. After each generation, the migration policy determines (line 4) if there will be an exchange of individuals among the islands. The output is a set of dispatching rules which can generate a Pareto front. Note

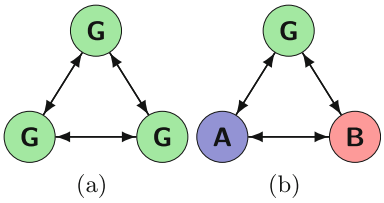


Fig. 1. (a) Standard island model, (b) Island model for successive reject heuristic

Algorithm 1. Island G

Input: $\mathcal{T}$
Output: $\{\omega_1, \omega_2, \ldots, \omega_p\}$
1 **for** $g \leftarrow 1 : N_G$ **do**
2 Sample an instance $\mathcal{I} \in \mathcal{T}$.
3 Run g^{th} iteration of NSGA-II using $\mathcal{I}$.
4 Receive/Send individuals using migration policies.
5 **end**
6 Collect the genetic programs corresponding to the Pareto front : $\{\omega_1, \omega_2, \ldots, \omega_p\}$.

Algorithm 2. Island Z ($Z \in \{A, B\}$)

Input: $\mathcal{C}_Z$,$\mathcal{N}_{SRH}$
Output: $\{\omega_1^z, \omega_2^z, \ldots, \omega_p^z\}$
1 **for** $g \leftarrow 1 : N_G$ **do**
2 Sample an instance $\mathcal{I} \in \mathcal{C}_Z$.
3 Run g^{th} iteration of NSGA-II using $\mathcal{I}$.
4 Receive/Send individuals using migration policies.
5 **if** $g \in \mathcal{N}_{SRH}$ **then**
6 $\mathcal{C}_Z \leftarrow SRH(P_k^A, P_k^B, \mathcal{C}_A, \mathcal{C}_B)$
7 **end**
8 Collect the genetic programs corresponding to the Pareto front : $\{\omega_1^z, \omega_2^z, \ldots, \omega_p^z\}$.

that the final output of the parallel evolutionary system is the combination of outputs from all individual islands (Table. 2).

In Algorithm 2, we describe the evolutionary process of the islands A and B (Fig. 1(b)). Both islands are similar, except that they sample their training instances from different clusters. Their migration policies are the same. The cluster $\mathcal{C}_Z$ is changed at discrete stages during the evolutionary process. The set $\mathcal{N}_{SRH}$ contains the generations at which the successive reject hypothesis is invoked to change $\mathcal{C}_Z$ (line 6). The rest of the procedure is the same as in Algorithm 1.

Table 2. Notation

Notation	Description
$\mathcal{T}$	Set of all DJSS problem instances for training
N_G	Total number of generations for evolutionary process
$\mathcal{C}_Z$	Cluster corresponding to island $Z \in \{A, B\}$
P_k^Z	Top k individuals from island $Z \in \{A, B\}$
$M_{\overrightarrow{X,Y}}$	Migration policy from island X to Y
P_{2k}	Combined list of top k individuals from islands A and B
TOP_k	List of top k individuals across island A and B
$TOP^{\mathcal{Z}_k}$	#individuals which are present both in $P_k^{\mathcal{Z}}$ and TOP_k, $\mathcal{Z} \in \{A, B\}$
$\mathcal{N}_{SRH}$	Set of generations at which SRH is invoked

Successive Reject Heuristic. The successive reject heuristic (SRH) is described in Algorithm 3. When the SRH is invoked by islands A and B at generation $g \in \mathcal{N}_{SRH}$, the top-k individuals from each island, P_k^A and P_k^B respectively, are sent to the island G on which the SRH algorithm is run. The two sets of individuals are then combined to get P_{2k} (line 3). A new set of DJSS problem instances,$\mathcal{I}$ is sampled from $\mathcal{T}$ with the purpose of assigning new fitness values to each individual in P_{2k} (lines 4–11).

After the fitness assignment, we use the NSGA-II fitness strategies to sort the individuals. NSGA-II first ranks the individuals based on dominance relation and then the individuals with same rank are ordered based on crowding distance [3]. We use the same approach to sort the individuals in P_{2k} (line 12). After sorting, the best k individuals are extracted from P_{2k} into the list TOP_k (line 13). Then we count the number of individuals corresponding to each island in the list TOP_k (lines 14–15). The cluster corresponding to the island with the lower number of individuals in TOP_k is rejected (line 17 or 20). The $\mathcal{C}_Z$ of the winning island is further clustered into two sub-clusters (lines 18 or 21). The new clusters which are the output of SRH algorithm are then randomly assigned to the islands. Till the next invocation of SRH, the evolution in the islands is continued using DJSS problem instances sampled from the new clusters.

Migration Policies. Migration policies play a major role in the performance of the island models [13]. A migration policy states the number of individuals to be sent to the destination island, frequency of migration and the generation from which the migration starts. For the standard island model shown in Fig. 1(a) designing a policy is straightforward. Due to symmetry, a single policy for all the islands will suffice [6]. Since we consider two classes of islands which are working together for a common goal, different migration policies must be designed for different classes islands.

Formally, a policy $\mathcal{M}_{\overrightarrow{I_1,I_2}}$ from island I_1 to I_2 is defined by a triplet <start generation, frequency, #individuals to send>. We consider the migration policy

Algorithm 3. Successive Reject Heuristic

Input: $P_k^A, P_k^B, \mathcal{C}_A, \mathcal{C}_B$
Output: $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\}$ to respective islands.
$\mathcal{T} \leftarrow$ set of all DJSS training instances.
$\mathbb{I} \leftarrow$sample from $\mathcal{T}$
$P_{2K} \leftarrow \{P_k^A, P_k^B\}$
foreach $p \in P_{2k}$ **do**
  $tot.fit. \leftarrow \overrightarrow{0}$
  foreach $\mathcal{I} \in \mathbb{I}$ **do**
    $obj.values \leftarrow$ Simulation for $(p, \mathcal{I})$.
    $tot.fit. \leftarrow tot.fit. + obj.values$
  end
  $fit(p) \leftarrow tot.fit.$
end
Sort P_{2k} using NSGA-II fitness startegies.
$TOP_k \leftarrow$ Extract top-k individuals from P_{2k}.
$TOP_k^{\mathcal{A}} \leftarrow |P_k^A \cap TOP_k|$
$TOP_k^{\mathcal{B}} \leftarrow |P_k^B \cap TOP_k|$
if $TOP_k^{\mathcal{A}} \geq TOP_k^{\mathcal{B}}$ **then**
  Reject $\mathcal{C}_B$.
  $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\} \leftarrow$ K-means cluster$(\mathcal{C}_A)$
else
  Reject $\mathcal{C}_A$.
  $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\} \leftarrow$ K-means cluster$(\mathcal{C}_B)$
end

$\mathcal{M}_{\overrightarrow{I_1,I_2}}$ to be different from $\mathcal{M}_{\overrightarrow{I_2,I_1}}$. The selection of individuals for migration is based on elitism, i.e., a proportion of fittest individual(s) are chosen from the population for migration.

For island G, it is more productive to receive individuals from A and B frequently as it will improve its diversity. This is because the evolved rules in A and B are exposed to training instances which are different from G. On the other hand a high frequency of migration between A and B will homogenize the islands, making SRH less effective. Moreover, the frequency of migration in $M_{\overrightarrow{AG}}$ is much higher than $M_{\overrightarrow{GA}}$ (similar for island B) for the same reasons. The same analysis holds for determining the number of individuals to be migrated between the two. Furthermore, the migration policy $M_{\overrightarrow{AB}}$ is restricted to exchanging individuals only and immediately after invocation of SRH.

4 Experiment Design

4.1 Simulation Model

We use a DES system (Jasima) [4] to generate DJSS problem instances. The job arrival follows a Poisson process with $\lambda = 0.85$ [7]. This assumption has been

used in large number of works [2,10,11]. For every run of the simulation, the first 500 jobs are considered as warm-up and the objective values are calculated for the next 2000 jobs.

The uncertainty in processing times is simulated using the model considered in [7]. Basically for an operation $o_{j,i}$ the relationship between the processing time with uncertainty $p'_{j,i}$ and processing time without uncertainty $p_{j,i}$ is:

$$p'_{j,i} = (1 + \theta_{j,i})p_{j,i}, \theta_{j,i} \geq 0.$$

θ follows exponential distribution [7]. In Table 3, the parameter β corresponds to the scale parameter of the exponential distribution.

In order to create problem instances with varying characteristics, DJSS problem instances are generated with many combinations of the simulation parameters shown in Table 3. The combination of these four pairs of parameters can simulate 16 types of jobs. For composing a training DJSS problem instance, *3* job types are considered at a time. On counting the unique combinations of *3* job types we find a total of 816 possible configurations (combinations without repetitions). Since we extract features from problem instances in order to perform clustering we create 20 DJSS problems for each configuration to build the training set $\mathcal{T}$. Our preliminary study showed that a larger training set would show no advantage but require more computational effort.

For testing, we create a new set (say $\mathcal{Y}$) of DJSS problems using the 816 possible configurations mentioned above. We sample 30 DJSS problem instances from $\mathcal{Y}$ to obtain our first test set. Due to large number of problem configurations it is not possible to test on each of them separately. Therefore, we create *four* more test sets by clustering $\mathcal{Y}$ and sampling 30 problem instances from each. These test sets are denoted by 3-$\mathcal{Y}$, 3-I, 3-II, 3-III and 3-IV, where 3 stands for number of job types.

We also want to observe the generalization ability of our methods over more complex configurations. Therefore, DJSS instances comprising of *4* job types are created. On counting, the total number of unique configurations in this case are as high as 3876 (combinations with repetitions). Performing the same procedure described above generates the following test sets: 4-$\mathcal{Y}$, 4-I, 4-II, 4-III and 4-IV.

Table 3. DJSS simulation parameters

Simulation paramter	Values
Processing time range	[0, 49], [20, 69]
Uncertainty scale parameter (β)	$\{0.2, 0.4\}$
Due date tightness	$\{1.5, 2.5\}$
# operations per job	$\{8, 10\}$

Table 4. Migration policies

Island-pairs	Policies
$\overrightarrow{GG}$	<20, 20, 30>
$\overrightarrow{AB}$	<50, 50, 60>
$\overrightarrow{BA}$	<50, 50, 60>
$\overrightarrow{AG}$	<20, 20, 30>
$\overrightarrow{BG}$	<20, 20, 30>
$\overrightarrow{GB}$	<50, 25, 10>
$\overrightarrow{GA}$	<50, 25, 10>

Table 5. Terminal sets for GP.

Terminal set	Meaning
PT	Processing time of operation
RO	Remaining operations for job
RJ	Ready time of job
RT	Remaining processing time of job
RM	Ready time of machine
DD	Due date
W	Job weight
ERC	Ephemeral random constant

4.2 Genetic Programming System

The terminal set for genetic programming is listed in Table 5. For all our islands we use a population size of 800 each. We also compare performance of our method with the standard NSGA-II for which the population size is set at 2500. With a tree depth of 6, the crossover and mutation are 0.85 and 0.1 respectively [10]. Each evolutionary algorithm is run for 150 generations.

4.3 Island Model

We use the SRH algorithm at generations 49 and 99, i.e., $\mathcal{N}_{SRH} = \{49, 99\}$. The SRH algorithm also requires the simulator to assign fitness to individuals. Furthermore, for GPHH to utilize the problem instances from a cluster, considerable number of generations are required. So frequently invoking SRH will not yield the desired outcome but only incur additional computational cost. Therefore the size of $\mathcal{N}_{SRH}$ is small and generations selected are far apart.

The migration policies are presented in Table 4. While deciding the frequency parameter of the migration policies involving islands A and B, $\mathcal{N}_{SRH}$ has been taken into account. The exchange of individuals starts after a delay as the evolved rules in the early generations are not good. For the TOP_k individuals the value $k = 30$ was chosen, after experimental evaluation.

5 Results and Discussion

In this Section, we present the results from our experiments. We compare the performance our method with standard NSGA-II algorithm and standard island model approach. The hypervolume ratio (HV), inverted generational distance (IGD) and spread (SPREAD) indicators are considered for comparison as they are frequently used in the literature [12] to compare the generated Pareto fronts. In order to approximate the true Pareto front as required by performance indicators the individuals from all the methods across all runs are combined. For

each method the solutions are compared over 30 problem instances from a test set. 30 independent runs produce 30 sets of dispatching rules for each method. The Wilcoxon-rank-sum test is used to compare the performance. We consider a significance level of 0.05.

The results are summarized in Tables 6, 7 and 8. Each cell in the tables consists of a triplet which represents $[win - draw - lose]$. For example, in Table 6 the comparison between standard island model and NSGA-II approach is summarized. For the training set 3-$\mathcal{Y}$, if we consider hypervolume indicator, then island model has significantly outperformed NSGA-II in 18 problem instances and there is no significant difference observed for 12 problem instances.

Table 6. Island-Model versus NSGA-II

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[18-12-0]	[11-19-0]	[18-12-0]	[19-11-0]	[17-13-0]	[14-16-0]	[15-15-0]	[18-12-0]	[16-13-0]	[12-18-0]
IGD	[24-6-0]	[21-9-0]	[25-5-0]	[29-1-0]	[27-3-0]	[24-6-0]	[21-9-0]	[22-8-0]	[22-8-0]	[19-11-0]
SPREAD	[3-22-5]	[0-18-12]	[4-23-0]	[5-25-0]	[2-28-0]	[3-21-6]	[6-22-2]	[4-23-3]	[5-20-5]	[2-24-2]

In Table 6, we compare NSGA-II with standard island model. As expected, the performance of island model is much better, which is line with the observations made in [6]. For HV and IGD performance indicators, the performance is very good, but for SPREAD indicator there is no clear winner. This significant difference in performance is consistent across all the test sets including 4-job type configurations.

Table 7. SRH-Island model versus NSGA-II

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[23-7-0]	[17-3-0]	[23-7-0]	[25-5-0]	[24-6-0]	[20-10-0]	[24-6-0]	[22-8-0]	[24-6-0]	[21-9-0]
IGD	[30-0-0]	[30-0-0]	[27-3-0]	[29-1-0]	[30-0-0]	[30-0-0]	[27-3-0]	[30-0-0]	[28-2-0]	[29-1-0]
SPREAD	[1-23-6]	[0-25-5]	[1-27-2]	[3-26-1]	[0-28-2]	[4-21-5]	[1-27-2]	[1-27-2]	[2-22-6]	[0-25-5]

In Table 7, we compare the performance of NSGA-II and SRH-based island model (SRH). Across all the test sets the proposed method has done well. Particularly for HV indicator, the SRH method has significantly done better in more than 20 problem instances for almost every test set. Similar performance is observed for IGD as well. Though once gain, with respect to SPREAD, there is no verifiable difference. This is because the obtained Pareto fronts are sparse for all the algorithms.

Finally we compare, the SRH approach with the standard island model. Once again the SRH approach performs significantly better on an average of 10 problem instances from each test set. This confirms that SRH approach was able to associate useful training instances through the successive rejection of clusters of training instances.

Table 8. SRH-Island model versus island model

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[10-20-0]	[5-24-1]	[6-22-2]	[9-21-0]	[14-16-0]	[10-20-0]	[10-20-0]	[8-21-1]	[9-21-0]	[11-19-0]
IGD	[18-12-0]	[20-10-0]	[19-11-0]	[19-11-0]	[20-10-0]	[15-15-0]	[14-15-1]	[13-17-0]	[15-15-0]	[18-20-0]
SPREAD	[5-18-7]	[10-20-0]	[3-24-3]	[1-25-4]	[0-23-7]	[5-20-5]	[1-22-7]	[2-20-8]	[4-17-9]	[3-24-3]

5.1 Analysis

A frequently observed path taken by the successive reject heuristic is represented below.

$$\mathcal{T} \rightarrow \{\cancel{\mathcal{C}_1}, \mathcal{C}_2\} \rightarrow \{\mathcal{C}_{21}, \cancel{\mathcal{C}_{22}}\} \rightarrow \{\mathcal{C}_{211}, \mathcal{C}_{212}\}$$

In **retrospect**, we analyze the clusters which showed potential to guide the GPHH toward evolving better rules. In order to further validate the ability of SRH, we took the clusters represented by $\mathcal{C}_{21}$ and $\mathcal{C}_{22}$ as training sets. We performed 30 independent runs of NSGA-II algorithm on each. We observed that the cluster rejected by SRH ($\mathcal{C}_{22}$) performed significantly poor on both HV and IGD indicators. Figure 2 shows a box plot for HV indicator on a test problem instance from the set 3-$\mathcal{Y}$.

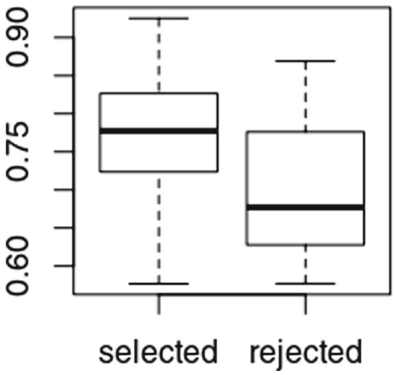


Fig. 2. Comparing $\mathcal{C}_{21}$(selected) and $\mathcal{C}_{22}$ (rejected) using HV.

Furthermore, we also analyzed the problem configurations associated with cluster $\mathcal{C}_{21}$. One of the reasons for analyzing $\mathcal{C}_{21}$ and not $\mathcal{C}_2$ is its smaller size and also the fact that out of 30 independent runs, this path was chosen by SRH for 20 of the runs. We observed that the DJSS instances whose job types were pertaining to equal proportion of high and low level of uncertainty were in high numbers. Also, DJSS instances comprising jobs with low and high number of operations per job were found in large numbers. In other words, SRH is biased towards instances with high variability in their jobs. A high variability in the training instances has more potential to present the GPHH with difficult and conflicting scenarios, as explained in a previous example.

6 Conclusions

Most of the research works in evolutionary scheduling focus on improving only the different aspects of algorithms. But it is also important to develop methods to select appropriate training instances for the evolutionary algorithms to produce desired outcome. We have successfully taken a step toward this direction by demonstrating that a simple sampling heuristic using basic features extracted from the problem instances could improve the effectiveness of the evolutionary process. By exploiting the potential of island model approach we obtained significantly better results while maintaining computational efficiency. We demonstrated the efficacy of our approach using just two objectives and in future, we would extend our work to tackle many-objective scheduling problems.

References

1. Bertels, A.R., Tauritz, D.R.: Why asynchronous parallel evolution is the future of hyper-heuristics: A CDCL SAT solver case study. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 1359–1365. ACM (2016)
2. Branke, J., Nguyen, S., Pickardt, C., Zhang, M.: Automated design of production scheduling heuristics: a review. IEEE Trans. Evol. Comput. **20**, 110–124 (2016)
3. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
4. Hildebrandt, T.: Jasima - an efficient java simulator for manufacturing and logistics. Last Accessed **16** (2012)
5. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evol. Comput. **23**(3), 343–367 (2015)
6. Karunakaran, D., Chen, G., Zhang, M.: Parallel multi-objective job shop scheduling using genetic programming. In: Ray, T., Sarker, R., Li, X. (eds.) ACALCI 2016. LNCS (LNAI), vol. 9592, pp. 234–245. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28270-1_20
7. Karunakaran, D., Mei, Y., Chen, G., Zhang, M.: Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 282–289. ACM (2017)
8. Kouvelis, P., Yu, G.: Robust Discrete Optimization and its Applications, vol. 14. Springer, Heidelberg (2013). https://doi.org/10.1007/978-1-4757-2620-6
9. Lawrence, S.R., Sewell, E.C.: Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. J. Oper. Manag. **15**(1), 71–82 (1997)
10. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. **3**(1), 41–66 (2017)
11. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Dynamic multi-objective job shop scheduling: a genetic programming approach. In: Uyar, A., Ozcan, E., Urquhart, N. (eds.) Automated Scheduling and Planning. SCI, vol. 505, pp. 251–282. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39304-4_10
12. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. IEEE Trans. Evol. Comput. **18**(2), 193–208 (2014)

13. Nowak, K., Izzo, D., Hennes, D.: Injection, saturation and feedback in metaheuristic interactions. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 1167–1174. ACM (2015)
14. Sudholt, D.: Parallel evolutionary algorithms. In: Kacprzyk, J., Pedrycz, W. (eds.) Springer Handbook of Computational Intelligence, pp. 929–959. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-43505-2_46
15. Xiao, N., Armstrong, M.P.: A specialized island model and its application in multiobjective optimization. In: Cantú-Paz, E. (ed.) GECCO 2003. LNCS, vol. 2724, pp. 1530–1540. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45110-2_24

Sensitivity of Parameter Control Mechanisms with Respect to Their Initialization

Carola Doerr[1(✉)] and Markus Wagner[2]

[1] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, 75005 Paris, France
carola.doerr@lip6.fr
[2] Optimisation and Logistics, University of Adelaide, Adelaide, SA 5005, Australia

Abstract. The parameter setting problem constitutes one of the major challenges in evolutionary computation, and is subject to considerable research efforts. Since the optimal parameter values can change during the optimization process, efficient parameter control techniques that automatically identify and track reasonable parameter values are sought.

A potential drawback of dynamic parameter selection is that state-of-the-art control mechanisms introduces themselves new sets of *hyper-parameters*, which need to be tuned for the problem at hand. The general hope is that the performance of an algorithm is much less sensitive with respect to these hyper-parameters than with respect to its original parameters. This belief is backed up by a number of empirical and theoretical results. What is less understood in discrete black-box optimization, however, is the influence of the initial parameter value. We contribute with this work an empirical sensitivity analysis for three selected algorithms with self-adjusting parameter choices: the $(1+1)$ EA_α, the 2-rate $(1+\lambda)$ $\text{EA}_{2r,r/2}$, and the $(1+(\lambda,\lambda))$ GA. In all three cases we observe fast convergence of the parameters towards their optimal choices. The performance loss of a sub-optimal initialization is shown to be almost negligible for the former two algorithms. For the $(1+(\lambda,\lambda))$ GA, in contrast, the choice of λ is more critical; our results suggest to initialize it by a small value.

Keywords: Parameter control · Evolutionary algorithms
Discrete black-box optimization · Initialization

1 Introduction

Every evolutionary algorithm (EA) and, more generally, every discrete black-box optimization heuristic, comes with a set of (explicit or implicit) parameters that needs to be set in order to run it. Among the most influential parameters are the population sizes, the mutation rates, the crossover probabilities, and the selective pressure. The choice of any of these parameters can have a significant

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 360–372, 2018.
https://doi.org/10.1007/978-3-319-99259-4_29

impact on the performance of the EA under consideration. It is therefore not surprising that the parameter selection question has evolved into an important research stream within the evolutionary computation community, cf. [17] for detailed discussions.

The last forty years of research on the parameter setting problem have contributed to a significant gain in performance, and have been a major building block for the success of evolutionary computation methods. According to Eiben, Hinterding, and Michalewicz [12] the parameter setting literature can be classified into two main research streams:

- *Parameter tuning* addresses the question how to efficiently identify good parameter values through an initial set of experiments. After their identification, these parameter values are not further adjusted during the optimization process, but remain fixed instead. Among the most-widely applied tools for parameter tuning are irace [18], SPOT [3], SMAC [14], ParamILS [15], and GGA [2].
- *Parameter control,* in contrast, studies ways to adjust ("control") the parameter values during the run of the optimization, to benefit from an adaptation to the different stages of the optimization process. Using such non-static parameter values, the EAs can, for example, evolve from a rather exploratory, globally acting heuristic to a more and more locally exploiting one. Among the best-known parameter control techniques are the step size and covariance matrix adaptation in the CMA-ES [13] and variants of the 1/5-th success rule [5,19,20].

The focus of our work is on parameter control for discrete black-box optimization, a topic that has been somewhat neglected in the evolutionary computation community, as confirmed by a quote of [16, Sect. 8], which says that "controlling EA parameters on-the-fly is still a rather esoteric option". A potential reason for this situation may be the common critique that parameter control mechanisms add yet another level of complexity to the algorithms.

The influence of the parameter control mechanisms are indeed difficult to grasp analytically, so that only few theoretical works addressing the parameter control question exist [7]. A related critique of parameter control is the fact that on-the-fly parameter selection techniques come with their own *hyper-parameters*, which need to be set to determine the exact update rules. From a high-level perspective one may feel that not much can be gained by replacing a parameter by one or more hyper-parameters, but the general hope is that the influence of these hyper-parameters is much less important than that of the original parameter values. Several studies confirm this hope for some specific settings, empirically as well as in rigorous mathematical terms, cf. the surveys [1,7,12,16] and references therein.

Our Contribution

Complementing our recent work on the sensitivity of parameter control mechanisms with respect to the hyper-parameters that determine the update

strength [11], we consider in this study their sensitivity with respect to initialization. More precisely, we analyze for three different EAs with self-adjusting parameter selection the influence of the initial parameter value on the performance: the $(1+1)$ EA_α proposed in [11], the 2-rate $(1+\lambda)$ $\text{EA}_{2r,r/2}$ from [10], and the $(1+(\lambda,\lambda))$ GA proposed in [8] and analyzed in [6]. In the first two algorithms the mutation rate is controlled by a success-based update rule. In the $(1+(\lambda,\lambda))$ GA the adaptation of λ influences the offspring population sizes, the mutation rate, and the crossover bias, cf. Sect. 3. For all three algorithms we test the influence of extreme initialization, i.e., $p = 1/n$ vs. $p = 1/2$ for the $(1+1)$ EA_α and the $(1+\lambda)$ $\text{EA}_{2r,r/2}$, and $\lambda = 1$ vs. $\lambda = n$ for the $(1+(\lambda,\lambda))$ GA.

Our selection of algorithms is clearly theory-biased, i.e., we favor those algorithms for which mathematical analyses of their running time behavior are available. This allows us to chose update mechanisms which are known to be (close to) optimal, so that our sensitivity analysis of the initial parameter values is not biased by a non-sensible choice of hyper-parameters.

Our testbed comprises of the well-known OneMax and LeadingOnes benchmark functions, again with the motivation to not bias the result by a non-suitable control mechanism, and to allow for a comparison with known optimal dynamic parameter values. The OneMax problem is the problem of maximizing a function of the type $\text{OM}_z : \{0,1\}^n \to [0..n], x \mapsto |\{i \in [n] \mid x_i = z_i\}|$, where the target string $z \in \{0,1\}^n$ is of course unknown to the algorithm. OneMax is a separable function, and thus easily solved in expected time $\Theta(n \log n)$ by a large range of standard EAs. LeadingOnes, in contrast, is non-separable, and requires a quadratic number of function evaluations, on average, by standard EAs. The LeadingOnes problem is the problem of optimizing functions of the type $\text{LO}_{z,\sigma} : \{0,1\}^n \to \mathbb{N}, x \mapsto \max\{i \in [0..n] \mid \forall j \in [i] : x_{\sigma(j)} = z_{\sigma(j)}\}$, where z is a length-n bit string and $\sigma : [n] \to [n]$ a permutation of the index set $[n]$. We acknowledge that these simplified benchmark problems may not be very representative for real-world optimization challenges. In accordance with [21] we nevertheless believe that they can test several important features of reasonable parameter control mechanisms, and give first indications into which update schemes to favor under which conditions.

Our results indicate quite stable performances for the $(1+1)$ EA_α and the $(1+\lambda)$ $\text{EA}_{2r,r/2}$. Even when initialized with extreme mutation rates, the dynamic choice very quickly converges to optimal mutation rates and the incurred performance loss of a sub-optimal initialization is small. The situation is different for the $(1+(\lambda,\lambda))$ GA. The number of function evaluations per iteration grows linearly with the value of λ (more precisely, up to 2λ offspring are evaluated per iteration), a cost that the additional drift towards the optimum cannot compensate for. This situation of a too large λ value does not last very long, as we observe again fast convergence of the parameter towards its optimal (dynamic) setting. It nevertheless causes significant and non-negligible performance losses: for the 1000-dimensional OneMax problem, for example, the worst initialization $\lambda = n$ yields a performance that is around 69% worse compared to that of the optimal initial parameter choice $\lambda = 1$.

2 Sensitivity Analysis for the $(1+1)$ EA$_\alpha$

In [11] we have presented a $(1+1)$ EA variant with success-based multiplicative mutation rate updates, the $(1+1)$ EA$_\alpha$. This algorithm starts the optimization process with a random initial solution and an initial mutation rate $p = p_0 \in (0, 1/2]$. In every iteration one new solution candidate is created from the current-best solution through a conditional standard bit mutation with mutation rate p. The condition requires that at least one bit is changed, to avoid useless function evaluations. In practice, this conditional mutation operator can be implemented by first sampling a number ℓ from the conditional binomial distribution $\mathrm{Bin}_{>0}(n,p)$ and then choosing uniformly at random and without replacement the ℓ positions in which the bits are flipped. If the so-created offspring is at least as good as its predecessor, it replaces the latter. In this case the mutation rate is increased to $\min\{Ap, 1/2\}$, where $A > 1$ is a constant that remains fixed during the execution of the algorithm. If, on the other hand, the offspring is strictly worse than its parent, it is discarded and the mutation rate decreased to $\max\{bp, 1/n^2\}$, where $0 < b < 1$ is another constant.

Altogether, the (1+1) EA$_\alpha$ has three *hyper-parameters*: the update strengths A and b as well as the initial mutation rate p_0. It was demonstrated in [11] that the $(1+1)$ EA$_\alpha(A,b,1/n)$ performs very well on the classic benchmark functions OneMax and LeadingOnes for a broad choice of values for A and b. For example, in 78% of all tested combinations of $A \in (1, 2.5]$ and $b \in [0.4, 1)$ the $(1+1)$ EA$_\alpha(A,b,1/n)$ achieved a better average running time than Randomized Local Search (RLS) on the 250-dimensional LeadingOnes function. About 90% of these configurations outperform the (1+1) EA$_{>0}$ (which is the $(1+1)$ EA$_\alpha(1,1,1/n)$) on the 1000-dimensional OneMax function. In this section we analyze how sensitive this performance is with respect to the choice of the initial mutation rate p_0.

2.1 Optimal Mutation Rates for OneMax and LeadingOnes

Before we present our empirical findings, we summarize in this section what is known about the optimal mutation rates for OneMax and LeadingOnes.

OneMax. In [9] it was shown that the RLS variant flipping in every step the number of bits that maximizes the expected progress cannot be significantly worse than the best unary unbiased algorithm, which is the one minimizing in every step the expected remaining running time. Denoting by $k_{\mathrm{opt,OM}}(n, \mathrm{OM}(x))$ the choice that maximizes the expected OM-progress $\mathbb{E}\Big[\max\{\mathrm{OM}(\mathrm{mut}_\ell(x)) - \mathrm{OM}(x), 0\}\Big] := \sum_{i=\lceil \ell/2 \rceil}^{\ell} \frac{\binom{n-\mathrm{OM}(x)}{i}\binom{\mathrm{OM}(x)}{\ell-i}(2i-\ell)}{\binom{n}{\ell}}$ of flipping ℓ bits in bit string x, the following is known. $k_{\mathrm{opt,OM}}(n, \mathrm{OM}(x))$ decreases monotonically with increasing function value $\mathrm{OM}(x)$; it is $n/2$ for $\mathrm{OM}(x) = n/2$ and equal to 1 for all x with $\mathrm{OM}(x) \geq 2n/3$.

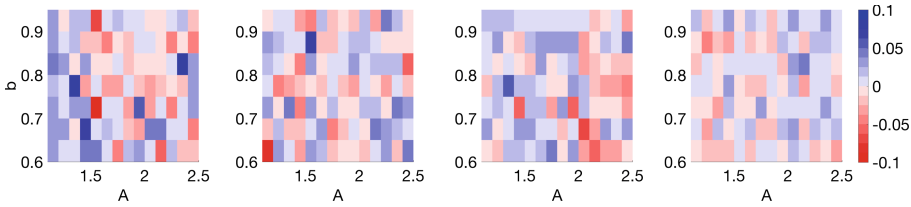
The expected OneMax value of a random initial solution x is $n/2$ for OneMax, and with high probability $\mathrm{OM}(x)$ lies in the interval $[n/2 \pm \sqrt{n}]$. The exact average optimal mutation strength is $\sum_{i=1}^{n} \mathbb{P}[\mathrm{OM}(x) = i] k_{\mathrm{opt,OM}}(n, i)$. We do not have any closed form for the drift maximizing value $k_{\mathrm{opt,OM}}(n, i)$, but we can evaluate this expression numerically. For $n = 1000$ the sum evaluates to 500.0252, which is very close to $n/2$.

LeadingOnes. For LeadingOnes the situation is much better understood. The optimal mutation rate of the classic (non-resampling) $(1 + 1)$ EA is $1/(\mathrm{LO}(x) + 1)$ [4] and the optimal number of bits to flip is $k_{\mathrm{opt,LO}}(n, \mathrm{LO}(x)) := \lfloor n/(\mathrm{LO}(x)+1) \rfloor$ [11, Lemma 1]. The average optimal number of bits to flip is thus $\sum_{i=0}^{n} k_{\mathrm{opt,LO}}(n, i)\mathbb{P}[\mathrm{LO}(x^{\mathrm{u.a.r.}}) = i] = \sum_{i=0}^{n} \lfloor n/(i + 1) \rfloor 2^{-(i+1)}$. For $n = 100$ (250, 1000) this value is around 69 (173, 693).

2.2 Evaluating the Relative Average Improvement

In light of the discussion in Sect. 2.1, one might wonder if significant gains are possible for the $(1 + 1)$ EA_α when the mutation rate is initialized as $p_0 = 1/2$ instead of $p_0 = 1/n$. As a first step towards analyzing the sensitivity of the $(1 + 1)$ EA_α with respect to this initialization, we compute for each of the 120 configurations with $A \in \{1.1, 1.2, \ldots, 2.5\}$ and $b \in \{0.6, 0.65, \ldots, 0.95\}$ the average optimization time of 101 independent runs of the $(1 + 1)$ $\mathrm{EA}_\alpha(A, b, 1/2)$. We compare this average value to that of the same configuration (A, b) for $p_0 = 1/n$, and we compute the relative gain of the $p_0 = 1/2$ initialization. That is, denoting by $T(A, b, p_0)$ the average optimization time of the $(1+1)$ $\mathrm{EA}_\alpha(A, b)$ with initialization p_0, we calculate for each configuration (A, b) the value $(T(A, b, 1/n) - T(A, b, 1/2))/T(A, b, 1/n)$. This data is displayed in the heatmaps of Fig. 1 for the 1000- and 1500-dimensional OneMax problem and the 100- and 250-dimensional LeadingOnes problem, respectively.

We observe that the data is rather unstructured, and that a good relative gain in one dimension does typically not apply to the other. The relative gains range from a negative -10% (-8%) to a positive 8% (7%) improvement for OneMax of dimension $n = 1000$ ($n = 1500$), and from -7% (-4%) to 5% (4%) for the 100-(250-)dimensional LeadingOnes problem. Note that here the relatively low number of repetitions has to be taken into account. The average gain of the $p_0 = 1/2$ initialization over the $p_0 = 1$ initialization in all 120 (A, b) configurations is about 0.17% (0.21%) for the OneMax problem of dimension $n = 1000$ ($n = 1500$) and is about -0.13% (-0.05%) for LeadingOnes in dimension $n = 100$ ($n = 250$). These small values indicate that the influence of the initial parameter value is not very important. It may be surprising that the average gain is negative for the LeadingOnes problem, but we suspect that this is an effect of the problem size, which may vanish in larger dimension.



(a) OM $n = 1000$ (b) OM $n = 1500$ (c) LO $n = 100$ (d) LO $n = 250$

Fig. 1. Relative difference $(T(A, b, 1/n) - T(A, b, 1/2))/T(A, b, 1/n)$ of the average running time for 120 configurations of the $(1 + 1)$ EA_α with $1 < A \leq 2.5$ and $0.6 \leq b < 1$

Table 1. Average running times of the $(1 + 1)$ $\text{EA}_\alpha(A, b, p_0)$ on OneMax for 1001 independent repetitions and results of the one-sided Wilcoxon rank-sum tests for the null hypothesis that $T(A, b, 1/2) < T(A, b, 1/n)$.

n	A	b	$T(A, b, 1/n)$	$T(A, b, 1/2)$	$(T_{1/n} - T_{1/2})/T_{1/n}$	$p(1/2 < 1/n)$
500	1.11	0.66	3,045	3,019	0.9%	0.096
500	1.2	0.85	3,063	2,994	2.3%	**0.028**
500	1.3	0.75	3,039	2,998	1.3%	0.092
500	2	0.5	3,035	2,980	1.8%	**0.005**
1000	1.11	0.66	6,780	6,788	−0.1%	0.231
1000	1.2	0.85	6,787	6,645	2.1%	**0.009**
1000	1.3	0.75	6,802	6,595	3.0%	**0.001**
1000	2	0.5	6,752	6,682	1.0%	0.086
2000	1.11	0.66	14,962	14,895	0.4%	0.112
2000	1.2	0.85	14,834	14,854	−0.1%	0.478
2000	1.3	0.75	14,839	14,768	0.5%	0.369
2000	2	0.5	15,297	15,133	1.1%	0.238

2.3 Testing for Statistical Significance

While the results displayed in the heatmaps do not suggest that we should expect *important* performance gains from a better initialization, this data does not answer the question whether the (dis-)advantages are *statistically significant*. We therefore investigate a few selected configurations in more detail, and use the Wilcoxon rank-sum tests to test for significance. Precisely, we run each of the four selected configurations $(A = 1.2, b = 0.85)$, $(1.3, 0.75)$, $(2.0, 0.5)$, and $(1.11, 0.66)$ investigated in [11] 1001 independent times on the OneMax problem of dimension $n \in \{500, 1000, 2000\}$ and on the LeadingOnes problem of dimensions $n \in \{100, 250, 500\}$. For each (function, dimension, configuration) triple we test whether there is a significant difference between the optimization times of the $(1 + 1)$ $\text{EA}_\alpha(A, b, 1/2)$ and the $(1 + 1)$ $\text{EA}_\alpha(A, b, 1/n)$. The results are summarized in Tables 1 and 2. The reported p-values are for

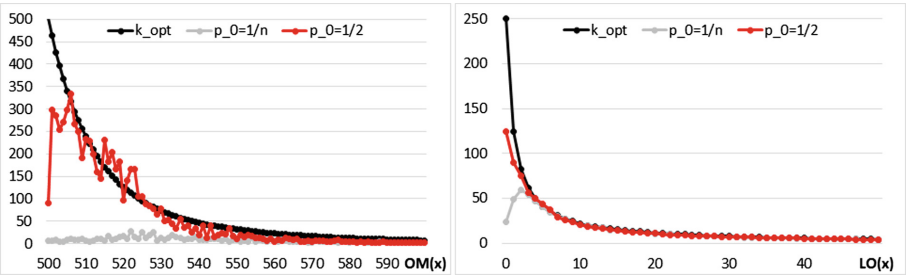
Table 2. Average running times of the $(1+1)$ $\text{EA}_\alpha(A, b, p_0)$ on LeadingOnes for 1001 independent repetitions and results of the one-sided Wilcoxon rank-sum tests for the null hypothesis that $T(A, b, 1/2) < T(A, b, 1/n)$.

n	A	b	$T(A, b, 1/n)$	$T(A, b, 1/2)$	$(T_{1/n} - T_{1/2})/T_{1/n}$	$p(1/2 < 1/n)$
100	1.11	0.66	4,493	4,508	−0.3%	0.602
100	1.2	0.85	4,125	4,105	0.5%	0.183
100	1.3	0.75	4,141	4,144	−0.1%	0.574
100	2	0.5	4,182	4,245	−1.5%	**0.954**
250	1.11	0.66	28,348	28,130	0.8%	0.081
250	1.2	0.85	25,386	25,513	−0.5%	0.708
250	1.3	0.75	25,720	25,954	−0.9%	0.884
250	2	0.5	26,142	26,302	−0.6%	0.796
500	1.11	0.66	112,583	113,135	−0.5%	0.882
500	1.2	0.85	102,018	101,605	0.4%	0.082
500	1.3	0.75	102,862	102,903	0.0%	0.528
500	2	0.5	105,329	105,129	0.2%	0.375

the test "$T(A, b, 1/2) < T(A, b, 1/n)$?"; i.e., small p-values indicate a strong support for the null hypothesis that the running time distribution of the $(1 + 1)$ $\text{EA}_\alpha(A, b, 1/2)$ is dominated by that of the $(1 + 1)$ $\text{EA}_\alpha(A, b, 1/n)$. Put differently, a small p-value is a strong evidence for the hypothesis that the $(1 + 1)$ $\text{EA}_\alpha(A, b, 1/2)$ is faster than the $(1 + 1)$ $\text{EA}_\alpha(A, b, 1/n)$. We recall that the result of the Wilcoxon rank-sum test for the other one-sided null hypothesis (i.e., the hypothesis that $T(A, b, 1/2) > T(A, b, 1/n)$) is $1 - p$. We highlight in Tables 1 and 2 p-values that are smaller than 5 % or larger than 95 %.

We observe that for OneMax the p-values for the one-sided Wilcoxon rank-sum test are smaller than 0.5 for all tested configurations and problem dimensions, indicating that, if at all, there is a bias supporting the claim that the $(1+1)$ $\text{EA}_\alpha(A, b, 1/2)$ is faster than the $(1+1)$ $\text{EA}_\alpha(A, b, 1/n)$. For three of the four configurations the p-values are much larger for problem dimension $n = 2000$ than for the smaller dimensions. For the configuration $(A = 1.11, b = 0.66)$, which corresponds to the 1/5-th success rule, the p-value is largest for $n = 1000$. We do not have an explanation for this, but did not investigate further as the value does not indicate a statistically significant difference.

For LeadingOnes, the situation is different. Some p-values are rather large, and one value even larger then 95%, which suggests that in this setting the initialization with $p_0 = 1/\text{n}$ may be more suitable than the initialization $p_0 = 1/2$. We recall, however, from Sect. 2.1 that the average optimal initial value is rather around 69/100. Note also that the absolute and relative differences in the running times are all very small.



(a) OneMax $n = 1000$ (b) LeadingOnes $n = 250$

Fig. 2. Average number of bit flips of the $(1+1)$ EA$_\alpha(A = 2, b = 0.5, p_0)$ in iterations starting with a parent individual of fitness $f(x)$

2.4 Visualizing the Mutation Rate Adaptation

Finally, we visualize the evolution of the mutation rate. To this end, we have tracked for 100 independent runs the number of bits that have been flipped in each iteration, along with the function value of the corresponding parent. From this data we compute the average number of bit flips per function value. These averages are plotted against the optimal mutation strengths $k_{\text{opt},f}(n, f(x))$ described in Sect. 2.1. Figure 2 summarizes this data. Note that we zoom in both plots into the interesting initial part of the optimization process.

We observe that the curves for $p_0 = 1/2$ have a better fit with k_{opt} than those for $p_0 = 1/n$. We also see that for the 1000-dimensional OneMax problem it is around $\text{Om}(x) = 560$ that the two curves converge. They are indistinguishable thereafter since the underlying adaptation rule is the same. For LeadingOnes the two curves do not differ by more than one for all $\text{Lo}(x)$-values greater than 11.

3 Sensitivity of the Self-adjusting $(1 + (\lambda, \lambda))$ GA

We also test the relevance of the initial parameter value for the self-adjusting $(1+(\lambda, \lambda))$ GA [6,8]. It stores in the memory a current-best solution, creates from it λ offspring by mutation, and another λ offspring by a biased recombination of the best of the mutated offspring with its parent. The best recombined offspring replaces the parent individual if its function value is at least as good.

Using the recommended parametrization $p = \lambda/n$ and $c = 1/\lambda$ for the mutation rate and the crossover bias, respectively, the only parameter of the $(1+(\lambda, \lambda))$ GA becomes the population size λ. In [8] the following multiplicative update rule was suggested to control λ: If an iteration was successful, i.e., if at the end of the iteration we have identified a strictly better search point, we decrease λ to λ/F. We increase λ to $\lambda F^{1/4}$ otherwise. According to experiments reported in [8] the influence of the update strength F is not very pronounced. In

Table 3. Results for the self-adjusting $(1 + (\lambda, \lambda))$ GA with different initialization. Nearly all differences are statistically significant.

n	λ_0	T	KW test	$p(1 < \ln n)$	$p(1 < n)$	$p(\ln n < n)$
500	1	3,293	**0**	0.178	**0**	**0**
500	$\ln n$	3,309				
500	n	5,562				
1000	1	6,715	**0**	**0.004**	**0**	**0**
1000	$\ln n$	6,678				
1000	n	11,366				
2000	1	13,716	**2.29E−155**	0.556	**2.49E−105**	**9.11E−106**
2000	$\ln n$	13,736				
2000	n	18,357				

line with common implementations of the 1/5-th success rule and the recommendations given in [6,8], we set F equal to 3/2. The self-adjusting $(1 + (\lambda, \lambda))$ GA achieves a linear expected running time on OneMax; this is asymptotically optimal among all possible parameter settings, and strictly better than what any static parameter choice can achieve [6].

We note that as in the (1+1) EA_α, and unlike the experiments reported in [8], we enforce that at least one bit is flipped in the mutation phase, by sampling the mutation strength from $\text{Bin}_{>0}(n, \lambda/n)$ instead of $\text{Bin}(n, \lambda/n)$. In addition, we evaluate a recombined offspring only if it is different from both of its parents. This can be tested efficiently and avoids useless function evaluations.

To test the influence of the initialization of λ, we perform 1001 runs of the algorithm on OneMax instances of dimension $n \in \{500, 1000, 2000\}$ with three different initialization rules: $\lambda_0 = 1$, $\lambda_0 = \ln n$, and $\lambda_0 = n$.

As already mentioned and explained in Sect. 1 the average optimization times vary drastically. To test for statistical significance, we first employ the Kruskal-Wallis test, which is an extension of the Wilcoxon rank-sum test for more than two data sets.[1] The outcomes of the Kruskal-Wallis test of zero (or effectively zero) provide strong evidence that the outcomes are not identically distributed. This is confirmed by the pairwise Wilcoxon rank-sum tests, whose values are also reported in Table 3.

To visualize the adaptation of λ, we plot in Fig. 3 its evolution in dependence of the $\text{OM}(x)$-value against the asymptotically optimal choice of $\lambda_{\text{opt}} = \lceil \sqrt{n/(n - \text{OM}(x))} \rceil$ [8] for the $n = 1000$-dimensional OneMax instance. The reported values are averages of 100 independent runs. In the middle range $650 < \text{OM}(x) < 850$ the average parameter values are all very close to the optimal ones. We therefore plot only the averages for the beginning of the optimization process, $n/2 = 500 < \text{OM}(x) \leq 650$, and its end, $850 \leq \text{OM}(x) \leq n = 1000$,

[1] We remark that a one-way ANOVA is not applicable as the Shapiro-Wilk normality test returns that the data is not normally distributed.

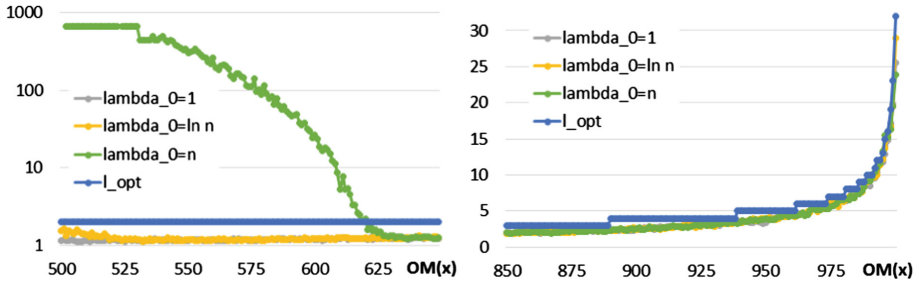


Fig. 3. Average value of λ per $\textsc{Om}(x)$-value for the self-adjusting $(1 + (\lambda, \lambda))$ GA with update strength $F = 3/2$ and different initial parameter values λ_0 on the 1000-dimensional OneMax problem. Note that the left figure uses a log scale.

respectively. We observe that the curves for $\lambda_0 = 1$ and $\lambda_0 = \ln n$ are indistinguishable for $\textsc{Om}(x) > 525$, while all three curves become indistinguishable for values $\textsc{Om}(x) > 624$. We also see that $\lambda = 1$ seems to suffice for this initial part, whereas the asymptotically optimal bound from above suggests to use $\lambda = 2$. In line with the empirical observations made in [6,8] we also see that all curves track the increase of the optimal λ-value towards the end of the optimization process very well.

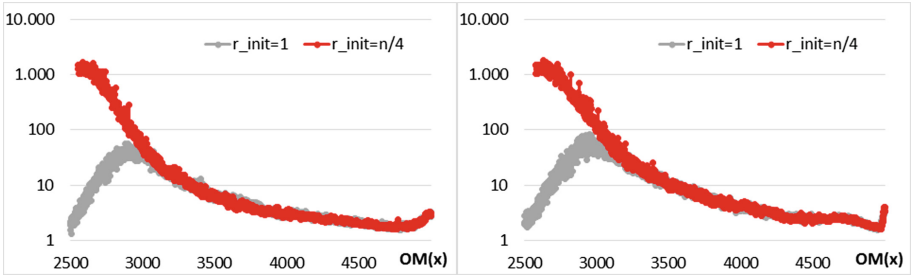
4 Sensitivity of the $(1 + \lambda)$ $\text{EA}_{r/2,2r}$

In [10] a theoretical analysis of the $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ has been presented for the OneMax problem. The $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ stores a parameter r and creates in every iteration half of the offspring by standard bit mutation with mutation rate $r/(2n)$, while the other offspring are created with mutation rate $2r/n$. At the end of the iteration the value of r is updated as follows. With probability $1/2$ it is replaced randomly by either $r/2$ or $2r$ and with the remaining $1/2$ probability it is set to the value that the winning individual of the last iteration has been created with. Finally, the value r is capped to remain in the interval $[1, n/4]$. As in previous sections, we implement this algorithm with the conditional standard bit mutation that enforces to flip at least one bit.

For the $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ we test two different initializations: $r_0 = 1$ and $r_0 = n/4$. Because of an efficient implementation, which samples waiting times instead of actually running the problem on the OneMax function, we can test the influence of these initial values for the $(1+\lambda)$ $\text{EA}_{r/2,2r}$ on OneMax instances of much larger dimensions $n = 5000$ and $n = 50\,000$. We perform tests for different values of λ: $\lambda = 100$, $\lambda = 500$, and $\lambda = 1000$. The results are summarized in Table 4. Note here that in contrast to all results presented above we report the average number of *generations* until an optimal solution has been evaluated for the first time, not the number of function evaluations. To obtain the latter, the $G(r)$-values need to be multiplied by λ.

Table 4. Results for the average of 1001 runs of the $(1+\lambda)$ $\text{EA}_{r/2,2r}$ on OneMax

n	λ	$G(r=1)$	$G(r=n/4)$	$(G_{n/4}-G_1)/G_1$	$p(1>n/4)$
5000	100	2,234	2,217	−0.74%	0.1144
5000	500	1,056	1,037	−1.73%	**1.97E−22**
5000	1000	852	834	−2.04%	**3.16E−10**
50000	100	63,627	62,666	−1.51%	0.6737
50000	500	65,139	65,722	0.90%	0.6833
50000	1000	62,814	61,567	−1.99%	0.2120



(a) $\lambda = 100$ (b) $\lambda = 500$

Fig. 4. Average value of r per $\text{OM}(x)$-value for the $(1+\lambda)$ $\text{EA}_{r/2,2r}$ on the 5000-dimensional OneMax problem

The Wilcoxon rank-sum single-sided test for $G(A,b,r=1) < G(A,b,r=n/4)$ shows a small but significant difference between the two distributions when $n = 5000$ for the two larger values of λ. For $n = 50\,000$, however, the difference is not significant.

We plot again the evolution of the r-values in Fig. 4 and observe that the curves are quite similar for the two settings.

5 Conclusions and Future Work

We have analyzed the influence of the initialization of success-based multiplicative update schemes on the performance of three different evolutionary algorithms. For all tested settings, we could observe that the parameter values converge very quickly, even if initialized in their extreme points. The different initialization could nevertheless lead to statistically significant performance gaps. In the case of the $(1+1)$ EA_α and the $(1+\lambda)$ $\text{EA}_{r/2,2r}$ the relative performance losses of non-optimal initial parameter values are, however, rather small. In the case of the $(1+(\lambda,\lambda))$ GA, however, the performance loss could be as large as 69%, suggesting that more care needs to be taken when controlling population sizes.

Extending our results to more complex combinatorial optimization problems could be a reasonable next step towards the long-term goal of developing a better understanding of which parameter control schemes to use under which conditions.

Acknowledgments. We would like to thank Eduardo Carvalho Pinto and Christian Giessen for providing their implementations of the $(1+1)$ EA_α and the $(1+(\lambda,\lambda))$ GA and the $(1+\lambda)$ $\text{EA}_{r/2,2r}$, respectively.

Our work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH and by the Australian Research Council project DE160100850.

References

1. Aleti, A., Moser, I.: A systematic literature review of adaptive parameter control methods for evolutionary algorithms. ACM Comput. Surv. **49**, 56:1–56:35 (2016)
2. Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., Tierney, K.: Model-based genetic algorithms for algorithm configuration. In: IJCAI 2015, pp. 733–739. AAAI Press (2015)
3. Bartz-Beielstein, T.: SPOT: an R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. CoRR abs/1006.4645 (2010). http://arxiv.org/abs/1006.4645
4. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the leadingones problem. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN 2010. LNCS, vol. 6238, pp. 1–10. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_1
5. Devroye, L.: The compound random search. Ph.D. dissertation, Purdue University, West Lafayette, IN (1972)
6. Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1+(\lambda,\lambda))$ genetic algorithm. Algorithmica **80**, 1658–1709 (2018)
7. Doerr, B., Doerr, C.: Theory of parameter control mechanisms for discrete black-box optimization: provable performance gains through dynamic parameter choices. In: Doerr, B., Neumann, F. (eds.) Theory of Randomized Search Heuristics in Discrete Search Spaces. Springer, Cham (2018, to appear). https://arxiv.org/abs/1804.05650
8. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theor. Comput. Sci. **567**, 87–104 (2015)
9. Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. In: GECCO 2016, pp. 1123–1130. ACM (2016)
10. Doerr, B., Gießen, C., Witt, C., Yang, J.: The $(1+\lambda)$ evolutionary algorithm with self-adjusting mutation rate. In: GECCO 2017, pp. 1351–1358. ACM (2017)
11. Doerr, C., Wagner, M.: On the effectiveness of simple success-based parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In: GECCO 2018. ACM (2018, to appear). https://arxiv.org/abs/1803.01425
12. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. **3**, 124–141 (1999)
13. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**, 159–195 (2001)

14. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
15. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
16. Karafotias, G., Hoogendoorn, M., Eiben, A.: Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans. Evol. Comput. **19**, 167–187 (2015)
17. Lobo, F.J., Lima, C.F., Michalewicz, Z. (eds.): Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence, vol. 54. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-69432-8
18. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
19. Rechenberg, I.: Evolutionsstrategie. Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart (1973)
20. Schumer, M.A., Steiglitz, K.: Adaptive step size random search. IEEE Trans. Autom. Control **13**, 270–276 (1968)
21. Thierens, D.: On benchmark properties for adaptive operator selection. In: Companion Material GECCO 2009, pp. 2217–2218. ACM (2009)

Tailoring Instances of the 1D Bin Packing Problem for Assessing Strengths and Weaknesses of Its Solvers

Ivan Amaya[1(✉)], José Carlos Ortiz-Bayliss[1], Santiago Enrique Conant-Pablos[1], Hugo Terashima-Marín[1], and Carlos A. Coello Coello[2]

[1] School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey, Mexico
{iamaya2,jcobayliss,sconant,terashima}@itesm.mx
[2] CINVESTAV-IPN (Evolutionary Computation Group), Mexico City, Mexico
ccoello@cs.cinvestav.mx

Abstract. Solvers for different combinatorial optimization problems have evolved throughout the years. These can range from simple strategies such as basic heuristics, to advanced models such as metaheuristics and hyper-heuristics. Even so, the set of benchmark instances has remained almost unaltered. Thus, any analysis of solvers has been limited to assessing their performance under those scenarios. Even if this has been fruitful, we deem necessary to provide a tool that allows for a better study of each available solver. Because of that, in this paper we present a tool for assessing the strengths and weaknesses of different solvers, by tailoring a set of instances for each of them. We propose an evolutionary-based model and test our idea on four different basic heuristics for the 1D bin packing problem. This, however, does not limit the scope of our proposal, since it can be used in other domains and for other solvers with few changes. By pursuing an in-depth study of such tailored instances, more relevant knowledge about each solver can be derived.

Keywords: 1D bin packing problem · Genetic algorithm
Instance generation

1 Introduction

The Bin Packing Problem (BPP) has been widely studied in the literature [2,3]. Moreover, different kinds of optimization problems can be modelled as BPPs [7]. This has led to a broad array of different solvers for tackling the problem: from basic low-level heuristics, going through metaheuristics, and even arriving to high-level solvers known as hyper-heuristics [17]. Even so, instances have

The authors would like to thank CONACyT for the support given through projects No. 241461 and 221551. They would also like to acknowledge the support from the Research Group with Strategic Focus in Intelligent Systems, from Tecnológico de Monterrey.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 373–384, 2018.
https://doi.org/10.1007/978-3-319-99259-4_30

remained mostly stable as solvers have evolved throughout the years. Tailoring instances to solvers is thus, needed, so that both of them can evolve in tandem.

There has been an effort for creating instances in different fields. For example, Martello and Toth [11], introduced a set of instances for the binary Knapsack Problem (0/1 KP) with different numbers of items, and a high correlation between weights and profits of the items. Similarly, Zitzler et al. [19] generated data for the multi-objective KP using random integers for weights and profits that has been widely used [6,9,18]. Another example is the OR-Library, which distributes test data for Operations Research (OR) problems [1]. Yet another example corresponds to the Mixed Integer Programming Library (MIPLIB) [7], originally proposed in 1992. The current version dates from 2010 and it covers several types of domains, including Bin Packing. Other works of interest on this regard include [10,13,14], which we do not detail here due to space constraints.

We consider that the aforementioned approach deals with the problem reactively: first creates problems and then verifies the performance of solvers. But, we can change this approach for a proactive one, where we create instances tailored to the attributes of a solver. In this work, we explore the idea of using an evolutionary-based model for creating such an approach. Our objective is finding instances where a solver exhibits a desired behavior, allowing for investigations that revolve around the strengths and weaknesses of such solvers.

This need is not new and it has been explored in recent years. A recurring idea is to use an 'intelligent' generator, based on evolutionary computation, to target a solver. Some authors have achieved quite interesting results. For example, van Hemert [4,5] showed how to use an evolutionary algorithm to detect hard to solve instances in Constraint Satisfaction Problems (CSPs). Smith-Miles et al. [15,16] focused on intentionally creating instances of the Traveling Salesman Problem (TSP) that were easy or hard for certain algorithms.

To the best of our knowledge, no prior works have proposed a tool that allows assessment of the strengths and weaknesses of different solvers, through an evolutionary approach. Based on that, we asked whether it was possible to use a genetic algorithm for producing synthetic 1D bin packing instances under different scenarios. By doing so, we can fill the knowledge gap about instance tailoring for bin packing problems, while at the same time offering a tool for facilitating the study of different solvers under a variety of scenarios. This work contributes by presenting the rationale behind such a tool. Through it, insights about the elements that allow a solver to excel or fail can be more easily obtained.

This manuscript is organized as follows. Section 2 presents an overview of different elements related to the research, focusing on the domain we selected for testing, as well as on the solvers and features of interest. Afterwards, Sect. 3 briefly presents our proposed tool by describing how it operates. We then move on to summarizing the testing we carried out in Sect. 4. The corresponding data is given in Sect. 5. We wrap up our manuscript by laying out our conclusions in Sect. 6.

2 Fundamentals

This section presents some of the fundamental concepts related to our work. Here, we have restricted our analysis to problems in one dimension, though our model can be expanded to p dimensions without difficulty.

2.1 The 1D Bin Packing Problem

Bin Packing Problems (BPPs) represent the task of packing a set of items into containers with a given capacity (known as bins). The objective is to use as few bins as possible. In the one-dimensional case, objects have a single dimension (e.g., cost, length, time, etc.). Higher dimensional cases are represented by a combination of those, though as aforementioned, we limit ourselves to 1D BPPs. In this work, we consider the dimension of the problem as the length of the item. Hence, the problem can be stated as:

> *Given a list of objects, and their lengths, as well as a collection of bins with fixed size, find the smallest number of bins that can contain all objects.*

One-dimensional BPPs represent NP-hard combinatorial optimization problems, based on the partition problem. Finding the optimal solution is known to be exponentially difficult. Performance of traditional exact methods, such as branch and bound, degrade as the problem grows. Thus, other approaches are required. One of them is to use heuristic methods, representing simple and purpose-specific strategies that can obtain an approximate solution in a short enough time.

2.2 Some Solvers of Interest for the 1D Bin Packing Problem

In this work, we focus on a particular case of online BPPs. We consider problems where information from the whole instance is available, but where packaging is restricted to the first element, e.g., a production line where packaging is carried out by a fixed robot at the end of the line. Hence, all of the following heuristics pack the first item in the list:

- **First Fit Heuristic (BP-FF).** Find all the bins in which the item fits and place it into the lowest numbered one.
- **Best Fit Heuristic (BP-BF).** Find all the bins in which the item fits and place it in the one that leaves the least free space.
- **Worst Fit Heuristic (BP-WF).** Place the item in the bin with the most available space, as long as it fits.
- **Almost Worst Fit Heuristic (BP-AWF).** Similar to BP-WF, but places the item in the second emptiest bin (as long as it can hold it).

We want to stress out that if the item does not fit in any bin, a new one is opened and the item is placed there. Moreover, selecting these heuristics does not limit the scope of our work. In fact, our proposed model can handle different solvers as it only requires knowing how it performs for the instance being tailored.

2.3 Some Features for the 1D Bin Packing Problem

Even though our proposed model is featureless, it can be useful to define a set of features for analyzing the generated instances. We considered some based on the cost (i.e., length) of elements remaining in the instance [8]: Average length (AL), Standard deviation of the length (SL), and Ratio of big pieces (RBP, i.e., the ratio of elements whose length is above half of the bin capacity).

2.4 Some Performance Measures for the 1D Bin Packing Problem

Based on [12], in this work we adopt three metrics for measuring the performance of a solver over a given instance: the number of bins used, the number of completely filled bins, and the average waste per bin.

Consider this brief example, assuming a bin capacity of 10 and that the set of items to be packed is: $3, 8, 7, 4, 9, 1, 6, 2$. Figure 1 shows the solution with each heuristic, and the aforementioned performance metrics. The best solution is given by the best fit heuristic (BP-BF) since it used the least number of bins and filled them completely. Moreover, by only focusing on the number of bins, it would seem that the remaining heuristics are equally good. However, the worst fit heuristic (BP-WF) was unable to completely fill the same number of bins, and thus represents the worst heuristic. In fact, the remaining two heuristics (BP-FF and BP-AWF) yield the same solution, so they are tied.

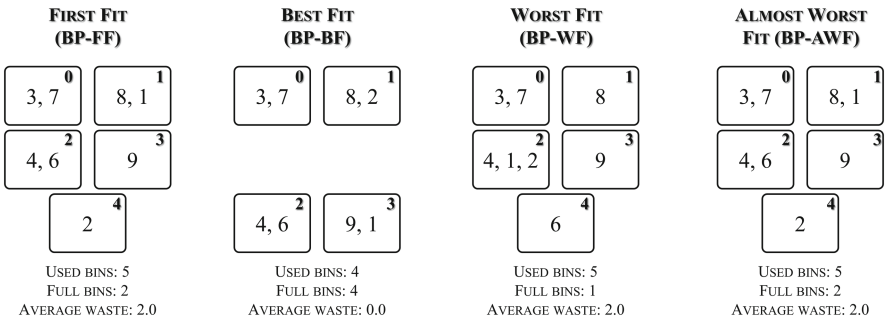


Fig. 1. Solution given by each heuristic, and their performance, for the set of items given by $3, 8, 7, 4, 9, 1, 6, 2$. The number in the upper right corner of each bin represents their ID.

2.5 Instances Used in This Work

Throughout our work, we only consider custom instances. As will be detailed in Sect. 4, we tailor these instances so that each solver behaves under the conditions defined by the user. The dataset is available upon request.

3 The Proposed Approach

In this work, we propose using a genetic algorithm for directly evolving the parameters of each item in the instance. To do so, a chromosome is represented by a binary string whose size depends on the number of items in the instance and on the maximum value for their length. Figure 2 shows an example of two chromosomes (A and B) with five items each, whose length is given by five bits. Since we are only interested in providing instances with positive length, these chromosomes can represent lengths in the range $[1, 32]$. As can be seen, repetitions are allowed within the instance.

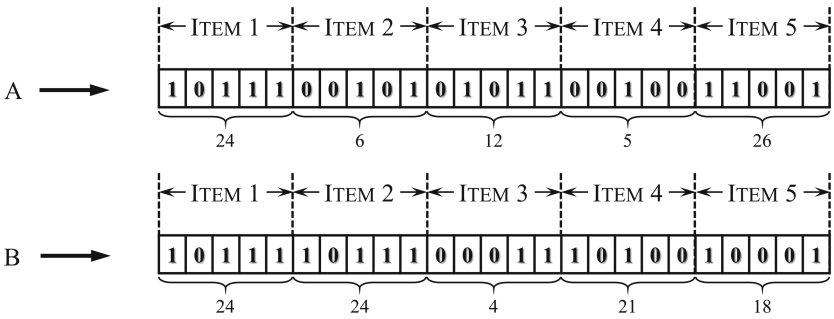


Fig. 2. Two sample chromosomes (A and B) for generating instances with five items and with item lengths in the range $[1, 32]$.

To produce an instance, three arguments are required: the capacity of the bin, C; the number of items for the instance, n; and the maximum length of each item, l_i. The process starts with a population of randomly initialized chromosomes. Afterwards, it keeps evolving guided by an user-defined objective function. This allows instances to be tailored to different means, therefore representing a powerful tool for studying specific traits of different solvers. As will be shown in Sect. 4, in this work we show different functions that can be used to guide the evolution through different paths.

As evolution progresses, chromosomes will change to reflect item lengths closer to the user requirements. To do so, two new offspring are created at each iteration, using standard genetic operators of selection, crossover, and mutation. The two worst chromosomes are then removed from the population so its size is preserved with each iteration. We want to stress that the number of items, as well as the capacity of the bin, are not encoded within the chromosome. Thus, they remain unchanged throughout the whole process.

4 Methodology

Testing was carried out on an Intel Core i7-6700 processor, with 16 GB of RAM, and running Windows 10 OS. The steady-state GA was tuned through exploratory experiments, omitted for the sake of space: 200 individuals, mutation rate of 0.01, crossover rate of 1.0 and 25000 generations per run (tops). To gather our data, we followed a three-stage methodology as described in Fig. 3.

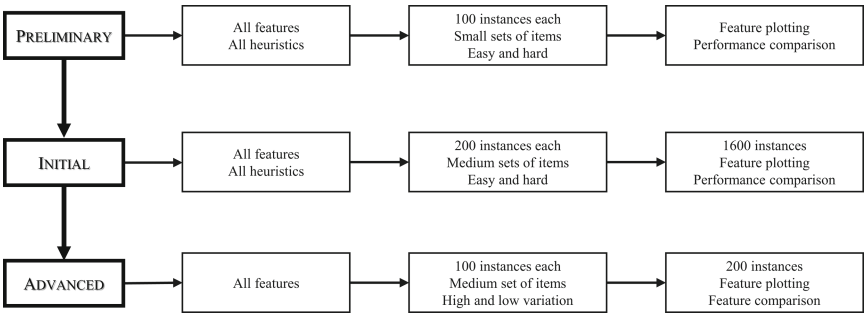


Fig. 3. Methodology followed throughout this work.

4.1 Preliminary Testing

Throughout this first stage, we focus on exploring whether our idea is promising. Thus, we strive to generate both, easy and hard instances, for each of the heuristics discussed in Sect. 2.2. For this batch of tests, we created 100 instances with 50 items each, for every scenario. This represents a total of 800 instances (8×100). In the first case, we used Eq. (1) to maximize the difference between the fitness (i.e., the average waste) given by the target heuristic (Fit_{one}) and that of the best remaining heuristic ($\min(Fit_{others})$). In the second case, we used Eq. (2) to also maximize the distance between the performance of the target heuristic and the worst of the remaining heuristics ($\max(Fit_{others})$).

$$F_{obj} = -(\min(Fit_{others}) - Fit_{one}) \tag{1}$$

$$F_{obj} = -(Fit_{one} - \max(Fit_{others})) \tag{2}$$

4.2 Initial Testing

Afterwards, we push our idea a bit further and strive to generate bigger instances. Hence, we demand sets of 100 items. Once again, we create instances for all 8 scenarios (i.e., 4 heuristics and 2 conditions). Moreover, we use our approach to generate 200 instances per scenario, to analyze the repeatability of our idea. This leads to a total of 1600 instances during this testing stage.

4.3 Advanced Testing

As a final effort to test our approach, we concentrate on generating a different kind of instance. This time, we modify the objective function in order to evolve instances with a maximum difference among solvers, and instances where all solvers perform the same. In the first case, we use Eq. (3) to maximize the standard deviation that heuristics exhibit over the instance. Here, Fit_i is the fitness (i.e., the average waste) of every heuristic, Fit_{avg} is the average fitness achieved by all solvers, and N_H is the number of heuristics (i.e., four for this work).

$$F_{obj} = -\sqrt{\frac{\sum_{i=1}^{N_H}(Fit_i - Fit_{avg})^2}{N_H - 1}} \tag{3}$$

In the second one, we use Eq. (4) to search for instances where all solvers perform equally. Thus, Fit_{Best} is the best fitness achieved by all solvers, whilst Fit_{Worst} is the worst one.

$$F_{obj} = (Fit_{Best} - Fit_{Worst})^2 \tag{4}$$

We pursue this idea to try and identify, based on the features from Sect. 2.3, the regions in the feature space where it is critical to select an appropriate solver and regions where it is of no importance. Since the idea is to also explore the versatility of our model, during this stage we use instances with 100 items each, and create 100 instances for each of the two scenarios.

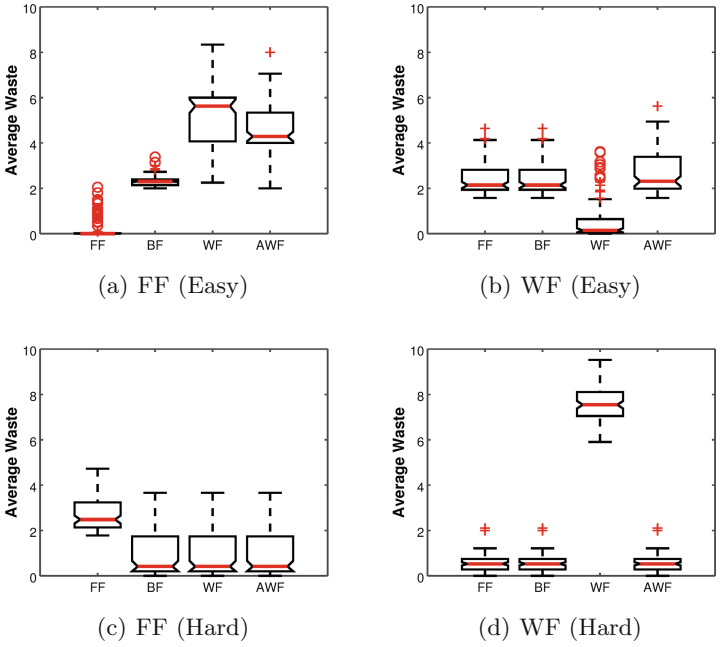
5 Experiments and Results

This section presents the most relevant data of our experiments. Because of space restrictions, we only focus on showing the performance of some heuristics in terms of average waste. Nonetheless, this does not mean that the other metrics were unsatisfactory.

5.1 Preliminary Testing

Figure 4 summarizes the average waste of the BP-FF and BP-WF heuristics over all generated instances. The first row relates to instances that can be efficiently solved by one heuristic (e.g., BP-FF or BP-WF), but which are poorly solved by the remaining ones. Similarly, the second row relates to instances poorly solved by one heuristic and efficiently solved by the remaining ones. In both cases, our proposed approach successfully tailors instances. In the first case, the average waste becomes minimum. In the second one, the average waste increases (sometimes even dramatically, e.g. for BP-WF).

Figure 5(a) shows the center of each set of instances, located at different positions. Interestingly, easy and hard to solve instances, for a specific heuristic, exhibit opposing behaviors. For example, increasing the difficulty of instances



(a) FF (Easy) (b) WF (Easy)

(c) FF (Hard) (d) WF (Hard)

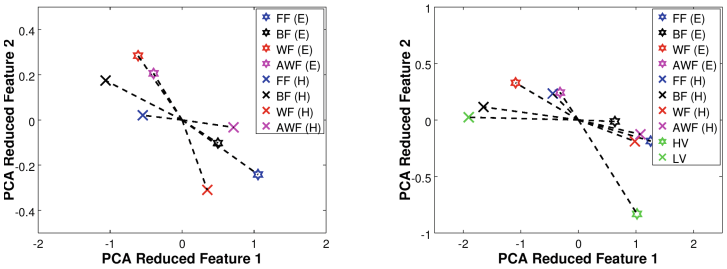
Fig. 4. Average waste achieved by the First Fit (FF) and the Worst Fit (WF) heuristics over the sets of instances generated in the preliminary testing stage. Boxplots reflect data for 100 instances. Other solvers are omitted due to space restrictions.

for BP-FF, leads to instances whose items are more varied, but which are also two units longer (on average). Even more, there are 10% more big pieces within hard instances. Nonetheless, in the case of BP-WF harder instances are those with fewer big pieces (about 7% less), leading to smaller items (about one unit on average).

5.2 Initial Testing

Although increasing the number of items in the instance to 100 makes it harder for our model to generate the instances with the requested behavior, it is still able to generate quite useful results (Fig. 6). This time around, there are also scenarios where a solver has virtually no waste at all (e.g., for the best fit heuristic), and scenarios where a heuristic wastes way more space than other approaches. Even in those cases where the behavior of the heuristic of interest is not that different from the others, there are still elements worth remarking. For example, consider hard-to-solve instances for the best fit heuristic. Even though medians are quite close, best fit is the only heuristic unable to achieve average waste values below one.

A plot of the location of the 1600 generated instances, again reveals an interesting pattern (Fig. 5(b)). As in the previous stage, easy and hard instances are



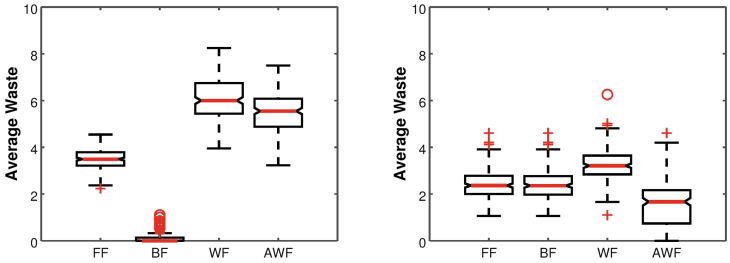
(a) Preliminary stage (b) Initial stage

Fig. 5. Centroid of all instances generated in the preliminary and initial testing stages. Stars indicate easy-to-solve (E) instances for a specific heuristic and crosses indicate hard-to-solve (H) instances for a specific heuristic. Data is also shown for instances with High variation (HV) and Low variation (LV).

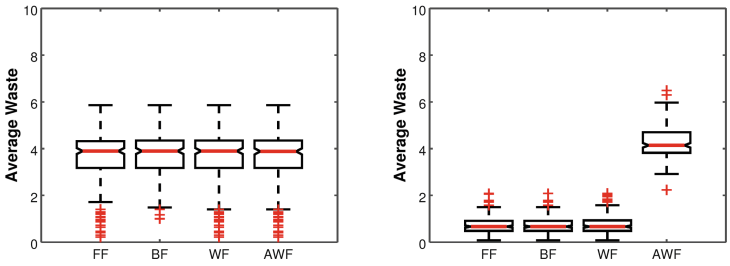
located on opposing parts of the feature space. It is also interesting to observe that changing the difficulty of a heuristic implies relatively the same movement in the feature space: for the first fit and best fit heuristics, it represents a shift from right to left and upwards; for the worst fit and almost worst fit, it changes and now goes from left to right and downwards. Hence, the behaviour from the previous stage holds. In fact, the ratio of big pieces (RBP) increased from 39% to 51% for BP-FF, while it decreased from 55% to 41% for BP-WF.

5.3 Advanced Testing

As mentioned in Sect. 3, this stage pushes our approach further by trying out different objective functions. Figure 7 shows two scenarios of interest: one with varied performance, and one with the same performance. Once again, our proposed model was able to cope with the situation, evolving instances favorable for the user needs. Taking into account that the bin capacity was set to 30, we consider that a gap in heuristic performance of about 10 is quite remarkable. Moreover, the model was actually able to evolve instances where heuristics had the exact same behavior, thus representing a region of the feature space where it is irrelevant to spend resources on determining which heuristic to use. The centroid of these two sets of instances can be seen in Fig. 5(b). As expected, the scenario with the highest variation is the one farthest away. This can be explained since the other scenarios only considered that a given solver was either good or bad in comparison to the others, but did not seek that the solvers exhibited a varied performance. Similarly, the set of instances with the lowest variation is close to the set of hard-to-solve instances for the best fit heuristic. This can also be explained since, as it was mentioned above, this scenario exhibits a low variation (with only some instances yielding average waste values below unity).

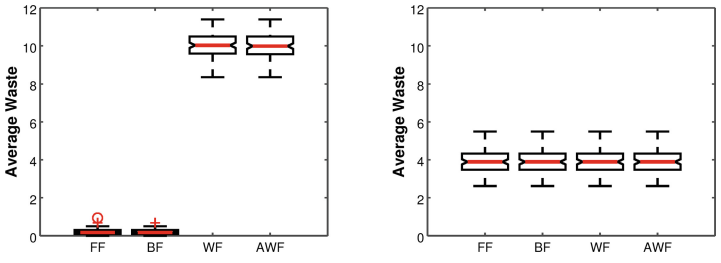


(a) Best Fit (Easy) (b) Almost Worst Fit (Easy)



(c) Best Fit (Hard) (d) Almost Worst Fit (Hard)

Fig. 6. Average waste achieved by the Best Fit (BF) and the Almost Worst Fit (AWF) heuristics over all the sets of instances generated in the initial testing stage. Boxplots reflect data for 200 instances. Other solvers are omitted due to space restrictions.



(a) High Variation (b) Low Variation

Fig. 7. Average waste achieved by all solvers when operating over instances generated with high variation (a) and with low variation (b).

6 Conclusions

In this work we presented an evolutionary-based model for tailoring instances to specific needs. The tool we propose may allow for an in-depth study of different heuristics. It can be used to create instances that exploit heuristic strengths and weaknesses, so knowledge about when to use each one may be derived. However, we did not pursue such endeavor here, mainly due to space restrictions.

Nonetheless, we tested our approach by generating over 2500 instances for the 1D Bin Packing Problem, distributed along four different heuristics. Our testing included scenarios where one solver excels while the others fail, and scenarios where one solver fails while the remaining ones excel. To push our generation model even further, we included instances where the solvers performed as diversely as possible, and instances where all solvers performed equally.

Data exhibited interesting elements. For example, we created easy instances for the first fit heuristic (BP-FF), where the median number of bins was between 7% and 20% lower than for the remaining heuristics. Similarly, the median number of completely filled bins for BP-FF was over 10% higher than for the second best heuristic, and about eight times higher than for the worst heuristic. In some cases results were not as astonishing. But, there was always some benefit. Consider easy-to-solve instances for the worst fit heuristic, and hard-to-solve instances for the best fit heuristic. In the former, the heuristic allowed for near zero average waste. In the latter, the heuristic was the only one unable to achieve near zero average waste. This means that, through our approach, one can find instances with specific behaviors without the need for exhausting all combinations of elements, nor deriving mathematical expressions that relate the items.

Also worth noting is that as difficulty increases, instance location within the feature domain shifts. For example, for the first fit and best fit heuristics, instances shifted from right to left and upwards. However, for the worst fit and almost worst fit, they migrated from left to right and downwards.

Regarding the final batch of tests, we can conclude that our proposed model can adapt to different kinds of situations. In this case, it was able to evolve instances where solvers exhibited a high variation on their performance, generating gaps of average waste of about 33%. Moreover, it was also able find configurations where all solvers performed exactly the same, representing the region of the feature space where it becomes unnecessary to select a specific heuristic.

References

1. Beasley, J.: OR-library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**(11), 1069–1072 (1990)
2. Drake, J.H., Swan, J., Neumann, G., Özcan, E.: Sparse, continuous policy representations for uniform online bin packing via regression of interpolants. In: Hu, B., López-Ibáñez, M. (eds.) EvoCOP 2017. LNCS, vol. 10197, pp. 189–200. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55453-2_13
3. Gomez, J.C., Terashima-Marín, H.: Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems. Genet. Program. Evol. Mach. **19**, 151–181 (2017). https://doi.org/10.1007/s10710-017-9301-4
4. van Hemert, J.I.: Evolving binary constraint satisfaction problem instances that are difficult to solve. In: Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC 2003), pp. 1267–1273. IEEE Press (2003)
5. van Hemert, J.I.: Evolving combinatorial problem instances that are difficult to solve. Evol. Comput. **14**(4), 433–462 (2006)

6. Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the pareto archived evolution strategy. Evol. Comput. **8**(2), 149–172 (2000)
7. Koch, T., et al.: MIPLIB 2010. Math. Program. Comput. **3**(2), 103–163 (2011)
8. López-Camacho, E., Terashima-Marín, H., Ross, P.: A hyper-heuristic for solving one and two-dimensional bin packing problems. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011), pp. 257–258 (2011). https://doi.org/10.1145/2001858.2002003
9. Lust, T., Teghem, J.: The multiobjective multidimensional knapsack problem: a survey and a new approach. Int. Trans. Oper. Res. **19**(4), 495–520 (2012)
10. Martello, S., Pisinger, D., Vigo, D.: The three-dimensional bin packing problem. Oper. Res. **48**(2), 256–267 (2000)
11. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, Hoboken (1990)
12. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation - GECCO 2011, p. 2011 (2011). https://doi.org/10.1145/2001576.2001846
13. Petursson, K.B., Runarsson, T.P.: An evolutionary approach to the discovery of hybrid branching rules for mixed integer solvers. In: Proceedings - 2015 IEEE Symposium Series on Computational Intelligence, SSCI 2015, pp. 1436–1443 (2016)
14. Pisinger, D.: Where are the hard knapsack problems? Comput. Oper. Res. **32**(9), 2271–2284 (2005)
15. Smith-Miles, K., van Hemert, J.: Discovering the suitability of optimisation algorithms by learning from evolved instances. Ann. Math. Artif. Intell. **61**(2), 87–104 (2011)
16. Smith-Miles, K., van Hemert, J., Lim, X.Y.: Understanding TSP difficulty by learning from evolved instances. In: Blum, C., Battiti, R. (eds.) LION 2010. LNCS, vol. 6073, pp. 266–280. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13800-3_29
17. Sosa-Ascencio, A., Terashima-Marín, H., Ortiz-Bayliss, J.C., Conant-Pablos, S.E.: Grammar-based selection hyper-heuristics for solving irregular bin packing problems. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO 2016 Companion, pp. 111–112. ACM Press, New York (2016). https://doi.org/10.1145/2908961.2908970
18. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: improving the strength pareto evolutionary algorithm. In: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, pp. 95–100 (2001)
19. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999)

Machine Learning and Evolutionary Algorithms

Adaptive Advantage of Learning Strategies: A Study Through Dynamic Landscape

Nam Le(✉), Michael O'Neill, and Anthony Brabazon

Natural Computing Research and Applications Group,
University College Dublin, Dublin, Ireland
namlehai90@gmail.com

Abstract. Learning can be classified into two categories: *asocial* learning, e.g. trial-and-error; and *social* learning, e.g. imitation learning. Theory using mathematical models suggest that social learning should be combined with asocial learning in a strategic way (called *learning rule* or *learning strategy*), and that that combination should be scrutinised under different environmental dynamics, to see how advantageous the learning rule is. More interestingly, learning has been shown to be beneficial to the evolutionary process through the **Baldwin Effect**. This paper investigates the adaptive advantage of social learning when combined with asocial learning under a number of environmental variations. We propose a Dynamic Landscape as well as an algorithm combining both asocial and social learning in order to test our hypotheses. Experimental results show that if each individual in the population is either asocial or social, but not both, the average fitness of the population decreases when the proportion of social learners increases as the environment changes. Moreover, a population consisting entirely of asocial learners outperforms the previous type of population. If every individual agent in the population can perform both asocial and social learning depending on a strategic rule, the evolving population outperforms the two previous populations with respect to average fitness.

Keywords: Social learning · Cultural evolution · Genetic algorithm
Dynamic environment · Baldwin effect

1 Introduction

Evolution and learning are two different ways in which the behavior, and other traits, of organisms can change in order to adapt to environmental variations. Evolution is change at the genetic level of a population, while learning, on the other hand, is change at the phenotypic level of an individual. The idea that the two forms of adaptation interact and complement each other was once proposed by Baldwin [1], called the Baldwin Effect. Hinton and Nowlan (henceforth

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 387–398, 2018.
https://doi.org/10.1007/978-3-319-99259-4_31

H&N) presented a computer model to investigate the Baldwin Effect in simulation [2], showing that learning, more specifically asocial learning, facilitates the evolutionary process and enhances the fitness of the population in a Needle-in-a-haystack landscape. Their initial success motivated several further studies, such as [3,4], to show how learning can enhance the evolutionary search.

Generally, learning can be classified into two forms. *Asocial* (or individual) learning (IL) – learning by oneself through direct interaction with the environment, e.g. trial-and-error, and *social* learning (SL) – learning from others, e.g. imitation – are two alternative ways for an individual agent to acquire information from the environment at the phenotypic level. SL has been observed in organisms as diverse as primates, birds, fruit flies, and especially humans [5]. Although the use of SL is widespread, understanding when and how individuals learn from others is a significant challenge. SL is generally less time-consuming, but relies on information produced by others. So when the environment changes, the information from others is likely to be outdated and SL becomes *maladaptive* (not adaptive). On the other hand, IL through trial-and-error is costly, but capable of producing new information when the environment happens to change.

This opens a curious question when an organism should rely on SL rather than IL, and under which environmental condition social learning would evolve, or be adaptive. Several theoretical models have shown that individual agents capable of learning in a strategic way outperform those that are able to learn individually or socially, but not both [6–8].

The main aim of this paper is to investigate, through computer simulation, whether organisms should rely on SL or IL, and what the plausible strategy for an organism could be when the environment changes. We combine evolution and both forms of learning to see how they behave under a dynamic landscape we call Dynamic Needle-in-a-hay-stack. This paper is built upon the success of the previous work in [9]. In the remainder of this paper, we briefly present research on learning and evolution. Social learning and related concepts are briefly introduced. We in turn describe the experiments we use in this paper. Results are analysed and discussed, then the conclusion and some future directions are proposed.

2 Background

2.1 Social Learning

SL has been studied in various disciplines, including Cognitive Biology, Evolutionary Psychology, Behavioral Ecology, Cognitive Science and Robotics. In general, SL covers several mechanisms through which individual organisms learn from others, such as stimulus enhancement, observational conditioning, imitation, and emulation (please refer to [5,10] for the definition of these mechanisms). In this study we focus on one of these mechanisms, namely imitation learning. In this instance of learning, the observer directly copies the behavior of the observed animal in order to complete a novel task.

SL, at first glance, seems to be adaptive at a low cost when individual agents can acquire information from others without incurring the cost of trial-and-error learning. Thus, it is plausible to think that SL will result in more effective learning outcomes. Contrary to this belief, it has been found that agents should not learn socially all the time [6,11]. It is argued that individual learners produce new information about the environment, though at a cost. Social learners avoid this cost by copying the existing behaviors of others, but do not generate new information. Therefore, it is highly likely that social learners will copy outdated information when the environment changes, reducing the average fitness of the population.

Several theoretical models have been proposed to investigate how to use SL effectively [6–8]. It is said that social learning should be combined with individual learning in a strategic way in order to have an adaptive advantage. Social learning strategies consist of rules specifying the way an individual relies on social learning by answering three questions as follows:

i. When an individual should learn;
ii. From whom they should learn; and
iii. What information should be learned.

The question of *when to copy* covers the decision as to when to seek social information. *Whom to copy* may depend on factors such as the social structure of the population and the ability of the individual to recognise whether other individuals are obtaining higher payoffs. Possibilities include the copying of the most successful individual, copying of kin, or adherence to a social norm by copying the majority. *What to copy* considers which behavior or more specifically what part of that behavior to copy.

In addition to the *Who* question, the transmission from demonstrators to observers are classified into three types [12]. The first is *vertical transmission* – transmission from parents to their children. The second is *oblique transmission* in which cultural traits will be passed to an individual from another individual from the previous generation but differs from its parent. The last is *horizontal transmission* in which an observer learns from a demonstrator in its current generation. In the scope of this paper, we only use *oblique transmission* in our experiments.

2.2 Learning and Evolution in Computer Simulation

H&N presented a classic paper in 1986 [2] to demonstrate an instance of the Baldwin Effect in computer. We discuss this model in detail for clarity. In H&N's model, suppose the task is to find the all-ones target string 111...1 (20 bits). There is only one correct solution – an individual with configuration exactly matched with the target string – which has the fitness of 20. All other configurations are wrong and have the same fitness of 1. This forms a Needle-in-a-haystack landscape whereby an evolutionary search alone cannot find the solution [2].

H&N presented an idea that encodes an individual's genotype whereby one part is fixed by genetic-like information, and the other part is plastic which

allows for learning during the lifetime of the individual. Each individual agent has a *genotype*—a string of twenty characters. Each position in a genotype, or *locus*, can have three alternative values: '0', '1', and '?'. Each locus is randomly initialized with 25% chance of being assigned a '0', 25% chance of being '1', and 50% chance a '?'.

In addition to the above two types of agent (correct or incorrect), there exists another type of agent – called *potential* individual – which will be allowed for life-time learning. The genotype-phenotype mapping is one-to-one and at birth, each individual has its phenotype string identical to its genotype string. An individual is potential only if in its initial genotype, every locus excluding locus with plastic value '?' is matched with corresponding locus in the target string. In case of H&N's problem, a potential individual could have its initial genotype comprising of only '1' and '?'. The allele '?' allows for lifetime learning (or plasticity), over 1000 rounds. On each round, an individual agent is allowed to do individual learning by changing its allele '?' to either '0' or '1' as the expressed value. After learning, the fitness of that potential individual agent x_i is calculated as:

$$f(x_i) = 1 + \frac{19(1000 - n)}{1000} \tag{1}$$

in which n is the number of trials required to find the correct combination of alleles - the all-one string. It can be inferred from the fitness function that the more trials an agent needs, the lower the fitness it will get. By allowing life-time learning, H&N showed that learning can create a gradient which facilitates evolution to search for the solution.

Since the success of H&N's model, there have been a number of studies showing that learning can enhance an evolutionary process, especially when the environment is changing [3,4]. Recently, Le et al. [9] presented a model building on H&N's simulation, in which they combine evolution, asocial and social learning. It was shown that social learning alone fails to search on Needle-in-a-haystack, but social learning when coupled with individual learning outperforms individual learning alone with respect to average fitness of the population.

In this paper, we build on the success of the previous simulation in [9], in which we propose a dynamic version of H&N's Needle-in-a-haystack to see how individual learning or social learning behaves under different environmental dynamics. Experimental designs are discussed in the following section.

3 Experimental Design

3.1 Dynamic Needle-in-a-haystack Landscape

We create a dynamic version of H&N's landscape called Dynamic Needle-in-a-haystack, in which we use two parameters to control the dynamics of the landscape, namely *frequency* and *magnitude* of change. The first parameter tells us after many generations the target (needle) will move to another point in the landscape, while the latter helps determine the likelihood of change for each element of the target. Assume that at a generation g the target is all-one (20

bits of one), *frequency* = 10 and *magnitude* = 0.1 (10%). This informs us that after 10 generations or at generation $g + 10$ the target t = 111...1 (20 bits of 1) is likely to be changed. The magnitude of 0.1 tells us that there are, on average, $20 \times 10\% = 2$ bits in the target that are likely to be modified. For each bit in the target sequence, a random number is generated and then compared with the *magnitude*: if the random value is less than 0.1, the current bit is mutated to its subtraction from 1 (1 becomes 0, and vice versa). Suppose the new target at generation $g + 10$ is $t_1 = 001...1$ (two first bits are changed). There is only one right sequence of bits that exactly matches the new target t_1 and has the fitness of 20. Otherwise, all other configurations are incorrect and get the same fitness of 1. This landscape, again, constitutes a Needle-in-a-haystack, but the needle is moving after a number of generations. That is why we call this landscape *Dynamic Needle-in-a-haystack*.

We also call the period when the environment is unchanged the *interval of stability*. Therefore, the interval of stability has the same value as the dynamic frequency.

3.2 Experiment Setup

In this section, we present the experimental setups used in our paper. It is often said that evolutionary search finds it hard to search in 'Needle-in-a-haystack' landscape. Furthermore, it was claimed that an evolutionary search alone failed in this type of landscape [2]. Le et al. [9] went further to show that an evolutionary search combined with social learning alone also failed to find a solution in a Needle-in-a-haystack. We conduct three experiments with the parameter settings, as shown in Table 1.

Table 1. Parameter setting

Parameter	Value
Original target	111...1 (20 bits of 1 s)
Genome length	20
Replacement	Generational
Generations	50
Population size	1000
Selection	Fitness-Proportionate selection
Reproduction	Sexual reproduction
Mutation rate	0.01
Fitness function	Eq. 1
Maximal learning trials	1000
Frequency	5, 10, 20
Magnitude	0.05, 0.075, 0.1

We run our experiments through 9 different combinations of *frequency* and *magnitude*. For frequencies 5, 10, 20 there will be 10, 5, and 2 times of change in the environment, respectively. In case frequency = 5 or 10, generation 50 will see a change in the environment. It can be understood that the lower the *frequency* value, the faster the target will change; the bigger the value of *magnitude*, the bigger the change of the target. The environment becomes more dynamic or harder to cope with by faster changing and bigger magnitude of change, and vice versa.

Please note that, unlike the so-called memetic algorithm and Lamarckian Evolution, learning in our experiments only happens at the phenotypic level, what an individual learns does not change its genotype. The recombination operators work on the genotypic level, so children may inherit question marks from their parents.

Setup I: A Population of Individual Learners

The first experimental setup is an evolving population of individual learners only (as in H&N model), in which an evolutionary search is combined with IL (denoted EVO+IL). IL performs a local search process by which each '?' allele will guess its value to be '0' or '1' in each learning trial. The evolutionary algorithm in our experiment is a genetic algorithm with crossover and mutation (with mutation rate of 0.01).

Setup II: A Population of Single-Role Learners

In the second experiment setup, we simulate a population of single-role individuals – individuals that are either social learners or individual learners, but not both (denoted EVO+IL&SL). The reason for this experimental design is that we are curious to know how social learning or individual learning would evolve under various environmental dynamics and how they contribute to the average fitness of the population. Unlike EVO+IL, we have two types of individuals in the population now. We add one more bit, or *gene*, called *learning mode*, which is either 0 or 1, onto the genome of each individual. If that value is 0, the individual is likely to learn individually; conversely, if that value is 1, the individual is likely to learn socially. A noteworthy point here is that in our landscape, only potential individuals are able to perform lifetime learning. That means, social and asocial learners are potential individuals with learning mode equal to 1 and 0, respectively. Learning mode is initialised with 50% at 0 and 50% at 1. It should be noted that through recombination, the learning mode of a child is set to be the learning mode of the better individual between its parent. Mutation does not touch the learning mode of the child.

In order to implement social learning, first we propose the imitation procedure, with pseudo-code described in Algorithm 1 below. This presents the process by which an individual observer imitates the phenotype of its demonstrator. The imitative process starts by extracting the positions of question marks in the phenotype of the observer. For each question mark position, the observer will decide whether to copy exactly the *trait* or a mutated version of that *trait* from the demonstrator.

Algorithm 1. IMITATION

```
1: function IMITATION(observer, demon, fidelity = 1)
2:     questions = [] comment: question mark position array
3:     for position i ∈ observer.pheno do
4:         if i =? then
5:             questions.add(i)
6:         end if
7:     end for
8:     for i ∈ questions do
9:         observer.pheno(i) = demon.pheno(i)
10:    end for
11: end function
```

At each generation, an asocial agent learns by itself like in EVO+IL model, whereas a social learner imitates its demonstrator. Every individual agent has the same demonstrator that is the best individual in terms of fitness from the previous generation. Because we adopt *oblique transmission*, there is no SL at the initial generation.

Experiment III: A Population of Strategic Individuals

The third setup we evolve a population of *strategic* individuals – individuals that can perform both SL and IL based on a learning rule (denoted EVO+Strategy). Unlike EVO+IL&SL, the population now has just one type of individual - strategic individuals. We specify the learning strategy for every individual agent as follows: At each generation, an agent first looks at its demonstrator (chosen the same as in EVO+IL&SL), and determines whether to learn from that demonstrator or not. If the demonstrator is still *adaptive* in the current generation, the agent imitates the demonstrator based on Algorithm 1; otherwise, the agent learns individually. The demonstrator is said to be *adaptive* in the current environment if its phenotype exactly matches with the target in the current environment. This means every agent determines whether it expresses as an individual learner or as a social learner based on a given rule. After lifetime learning process for each agent, the population goes through selection and reproduction as in EVO+IL.

4 Results, Analyses, and Explanations

First, we look at the average fitness of the population as a measurement of how well each simulation performs. All plots are grouped together, sharing the same labels for x-axis and y-xis as well as the annotation. The results are presented and discussed in a comparative manner below.

A similar trend can be recognised in Fig. 1 that there is a drop in every population with all settings at the generation when the environment begins to change. This is understandable because when the environment changes, a number of adaptive behaviors from previous generations are no longer fit in the current generation, reducing the average fitness of the population. By looking at the behavior of each corresponding line through each row or each column of Fig. 1,

we can see another shared behavior that the more dynamic, or difficult, the environment is, the lower the average fitness of the population.

How does each type of population comparatively cope with these environmental dynamics? It is shown that EVO+IL outperforms EVO+IL&SL in all environmental circumstances. More than that EVO+IL&SL shows its inability to track the dynamics of the environment as the average fitness goes down to or much closer to the lowest value of 1 in most cases, except the easiest landscape (when the environment changes most slowly and the magnitude of change is smallest) though it just reaches around 2.5 at the end.

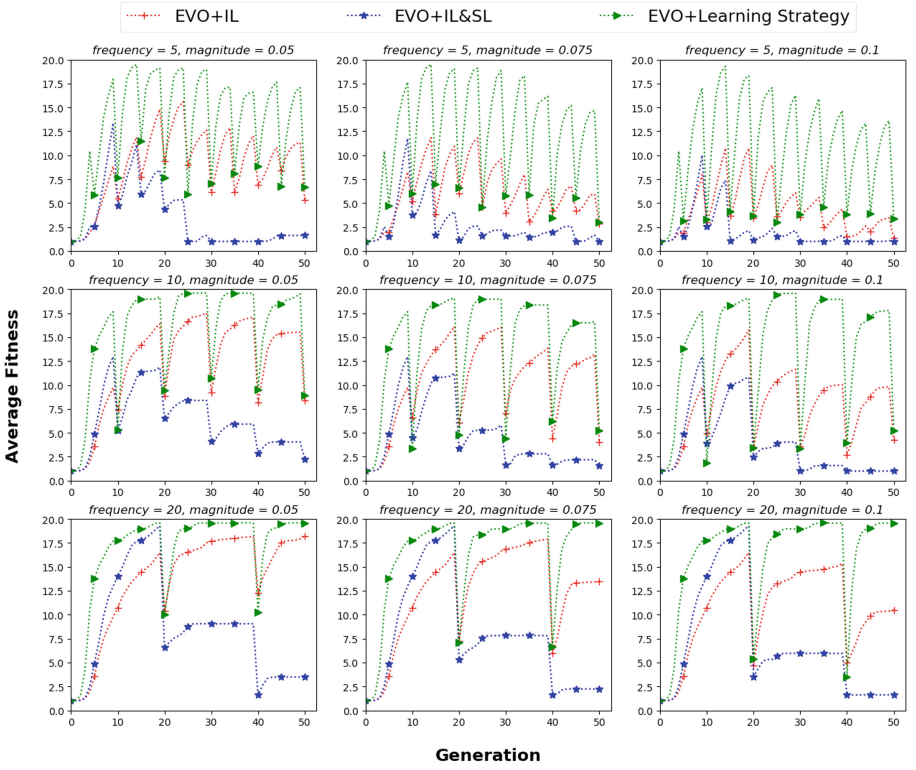


Fig. 1. Average fitness over generations. The red, blue, and green dotted-lines for EVO+IL, EVO+IL&SL, and EVO+Strategy, respectively. (Color figure online)

Remember that EVO+IL&SL evolves a population comprising of both asocial and social learners, so why does it behave very badly and even worse than the population with individual learners alone? We have briefly talked of the fact that SL, though less costly, is 'information parasitism' that is unable to produce new information. When the environment changes, social learners are likely to copy outdated information in case their demonstrators are no longer adaptive in the

current generation. In our experiments, the best individual at generation g is set to be the demonstrator for all social learners in generation $g + 1$. Assume that the environment changes at generation $g + 1$, the demonstrator becomes *maladaptive*. All social learners at generation $g + 1$ copy *maladaptive* behavior, thus becoming *maladaptive* and get the same lowest fitness of 1 and reducing the average fitness of the population.

However, the population in EVO+IL&SL is composed of both asocial and social learners at the initial generation. It is said that individual learners through trial-and-error are able to track the dynamics in the environment. So why are individual learners unable to help the population to cope with environmental dynamics? We hypothesise that in EVO+IL&SL the proportion of asocial learner decreases to a very small amount so that there are not enough asocial learners to track the environmental changes.

We know that the imitative process gives social learners an advantage in terms of time and learning trial required to find the correct solution. While asocial learners have to trial-and-error through a number of trials, social learners just need to copy all bits from the correct demonstrator's phenotype to its phenotype. Based on our fitness function (Eq. 1), the more the learning trials needed, the lower the fitness value. Consequently, lower cost gives social learners advantages over asocial learners when the environmental is stable. This argument can be verified by looking at Fig. 1 as at some initial generations, the average fitness of EVO+IL&SL is higher than that of EVO+IL during the interval of stability. Therefore, natural selection will favor social learners during these earlier generations, individual learners become less dominant and are likely to disappear.

We calculate the frequency of asocial and social learners over generations to verify this hypothesis. Figure 2 shows that in EVO+IL&SL the frequency of asocial learner is very low in all cases and tends to go down to zero sooner when the environment becomes harder. This fits with our above hypothesis. In addition to the above analyses, remember that in EVO+IL&SL each individual is initially encoded as either an asocial or a social learner. When social learners are more likely to be favoured by natural selection, more social learner 'genes', or learning mode 1, occur in the reproductive pool, so that offspring produced through sexual recombination are more likely to have the learning mode as social learners. The asocial learner gene becomes less prevalent over time, and in most cases we see it becomes distinct. That is why as the environment changes, the population has less asocial learners to track the environment, hence the average fitness reduces down to the lowest value of 1 in almost all cases.

Conversely, EVO+IL still maintains a higher number of asocial learners than that of EVO+IL&SL to track environmental variations. That is why EVO+IL has a higher average fitness than that of EVO+IL&SL in all cases.

One important point to be extracted here is that SL can give a population advantages when the environment is in a stable interval, whereas IL is much more powerful at the point when the environment changes. That is why we have designed the learning rule in EVO+Strategy to make use of the advantages of both IL and SL. It is easily seen in Fig. 1 that EVO+Strategy outperforms

other settings in all cases with respect to average fitness. Moreover, during the interval of stability, EVO+Strategy can reach higher point of average fitness than EVO+IL, especially in the hardest case (frequency = 5, magnitude = 0.1).

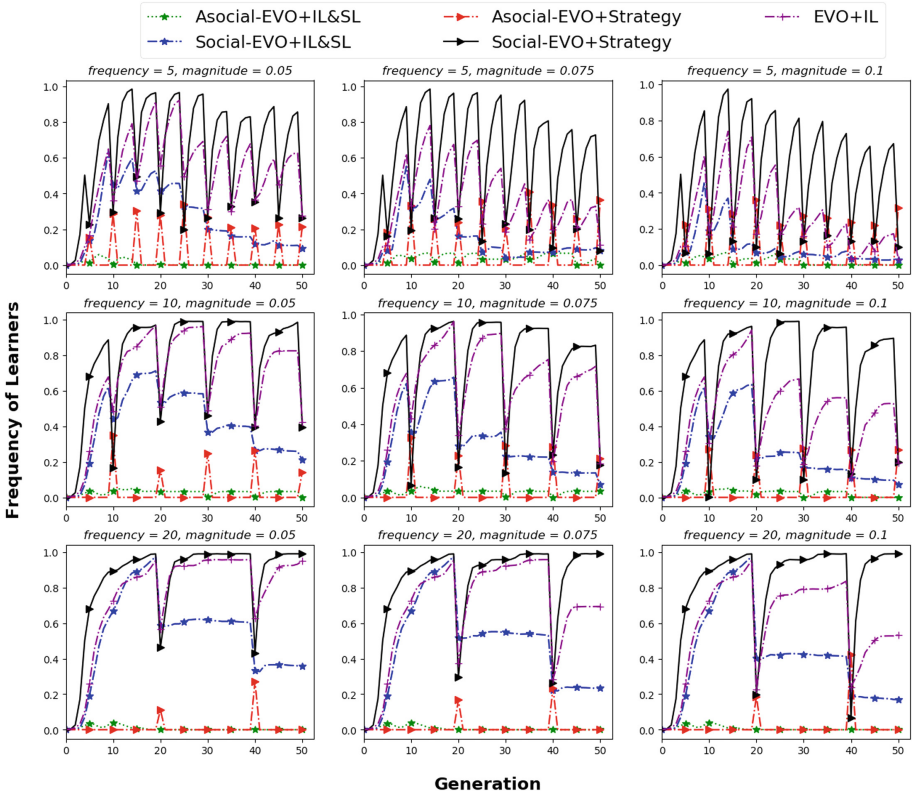


Fig. 2. Frequency of social and asocial learners over generations. Green and blue, red and black lines show the frequency of asocial and social learners in EVO+IL&SL and EVO+Strategy, correspondingly. The purple lines represents the frequency of asocial learners in EVO+IL. (Color figure online)

One plausible explanation for the superiority of EVO+Strategy is that the learning rule used in EVO+Strategy can exploit the advantage of both IL and SL. Remember that every individual agent in EVO+Strategy can perform as either a social learner or an asocial learner. At each generation, each strategic agent checks if its demonstrator is adaptive or not: if adaptive, it imitates the demonstrator; otherwise the agent learns asocially. By this learning rule, EVO+Strategy removes the case that agents will learn maladaptive behaviors like what has been observed in EVO+IL&SL by allowing more asocial learning in case the environment changes. Moreover, when the environment is in a stable

interval, the system now allows for more SL to be expressed to make use of the advantage of SL over IL, hence the average fitness is remarkably increased.

We also calculate the frequency of social and asocial learners in EVO+Strategy. It can be observed in Fig. 2 that the frequency of asocial learner is closer to or becomes zero when the environment is stable. However, at the point when the environment changes, the frequency of asocial learners increases. The frequency of social learners, in contrast, goes down at the point when the environment changes, reducing the 'information parasitism' issue and the problem of copying outdated information. Furthermore, when there are some asocial learners to help the population to track the environmental dynamics, the population is shown to maintains a high proportion of social learners and this percentage goes higher in every stable interval. Therefore, the average fitness of the population increases during every interval of stability. This observation fits with the description of the system as well as the behavior and analyses we have given above on the average fitness.

It can also be observed that during the interval of stability, the frequency of asocial learners in EVO+IL is lower than that of social learners in EVO+Strategy in all cases. This gives EVO+Strategy an advantage over EVO+IL to maintain a higher average fitness because social learners can copy correct behaviors at a lower cost compared to asocial learners during an interval of stability.

5 Conclusion and Future Work

We have set out to understand the role social learning may have on the evolutionary process in various environmental dynamics. For the specific landscape and the parameter setting used in this paper, experimental results have empirically shown that social learning is more advantageous when the environment is stable, whereas when the environment happens to change asocial learning is required to track the environment. A learning rule combining both social and asocial learning has been designed and the population with a learning strategy has shown to have a much better adaptive advantage, measured by the average fitness. Several plausible explanations have been presented in this paper for these observations.

In the scope of this paper, we have mainly discussed the adaptive advantage of social learning, asocial learning, and learning strategies in dynamic environments. The evolution of social learning can also been extracted from our results and analyses on the frequency of learners. It is suggested to investigate the question as to how social learning evolves more deeply in future work. It is also recommended that we use different forms of social transmission, such as *vertical transmission* and *horizontal transmission*.

The learning strategy used in this paper is designed by the system designer. In future work, we would like to let the individuals themselves evolve their own learning strategies. One proposal is to create *genes* controlling a learning strategy for each individual agent as was done with EVO+IL&SL. Instead of encoding just one bit for learning mode, we can encode two bits to specify the learning

rule, whether to be expressed as social or asocial learning. By doing this, we are able to observe the dynamics of rule changing when the environment changes over time. The motivation for this is that we want to let evolution to optimise the learning strategy for each individual agent, as evolution has done for living organisms, including humans [6].

The paper has empirically verified that social learning should be used in a savvy way to enhance the behavior of a population, complementing to some theoretical findings in the trans-disciplinary research on social learning. Future work will investigate the method and verify the findings in this paper on different types of problems and landscapes, such as NK-landscape [13] or Artificial Life and Robotics domains [3,4].

References

1. Baldwin, J.M.: A new factor in evolution. Am. Nat. **30**(354), 441–451 (1896)
2. Hinton, G.E., Nowlan, S.J.: How learning can guide evolution. In: Belew, R.K., Mitchell, M. (eds.) Adaptive Individuals in Evolving Populations, pp. 447–454. Addison-Wesley Longman Publishing Co., Inc, Boston (1986)
3. Ackley, D., Littman, M.: Interactions between learning and evolution. In: Langton, C.G., Taylor, C., Farmer, J.D., Rasmussen, S. (eds.) Artificial Life II, SFI Studies in the Sciences of Complexity, vol. X, pp. 487–509. Addison-Wesley, Reading (1992)
4. Nolfi, S., Parisi, D., Elman, J.L.: Learning and evolution in neural networks. Adapt. Behav. **3**(1), 5–28 (1994)
5. Heyes, C.M.: Social learning in animals: categories and mechanisms. Biol. Rev. **69**(2), 207–231 (1994)
6. Laland, K.N.: Social learning strategies. Learn. Behav. **32**, 4–14 (2004)
7. Feldman, M.W., Aoki, K., Kumm, J.: Individual versus social learning: evolutionary analysis in a fluctuating environment. Santa Fe Institute, Working Papers (1996)
8. Wakano, J.Y., Aoki, K., Feldman, M.W.: Evolution of social learning: a mathematical analysis. Theor. Popul. Biol. **66**(3), 249–258 (2004)
9. Le, N., O'Neill, M., Brabazon, A.: The Baldwin effect reconsidered through the prism of social learning. In: IEEE Congress on Evolutionary Computation, CEC 2018, 8–13 Jul forthcoming. IEEE Press, Rio (2018)
10. Hoppitt, W., Laland, K.N.: Social Learning: An Introduction to Mechanisms, Methods, and Models. Princeton University Press, Princeton (2013)
11. Rogers, A.R.: Does biology constrain culture? Am. Anthropol. **90**(4), 819–831 (1988)
12. Richerson, P.J., Boyd, R.: Culture and the Evolutionary Process. University of Chicago Press, Chicago (1985)
13. Suzuki, R., Arita, T.: Repeated occurrences of the Baldwin effect can guide evolution on rugged fitness landscapes. In: 2007 IEEE Symposium on Artificial Life. IEEE, April 2007

A First Analysis of Kernels for Kriging-Based Optimization in Hierarchical Search Spaces

Martin Zaefferer[1(✉)] and Daniel Horn[2]

[1] Institute of Data Science, Engineering, and Analytics,
TH Köln, Steinmüllerallee 6, 51643 Gummersbach, Germany
martin.zaefferer@th-koeln.de
[2] Faculty of Statistics, TU Dortmund University,
Vogelpothsweg 87, 44227 Dortmund, Germany
daniel.horn@tu-dortmund.de

Abstract. Many real-world optimization problems require significant resources for objective function evaluations. This is a challenge to evolutionary algorithms, as it limits the number of available evaluations. One solution are surrogate models, which replace the expensive objective.

A particular issue in this context are hierarchical variables. Hierarchical variables only influence the objective function if other variables satisfy some condition. We study how this kind of hierarchical structure can be integrated into the model based optimization framework. We discuss an existing kernel and propose alternatives. An artificial test function is used to investigate how different kernels and assumptions affect model quality and search performance.

Keywords: Surrogate model based optimization
Hierarchical search spaces · Conditional variables · Kernel

1 Introduction

When objective function evaluations become expensive, surrogate models may be employed to reduce the resource consumption in an optimization process. One challenging issue in this context are *conditional* or *hierarchical* variables. Hierarchical variables are only active (i.e., have an influence on the result) if other variables fulfill certain conditions. This occurs in many algorithm tuning problems. For instance, in machine learning algorithms, parameters of an SVM kernel are only active if that kernel is utilized. Similarly, a variable of a variation operator in an evolutionary algorithm only has an effect if that operator is actually used. Such parameters may also occur in engineering problems. For instance, if a variable defining the amount of energy fed into the system exceeds a certain level, it may require an additional cooling step which itself has variables.

We require tools to model these cases efficiently. In previous studies, three alternatives have been employed: Firstly, the hierarchical nature of a variable

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 399–410, 2018.
https://doi.org/10.1007/978-3-319-99259-4_32

could be ignored and the data handled by standard modeling methods. This approach could be suboptimal since the available information on variable activity is not used. Secondly, a pre-processing step could impute a constant value for the inactive variables, e.g., the mean, or some lower/upper bound [1–3]. We refer to this as the *imputation approach*. Thirdly, the information about hierarchical variables can be incorporated into the modeling process. It can be be integrated into the kernel, e.g., the Arc-kernel [4,5]. In other approaches, Gaussian processes are placed on the leaves of a tree structure that is assumed to represent the hierarchical dependencies of the variables [6–8].

In this article, we focus on kernels in the context of the third case, and propose alternatives to the Arc-kernel. We present a numerical comparison based on a simple test function to verify that the performance of these kernels meets our expectations. We aim to answer the following research questions:

1. Do kernels have to incorporate knowledge about the search space hierarchy?
2. When should which kernel be used?
3. Does definiteness of the kernel play a role?

We give a short introduction to model based optimization in Sect. 2 and to Kriging models in Sect. 3. Afterwards, we introduce kernels for hierarchical search spaces in Sect. 4. We describe our experimental setup in Sect. 5 and analyze the results in Sect. 6. A final evaluation and outlook on future work is given in Sect. 7.

2 Surrogate Model-Based Optimization

Let $f : \mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_d \rightarrow \mathbb{R}$ be a black-box function with a d-dimensional input domain and a deterministic output y. Each $\mathcal{X}_i$ can either be numeric and bounded ($\mathcal{X}_i = [l_i, u_i] \subset \mathbb{R}$) or categorical. We want to solve the optimization problem (OP) and find the input $\boldsymbol{x}^* = \text{argmin}_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x})$. We assume that evaluations of f are expensive, which limits the number of evaluations severely.

Sequential model-based optimization (SMBO) is a state-of-the-art method for solving expensive OPs. It is based on the Efficient Global Optimization (EGO) procedure [9]. First, SMBO samples and evaluates an initial set of candidate solutions. Then, a surrogate regression model is fitted to the data. The model is optimized with respect to an infill criterion in order to find a new, promising candidate $\boldsymbol{x}^*$. The candidate $\boldsymbol{x}^*$ is evaluated with f and added to the data set. This allows to train an improved surrogate model. The procedure iterates until a stopping criterion is reached, e.g., a budget on the number of function evaluations. A detailed introduction is given by Bischl et al. in [10].

Four components of the SMBO procedure have to be specified: the generation of the initial candidate set, the surrogate model, the infill criterion and the optimizer of the infill criterion. We use Latin Hypercube Sampling (LHS), Kriging models, the expected improvement criterion and Differential Evolution (DE) [11]. Our methods can be easily extended to other SMBO variants that employ kernel-based models.

3 Kriging

Frequently, SMBO employs Kriging models, which interpret observations as realizations of a Gaussian process. Forrester et al. [12] give a detailed description. In its core, Kriging models the correlation between observations, e.g., with an exponential correlation function $\mathrm{k}(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\theta \cdot \mathrm{d}(\boldsymbol{x}, \boldsymbol{x}'))$. Here, $\boldsymbol{x}$ and $\boldsymbol{x}'$ are samples, θ is a kernel parameter and $\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}')$ is a distance function, e.g., the Euclidean distance if $\boldsymbol{x}$ is real valued. The correlation matrix $\boldsymbol{K}$ collects all pairwise correlations. Usually, correlation functions should be positive semi-definite (PSD), i.e., all eigenvalues of $\boldsymbol{K}$ are non-negative. The Kriging predictor is:

$$\hat{y}(\boldsymbol{x}) = \hat{\mu} + \boldsymbol{k}^T \boldsymbol{K}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}),$$

where $\mathbf{y}$ are the training observations, $\hat{\mu}$ represents the process mean, $\mathbf{1}$ is a vector of ones and $\boldsymbol{k}$ is the column vector of correlations between the set of training samples $\boldsymbol{X}$ and the new sample $\boldsymbol{x}$. All parameters are usually determined by Maximum Likelihood Estimation (MLE). Kriging is a popular choice in SMBO algorithms, as it provides an estimate of the prediction uncertainty:

$$\hat{s}^2(\boldsymbol{x}) = \hat{\sigma}^2(1 - \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{k}),$$

where the process variance $\hat{\sigma}$ is also determined by MLE. The estimate $\hat{s}^2(\boldsymbol{x})$ can be used to balance exploration and exploitation by computing the Expected Improvement (EI) of candidate solutions [13]. The EI is a frequently employed infill criterion, e.g., in EGO [9].

Kriging also allows to deal with noisy data, using the so called nugget effect. The nugget adds a small constant $\eta > 0$ to the diagonal of $\boldsymbol{K}$. Thus, the otherwise interpolating Kriging model is able to regress the data, introducing additional smoothness into the predicted value. The nugget effect may also help to increase the numerical stability. A re-interpolation approach can be used to avoid that the nugget effect deteriorates the uncertainty estimate [12].

4 Kernels for Hierarchical Search Spaces

Hierarchical variables can be defined as variables that are only *active* if other variables fulfill a condition. An *active* variable has an impact on the objective function value. We use the notation of Hutter and Osborne [4]: a function $\delta_i(\boldsymbol{x})$ determines whether the i-th variable of $\boldsymbol{x}$ is active ($\delta_i(\boldsymbol{x}) = \text{true}$) or not (false). In the following, only the per-variable distance $\mathrm{d}_i(x_i, x_i')$ will be introduced for each kernel. The combined kernel structure is identical for all cases unless stated otherwise, i.e., $\mathrm{k}(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\sum_{i=1}^{d} \mathrm{d}_i(x_i, x_i'))$. We describe an existing kernel (Arc) and propose four alternatives (Ico, IcoCorrected, Imp, ImpArc).

4.1 The Arc-Kernel

The Arc-kernel proposed by Hutter and Osborne [4] is specifically developed to handle hierarchical structures. It is based on three assumptions. First, if a

hierarchical variable is inactive in two configurations $\boldsymbol{x}$ and $\boldsymbol{x}'$, then the distance in that dimension should be zero. Second, if it is active in both configurations, the distance depends on the respective variable values. Third, if the variable is only active in one configuration, the distance should be a constant, because no information is available to compare an inactive with an active variable.

An embedding is required to encode these assumptions in valid distance measures that yield a PSD kernel. It is for continuous variables [4]:

$$\mathrm{d}_{\mathrm{Arc}i}(x_i, x_i') = \begin{cases} 0, & \text{if } \delta_i(\boldsymbol{x}) = \delta_i(\boldsymbol{x}') = \text{false} \\ \theta_i, & \text{if } \delta_i(\boldsymbol{x}) \neq \delta_i(\boldsymbol{x}') \\ \theta_i \sqrt{2 - 2\cos(\pi \rho_i \frac{x_i - x_i'}{u_i - l_i})}, & \text{if } \delta_i(\boldsymbol{x}) = \delta_i(\boldsymbol{x}') = \text{true} \end{cases} \tag{1}$$

The kernel variables $\theta_i \in \mathbb{R}^+$ and $\rho_i \in [0, 1]$ are determined by MLE. A respective measure for categorical variables can be found in [4]. We follow up on [5] and skip the notion of putting further restrictions on θ_i to encode lower importance of lower hierarchical levels as proposed in [4]. Moreover, we use the square of the distance in the embedded space (i.e., removing the square root in Eq. (1)), since we also use squared deviations in all other distances.

4.2 Indefinite Conditional Kernel

We propose a simplified alternative to the Arc-kernel:

$$\mathrm{d}_{\mathrm{Ico}i}(x_i, x_i') = \begin{cases} 0, & \text{if } \delta_i(\boldsymbol{x}) = \delta_i(\boldsymbol{x}') = \text{false} \\ \rho_i, & \text{if } \delta_i(\boldsymbol{x}) \neq \delta_i(\boldsymbol{x}') \\ \theta_i \mathrm{d}_i(x_i, x_i'), & \text{if } \delta_i(\boldsymbol{x}) = \delta_i(\boldsymbol{x}') = \text{true} \end{cases}$$

Here, $\mathrm{d}_i(x_i, x_i')$ is an appropriate default distance (numerical: square deviation $(x_i - x_i')^2$, categorical: Hamming distance). The distance parameter $\rho_i \in \mathbb{R}^+$ is determined by MLE. The kernel follows the same intuitive assumptions as $\mathrm{d}_{\mathrm{Arc}}$, but it does not use the complicated cylindrical embedding. This may lead to indefinite kernel matrices for some data sets or choices of parameters. Due to this, it will be denoted as the indefinite conditional kernel, or Ico-kernel.

As a variant of the Ico-kernel, the IcoCorrected (IcoCor) kernel is the same kernel subject to a correction via a spectrum-flip. This transformation of the eigenspectrum generates PSD kernel matrices from indefinite kernels, cf. [14]. Note, that the nugget effect may also correct issues with definiteness if η is large enough. Thus, even the uncorrected Ico-kernel can produce a valid model.

4.3 Imputation Kernel

Alternatively, we propose a simple PSD kernel. It is based on a different assumption: If the hierarchical variable is only active in one of two configurations ($\delta_i(\boldsymbol{x}) \neq \delta_i(\boldsymbol{x}')$), their distance in that dimension is *not* assumed to be constant. Rather, it is assumed that the value of the active configuration does influence the

dissimilarity. This is achieved by introducing a kernel parameter against which the respective active value is compared. Thus,

$$\mathrm{d}_{\mathrm{Imp}_i}(x_i, x_i') = \begin{cases} 0, & \text{if } \delta_i(\boldsymbol{x}) = \delta_i(\boldsymbol{x}') = \text{false} \\ \theta_i \mathrm{d}_i(x_i', \rho_i), & \text{if } \delta_i(\boldsymbol{x}) = \text{false} \neq \delta_i(\boldsymbol{x}') \\ \theta_i \mathrm{d}_i(x_i, \rho_i), & \text{if } \delta_i(\boldsymbol{x}) = \text{true} \neq \delta_i(\boldsymbol{x}') \\ \theta_i \mathrm{d}_i(x_i, x_i'), & \text{if } \delta_i(\boldsymbol{x}) = \delta_i(\boldsymbol{x}') = \text{true} \end{cases}$$

where d_i is again the appropriate default distance (square deviation, Hamming) and ρ_i is of the same data type as x_i. For real x_i, the bounds of x_i and ρ_i can differ. We use $\rho_i \in [l_i - a, u_i + a] \subset \mathbb{R}$ with $a = 2 * (u_i - l_i)$. Larger bounds may be necessary, depending on the problem. Similarly, if x_i is categorical ρ_i can have one more level (category) than x_i, to emulate the case where none of the other levels is a good replacement. An exponential kernel based on $\mathrm{d}_{\mathrm{Imp}_i}(x_i, x_i')$ can be proven to be PSD. Using proposition 2 in [4], we only need to show that there exists a mapping function $f_i(x_i)$ that maps to a space in which a valid distance can be used, i.e., $\mathrm{d}_{\mathrm{Imp}_i}(x_i, x_i') = \mathrm{d}_i(f_i(x_i), f_i(x_i'))$. For $\mathrm{d}_{\mathrm{Imp}}$, the mapping function is

$$f_i(x_i) = \begin{cases} x_i \text{ if } \delta_i(\boldsymbol{x}) \\ \rho_i \text{ otherwise.} \end{cases}$$

Hence, the resulting kernel based on $\mathrm{d}_i(f_i(x_i), f_i(x_i'))$ is PSD.

Clearly, this kernel has relations to the imputation approach mentioned in Sect. 1. Essentially, inactive values are replaced by an imputed value ρ_i. Instead of choosing that value a-priori, it is defined as a parameter and determined by MLE. Hence, it will be denoted as the imputation kernel or Imp-kernel. One drawback of this kernel is, that if x_i is categorical, ρ_i is also categorical. This may complicate the MLE procedure. Also, the assumption that some value can be imputed is less conservative than the assumptions of the Arc-kernel.

4.4 The Imputation-Arc Kernel

When it is unclear whether the Arc- or Imp-kernel is more appropriate, we suggest a linear combination denoted as the ImpArc-kernel,

$$\mathrm{k}_{\mathrm{ImpArc}}(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\sum_{i=1}^{d} \beta_{1,i} \mathrm{d}_{\mathrm{Arc}\,i}(x_i, x_i') + \beta_{2,i} \mathrm{d}_{\mathrm{Imp}_i}(x_i, x_i')),$$

with weights $\beta_{k,i} \in \mathbb{R}^+$ determined by MLE. Other combinations (e.g., Ico-Imp, Imp-Arc-Ico) are possible. We only test the ImpArc combination, because the Ico- and Arc-kernel express very similar information. Also, a three-way combination would require to learn an additional weight $\beta_{3,i}$.

5 Experimental Setup

While synthetic, tree-based test functions for hierarchical search spaces have been proposed by Jenatton et al. [8], they are not able to respect the different

definitions and assumptions of our kernels. Hence, we suggest a simple two-dimensional quadratic function

$$f(\boldsymbol{x}) = (x_1 - d)^2 + \begin{cases} 0 & \text{if} \quad x_1 \leq c \\ (x_2 - 0.5)^2 + b & \text{else} \end{cases}.$$

The function's behavior (see Fig. 1) is defined by the constants b, c and d. The constant b controls whether the Imp-kernel is a good match, c controls the size of the active region and d controls the location of the optimum. The function is influenced by the hierarchical variable x_2 only if $x_1 > c$ and does have a discontinuity at $x_1 = c$. For $b = 0$, the function is continuous at $x_2 = 0.5$. Hence, the if-else term of $f(\boldsymbol{x})$ yields identical results if $x_2 = 0.5$ and if $\delta_2(\boldsymbol{x}) = false$. In this case, the assumption of the Imp-kernel is fulfilled, i.e., the kernel definition matches the problem structure. The Imp-kernel should learn to impute $\rho = 0.5$.

We identified five situations with different expected performances.

(A) $d < c$ (the optimum is in the *inactive* region of x_2 at $x_1 = d, x_2 \in \mathbb{R}$) and $b = 0$ (imputation potentially *profitable*). The function is *unimodal*.
(B) $d < c$ (the optimum is in the *inactive* region at $x_1 = d, x_2 \in \mathbb{R}$) but $b > 0$ (imputation potentially *unprofitable*). The function is *unimodal*.
(C) $d > c$ (the optimum is in the *active* region at $x_1 = d, x_2 = 0.5$) and $b = 0$ (imputation potentially *profitable*). The function is *bimodal*.
(D) $d > c$ (the optimum is in the *active* region at $x_1 = d, x_2 = 0.5$) and $b = 0.1$ (imputation potentially *unprofitable*) and $b < (c - d)^2$. The function is *bimodal*. The discontinuity at c is not as important, since the optimum is remote from it.
(E) $d > c$ (the optimum is in the *active* region at $x_1 = c, x_2 \in \mathbb{R}$) and $b = 0.1$ (imputation potentially *unprofitable*) and $b > (c - d)^2$. The function is *bimodal*. The discontinuity at c has to be approximated well, since the optimum is at $x_1 = c$.

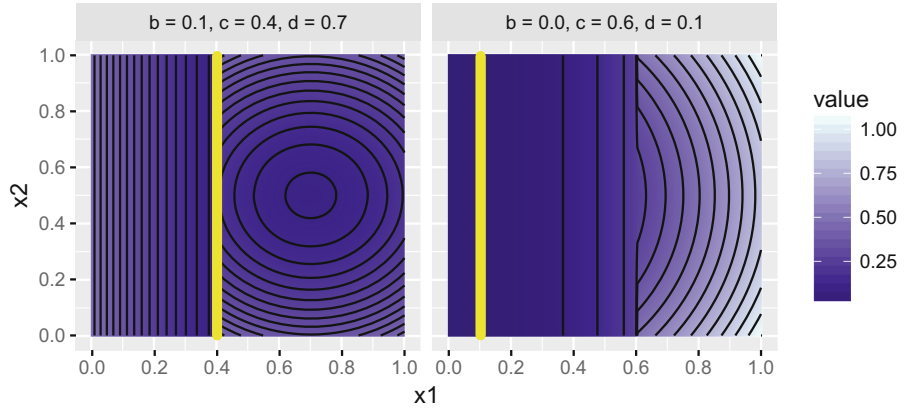


Fig. 1. Visualization of the test function, the optimum is marked in yellow. (Color figure online)

Covering all of these five situations, we tested all combinations of the values $b = \{0, 0.1\}$, $c = \{0.2, 0.4, 0.6, 0.8\}$, and $d = \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

To estimate model quality, we measured the model's Root Mean Squared Error (RMSE). The models were trained with 10, the error was estimated on 1 000 uniform random samples. The Kriging model was trained with the `CEGO` package in R [15,16]. It was configured to use the nugget effect and re-interpolation. The Dividing Rectangles algorithm [17] was chosen to optimize the model parameters via 200 likelihood evaluations. We used all kernels from Sect. 4 and a standard exponential kernel with square deviation in each dimension (which does not incorporate hierarchical information), denoted as the Stan-kernel.

The same type of model was used in the SMBO algorithm from the `CEGO` package. The search was limited to 10 evaluations of $f(\boldsymbol{x})$, due its low difficulty, low dimensionality and assumed cost. The search was initialized with three uniform random samples. Based on the model, the EI criterion was optimized by DE [11]. We used the `DEoptim` package [18] with 10 000 EI evaluations per iteration and used default parameters otherwise. Each experiment was repeated 100 times, with 100 unique random seeds (one per replication). We recorded the difference between the best found and the optimal function value (*suboptimality*) for each replication.

6 Results

First, we analyze the model quality produced by the different kernels. Figure 2 shows the median RMSE value for all parameter constellations and kernels. Clearly, the fit of the Stan-kernel is inferior to most specialized hierarchical kernels for almost all parameter constellations, especially if $b = 0.1$.

If $b = 0$, the assumption of the Imp-kernel is fulfilled. Hence, both the Imp- and the ImpArc-kernel produce a better fit than most other kernels. However, for $b = 0.1$, the Imp-kernel mostly has the second or third worst performance. Only the Stan-kernel and sometimes the IcoCorrected-kernel perform worse. The Arc- and the Ico-kernel achieve very similar performances in most cases, with near-to-best performance if $b = 0.1$. The ImpArc-kernel, combining the advantages of the Arc- and the Imp-kernel, has a good, sometimes best fit in all situation, for both $b \in \{0, 0.1\}$. Contrarily, the IcoCorrected-kernel has a rather poor fit in several cases, sometimes even worse than the Stan-kernel. Overall, differences between kernels tend to disappear for large values of c, which is to be expected due to the reduced influence of the hierarchical variable x_2.

To get a better understanding of the kernels, we visualize an example for Situation E with $(b, c, d) = (0.1, 0.4, 0.7)$. Figure 3 shows line plots for the test function as well as fitted models for all six kernels, trained with ten uniform random samples. Here, the global optimum is at $x_1 = c = 0.4$, i.e., at the jump discontinuity. The function value of the global optimum (0.09) is only slightly better than the value of the local optimum (0.1) at $(x_1 = 0.7, x_2 = 0.5)$. Hence, to find the global optimum, it is important to model the discontinuity well.

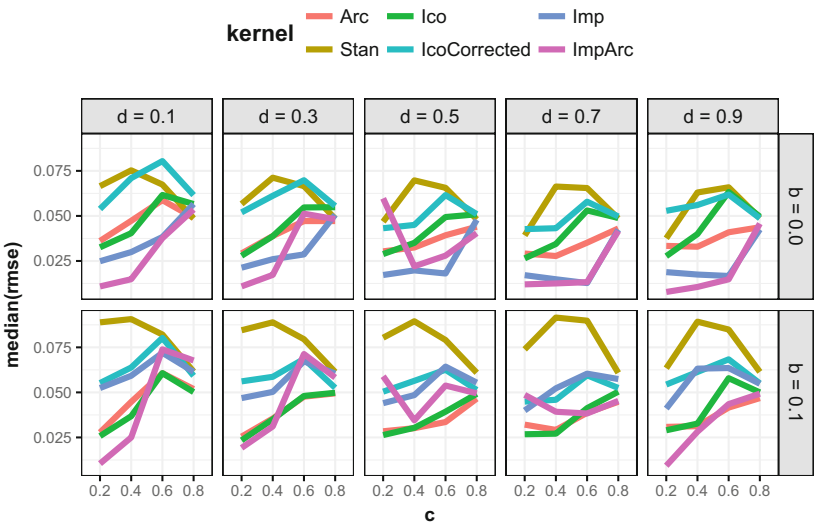


Fig. 2. Median RMSE values over 100 replications for all configurations and kernels.

The Stan-kernel is not able to model the discontinuity and therefore tries to fit a smooth curve to the function. Hence, the Stan-kernel approximates the optimum poorly. For $x_1 = 0.5$ the model of the Imp-kernel shares the poor performance of the Stan kernel: It is not able to fit the discontinuity. Still, the fit is much closer to the true objective function. For $x_1 = 0.75$ the Imp-kernel is able to fit the discontinuity, but the fit is inferior to the Arc-, Ico- and ImpArc-kernel. All of them reproduce the discontinuity quite well. However, their approximation of the function for $x_1 > c, x_2 = 0.75$ has a strong offset. While this is not a perfect fit, it will not necessarily deteriorate optimization performance. The model based on the IcoCorrected-kernel is able to reproduce the discontinuity, but the jump is not large enough to identify the optimum at $x_1 = 0.4$.

Next, we analyze the optimization performance. Due to space restrictions, we present statistical test results that summarize the experimental data. Following Demšar [19], we apply Friedman and corresponding post-hoc Nemenyi tests in order to find significant differences between the kernels, using the function parameters b, c and d as blocking variables for the tests. We extend Demšar's approach, since we do not apply our tests to the median suboptimalities. Instead, we use the replication identifier as an additional blocking variable. This accounts for the effect of the initial design. We visualize the test results using ordered graphs that present a rough order on the kernels.

We start by investigating the combined results of all optimization experiments. With a p-value that is numerically approximating zero ($< 10^{-16}$), the Friedman-test indicates that there are significant differences between the different kernels. Note, if differences are present p-values tend to be small due to the

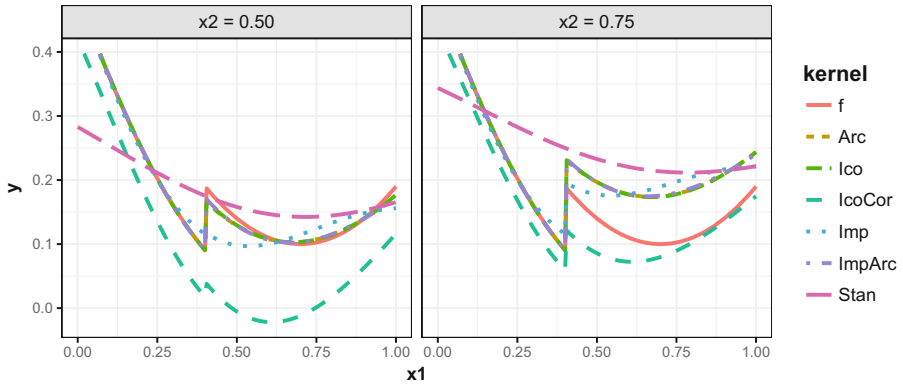


Fig. 3. Example fits of the kernels. Two slice planes are shown for $x_2 \in \{0.5, 0.75\}$.

large number of experiment replications, and differentiating between significant and relevant differences is an open issue in the analyses of computer experiments.

Figure 4 shows the results of the corresponding Nemenyi-test, including a graph representation of the test results as well as mean ranks for each kernel. As expected, the Stan-kernel is clearly outperformed by all other kernels. For the other kernels, we can identify two groups: The Imp- and the ImpArc-kernel seem to perform slightly better than the rest. Within each group, there are no significant differences between the kernels, while tests between kernel from groups are significant. It is questionable how reliable this result is. We expect diverse behavior of the kernels in the five situation and the overall performance is of course influenced by the selection of the specific test instances. Hence, we will now examine individual tests for situations A to E.

As in the global situation, all Friedman-tests result into very small p-values (numerically approximating zero). Hence, there is evidence for significant differences between at least some kernels in each situation. Figure 5 shows the results

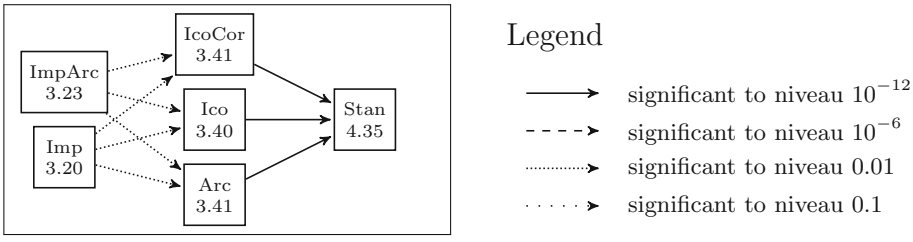


Fig. 4. Ordering of the six kernels with respect to their mean ranks (printed below each kernel) over all test instances. A path (possibly using multiple edges) between two kernels denotes a significant difference of the post-hoc Nemenyi test, the directions of the arrows follows the ordering of the mean ranks.

of the post-hoc Nemenyi-tests in all five situations. In situation A and C, the assumption of the Imp-kernel is fulfilled, since $b = 0$ allows for imputation. This is reflected by the results: In both situations A and C the Imp-kernel performs best. The Imp-kernel outperforms the Arc- and Ico-kernel with a large margin in situation C. Contrarily, in situation B (unimodal, not imputable) and E (bimodal, not imputable), where the imputation assumption is violated, the Imp-kernel performance is inferior. These observations fit to our expectations: b controls whether or not the Imp-kernel is able to find a good value to impute.

The Arc- and the Ico-kernel have similar results in most situations, except for situation C. This confirms that these kernels encode similar information, and it also shows that the indefiniteness of the Ico-kernel does not seem to impact optimization performance. At least, the indefiniteness is sufficiently well mitigated by the employed nugget effect.

While performing reasonably well, the ImpArc-kernel never achieves a top performance. It is usually positioned in the second-best group. This can be explained by the fact that it attains some middle ground between the kernels

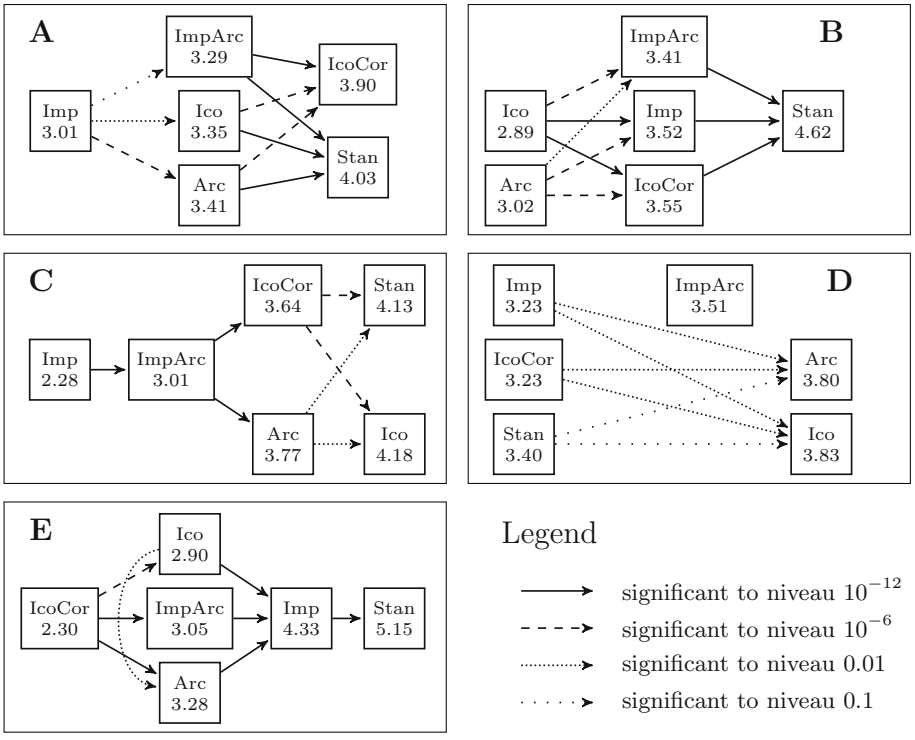


Fig. 5. Ordering of the kernels in the five situations with respect to their mean ranks (printed below each kernel) over all test instances. A path (possibly using multiple edges) between two kernels denotes a significant difference of the post-hoc Nemenyi test the directions of the arrows follows the ordering of the mean ranks.

that it combines. The IcoCorrected-kernel performs poorly in some situations (A, B, C), but it performs best in situation E. Poor performance may be caused by inconsistencies in the employed definiteness repair methods. However, it remains unclear to us why the performance is distinctively better in situation E.

Situation D (bimodal, not imputable) has a rather special behavior. Only few distinct differences between the kernels can be detected. Moreover, it is the only situation in which the Stan kernel does not perform in the worst group. We suggest that this is due to the fact that modeling the discontinuity is not as important here. The optimum lies in the region where x_2 is active, hence it may even be detrimental to model the discontinuity. That means, if the optimum is far enough from the discontinuity, it may be helpful to smoothen through the local optimum that lies at the discontinuity, since this will drive the search towards the global optimum. This could also explain why the Imp-kernel outperforms the Arc-kernel in situation D, despite $b = 0.1$.

7 Conclusion and Outlook

We investigated different kernels for SMBO in hierarchical search spaces, e.g., the Arc-kernel previously proposed by Hutter and Osborne [4], the Ico-kernel which is similar, yet indefinite, and the Imp-kernel which attempts to learn suitable imputed values for inactive variables. We tested both the model quality and the optimization performance of six kernels, and received consistent results. Hence, we can answer our research questions and deduct simple recommendations for choosing a kernel.

1. The hierarchical structure should be incorporated into the kernel.
2. The Imp-kernel should be chosen if it is a-priori known that its assumption is fulfilled. If the assumption is violated, the Arc- and Ico-kernel are good choices. Without prior knowledge, the ImpArc-kernel is a sound compromise.
3. We did not observe many significant differences between the Arc- and the Ico-kernel. The kernels' definiteness does not seem to have a strong impact.

These result rely on tests with a rather simple test function, and hence have to interpreted with care. Devising more complex test functions with higher input dimensions is clearly of interest. But while artificial tests are instructive due to their controlled behavior, it is not always clear how this translates to real world problems. Hence, it would be desirable to make tests with real world applications, such as algorithm tuning.

Furthermore, it would be interesting to let the infill optimizer exploit the information on variable activity, to avoid searching in inactive areas of the search space. The same is true for the initialization of the SMBO algorithm. Spreading a space-filling design in inactive areas is wasteful.

Finally, all discussed distances $\mathrm{d}_i(x_i, x_i')$ are defined for a single dimension i. Therefore, we are not limited to a single choice. Rather, different distances can be chosen for each dimension (e.g., Arc for x_i, and Imp for $x_{j \neq i}$).

References

1. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA. In: Proceedings of the 19th International Conference on Knowledge Discovery and Data Mining. ACM Press (2013)
2. Horn, D., Bischl, B.: Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI) (2016)
3. Cáceres, L.P., Bischl, B., Stützle, T.: Evaluating random forest models for irace. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM Press (2017)
4. Hutter, F., Osborne, M.A.: A kernel for hierarchical parameter spaces. Technical report arXiv:1310.5738, arXiv (2013)
5. Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.: Raiders of the lost architecture: kernels for Bayesian optimization in conditional parameter spaces. In: NIPS workshop on Bayesian Optimization in Theory and Practice (2013)
6. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, vol. 24. Curran Associates, Inc. (2011)
7. Bergstra, J., Yamins, D., Cox, D.: Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of the 30th International Conference on Machine Learning, PMLR (2013)
8. Jenatton, R., Archambeau, C., González, J., Seeger, M.: Bayesian optimization with tree-structured dependencies. In: Proceedings of the 34th International Conference on Machine Learning, PMLR (2017)
9. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Glob. Optim. **13**(4), 455–492 (1998)
10. Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., Lang, M.: mlrMBO: a modular framework for model-based optimization of expensive black-box functions. arXiv preprint arXiv:1703.03373 (2017)
11. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
12. Forrester, A., Sobester, A., Keane, A.: Engineering Design Via Surrogate Modelling. Wiley, Hoboken (2008)
13. Mockus, J., Tiesis, V., Zilinskas, A.: The application of Bayesian methods for seeking the extremum. In: Towards Global Optimization, North-Holland, vol. 2 (1978)
14. Zaefferer, M., Bartz-Beielstein, T.: Efficient global optimization with indefinite kernels. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 69–79. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_7
15. Zaefferer, M.: Combinatorial efficient global optimization in R - CEGO v2.2.0 (2017). https://cran.r-project.org/package=CEGO. Accessed 10 Jan 2018
16. Zaefferer, M., Stork, J., Friese, M., Fischbach, A., Naujoks, B., Bartz-Beielstein, T.: Efficient global optimization for combinatorial problems. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM (2014)
17. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. J. Optim. Theory Appl. **79**(1), 157–181 (1993)
18. Mullen, K., Ardia, D., Gil, D., Windover, D., Cline, J.: DEoptim: an R package for global optimization by differential evolution. J. Stat. Softw. **40**(6), 1–26 (2011)
19. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)

Challenges in High-Dimensional Reinforcement Learning with Evolution Strategies

Nils Müller and Tobias Glasmachers(✉)

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

Abstract. Evolution Strategies (ESs) have recently become popular for training deep neural networks, in particular on reinforcement learning tasks, a special form of controller design. Compared to classic problems in continuous direct search, deep networks pose extremely high-dimensional optimization problems, with many thousands or even millions of variables. In addition, many control problems give rise to a stochastic fitness function. Considering the relevance of the application, we study the suitability of evolution strategies for high-dimensional, stochastic problems. Our results give insights into which algorithmic mechanisms of modern ES are of value for the class of problems at hand, and they reveal principled limitations of the approach. They are in line with our theoretical understanding of ESs. We show that combining ESs that offer reduced internal algorithm cost with uncertainty handling techniques yields promising methods for this class of problems.

1 Introduction

Since the publication of DeepMind's Deep-Q-Learning system [18] in 2015, the field of (deep) reinforcement learning (RL) [34] is developing at a rapid pace. In [18] neural networks learn to play classic Atari 2600 games solely from interaction, based on raw (unprocessed) visual input. The approach had a considerable impact because it demonstrated the great potential of deep reinforcement learning. Only one year later AlphaGo [8] demystified the ancient game of Go by beating multiple human world experts. In this rapidly moving field, Evolution Strategies (ESs) [4,22,30] have gained considerable attention by the machine learning community when OpenAI promoted them as a "scalable alternative to reinforcement learning" [17], which spawned several follow-up works [6,9].

Already long before deep RL, controller design with ESs was studied for many years within the domain of neuroevolution [16,23,24,29,32,33]. The optimization of neural network controllers is frequently cast as an episodic RL problem, which can be solved with direct policy search, for example with an ES. This amounts to parameterizing a class of controllers, which are optimized to maximize reward or to minimize cost, determined by running the controller on the task at hand, often in a simulator. The value of the state-of-the-art covariance

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 411–423, 2018.
https://doi.org/10.1007/978-3-319-99259-4_33

matrix adaptation evolution strategy (CMA-ES) algorithm [22] for this problem class was emphasized by several authors [16,23]. CMA-ES was even augmented with an uncertainty handling mechanism, specifically for controller design [14].

The controller design problems considered in the above-discussed papers are rather low-dimensional, at least compared to deep learning models with up to millions of weights. CMA-ES is rarely applied to problems with more than 100 variables. This is because learning a full covariance matrix introduces non-trivial algorithm internal cost and hence prevents the direct application of CMA-ES to high-dimensional problems. In recent years it turned out that even covariance matrix adaptation can be scaled up to very large dimensions, as proven by a series of algorithms [1,11,28,31], either by restricting the covariance matrix to the diagonal, to a low-rank model, or to a combination of both. Although apparently promising, none of these approaches was to date applied to the problem of deep reinforcement learning.

Against this background, we investigate the suitability of evolution strategies in general and of modern scalable CMA-ES variants in particular for the design of large-scale neural network controllers. In contrast to most existing studies in this domain, we approach the problem from an optimization perspective, not from a (machine) learning perspective. We are primarily interested in how different algorithmic components affect optimization performance in high-dimensional, noisy optimization problems. Our results provide a deeper understanding of relevant aspects of algorithm design for deep neuroevolution.

The rest of the paper is organized as follows. After a brief introduction to controller design we discuss mechanisms of evolution strategies in terms of convergence properties. We carry out experiments on RL problems as well as on optimization benchmarks, and close with our conclusions.

2 Problems Under Study

General Setting. In this paper, we investigate the utility of evolution strategies for optimization problems that pose several difficulties at the same time:

- a large number d of variables (high dimension of the search space $\mathbb{R}^d$),
- fitness noise, i.e., the variability of fitness values $f(x)$ when evaluating the non-deterministic fitness function multiple times in the same point x, and
- multi-modality, i.e., the presence of a large number of local optima.

Additionally, a fundamental requirement of relatively quick problem evaluation time (typically requiring simulation of real world phenomena) is appropriate.

State-of-the-art algorithms like CMA-ES can handle dimensions of up to $d \leq 100$ with ease. They become painfully slow for $d \geq 1000$ due to their time and memory requirements. In this sense, a high-dimensional problem is characterized by $d \geq 1000$. Yet, recent advances led to the successful application of ESs with a diagonal and/or low-rank model of the covariance matrix to up to 500,000-dimensional (noise-free) problems [28]. Mainly fueled by the reduction of internal algorithm cost, modern ESs thereby become applicable to new classes of

problems. Deep reinforcement learning problems present such a new challenge, characterized by a combination of three aspects, namely high search space dimension, fitness noise, and multi-modality. While neural networks are known to give rise to highly multi-modal landscapes, several recent studies suggest that many if not all local optima are of good or even equal quality [27]. Furthermore, the problem can be addressed effectively with simple generic methods like restarts. Therefore we focus on the less well understood interaction of noise and high dimensions. As several components of modern ESs are impaired by uncertainty and sparsity in sampling, their merit—especially as with increasing dimension the relative share of function evaluations becomes prevalent in time—needs to be assessed. To this end, we draw from previous work on uncertainty handling [3,14,36] in order to face fundamental challenges like a low signal-to-noise ratio.

Despite the greater generality of the described problem setting, a central motivation for studying the above problem class is controller design. In evolutionary controller design, an individual (a candidate controller) is evaluated in a Monte Carlo manner, by sampling its performance on a (simulated) task, or a set of tasks and conditions. Stochasticity caused by random state transitions and randomized controllers is a common issue. Due to complex and stochastic controller-environment interactions, controller design is considered a difficult problem, and black-box approaches like ESs are well suited for the task, in particular, if gradients are not available.

Reinforcement Learning. In reinforcement learning, control problems are typically framed as stochastic, time-discrete, Markov decision processes $(S, A, P_{\cdot,\cdot}(\cdot), R_{\cdot}(\cdot,\cdot), \gamma)$ with the notion of a (software) agent embedded in an environment. The agent ought to take an action $a \in A$ when presented with a state $s \in S$ of the environment in order to receive a reward $(s, s', a) \mapsto R_s(s', a) \in \mathbb{R}$ for a resulting state transition to new state $s' \in S$ in the next time step. An individual encodes a (possibly randomized) controller or policy $\pi_\theta : S \to A$ with parameters $\theta \in \Theta$, which is followed by the agent. It is assumed that each policy yields a constant expected cumulative reward over a fixed number of actions τ taken when acting according to it, as the state transition probability $(s, s', a) \mapsto P_{s,a}(s') = Pr(\mathbf{s}' = s' | \mathbf{s} = s, \mathbf{a} = a)$, to a successor state s' is stationary (time-independent) and depends only on the current state and action (Markov property), for all $s, s' \in S, a \in A$. This cumulative reward acts as a fitness measure $F_\pi : \Theta \to \mathbb{R}$, while the policy parameters θ (e.g., weights of neural networks π_θ) are the variables of the problem. Thus, we consider the (reinforcement learning) optimization problem

$$\min_{\theta \in \Theta} \; F_\pi(\theta) = - \sum_{s_0, \ldots, s_\tau \in S} \left(\sum_{k=0}^{\tau-1} \gamma^k R_{s_k}(s_{k+1}, \pi_\theta(s_k)) \right) \cdot \left(\prod_{j=0}^{\tau-1} P_{s_j, \pi_\theta(s_j)}(s_{j+1}) \right),$$

where $\gamma \in (0, 1]$ is a discount factor.

Developments in RL demonstrated the merit in utilizing "model-free" approaches to the design of high-dimensional controllers such as neural

Algorithm 1. Generic Evolution Strategy Template

initialize λ, $m \in \mathbb{R}^d$, $\sigma > 0$, $C = I$
repeat
  repeat
    for $i \leftarrow 1, \ldots, \lambda$ **do**
      sample offspring $x_i \sim \mathcal{N}(m, \sigma^2 C)$
      evaluate fitness $f(x_i)$ by testing the controller encoded by x_i on the task
    actual optimization: update m
    step size control: update σ
    covariance matrix adaptation: update C
    uncertainty handling, i.e., adapt λ or the number of tests per fitness evaluation
  until *stopping criterion is met*
  prepare restart, i.e., set new initial m, σ, and λ, and reset $C \leftarrow I$
until *budget is used up*
return m

networks for solving a variety of tasks previously inaccessible [8,18], as well as novel frameworks for scaling evolution strategies to CPU clusters [17].

ESs have advantages and disadvantages compared to alternative approaches like policy gradient methods. Several mechanisms of ESs add robustness to the search. Modeling distributions over policy parameters as done explicitly in natural evolution strategies (NES) [7] and also in CMA-ES serves this purpose [12], and so do problem-agnostic algorithm design and strong invariance properties. Direct policy search does not suffer from the temporal credit assignment problem or from sparse rewards [17]. ESs have demonstrated superior exploration behavior, which is important to avoid a high bias when sampling the environment [13]. On the contrary, ESs ignore the information contained in individual state transitions and rewards. This inefficiency can (partly) be compensated by better parallelism in ESs [17].

3 Evolution Strategies

In this section, we discuss Evolution Strategies (ESs) from a bird's eye perspective, in terms of their central algorithmic components, and without resorting to the details of their implementation. For actual exemplary algorithm instances with the properties under consideration, we refer to the literature. Algorithm 1 summarizes commonly found mechanisms without going into any details.

ESs enjoy many invariance properties. This is generally considered a sign of good algorithm design: due to their rank-based processing of fitness values, they are invariant to strictly monotonically increasing transformations of fitness; furthermore, they are invariant to translation, rotation, and scaling provided that the initial distribution is transformed accordingly, and with CMA (see below) they are even asymptotically invariant under arbitrary affine transformations.

Step Size Control. The algorithms applied to RL problems in [6,9,17] are designed in the style of non-adaptive algorithms, i.e., applying a mutation distribution with fixed parameters σ and C, adapting only the center m. This method is known to converge as slowly as pure random search [21]. Therefore it is in general imperative to add step size adaptation, which has always been an integral mechanism since the inception of the method [4,30]. Cumulative step size adaptation (CSA) is a state-of-the-art method [22]. Step size control enables linear convergence on scale invariant (e.g., convex quadratic) functions, and hence locally linear convergence into twice continuously differentiable local optima [21], which puts ESs into the same class as many gradient-based methods. It was shown in [35] that convergence of rank-based algorithms cannot be faster than linear. However, the convergence rate of a step size adaptive ESs is of the form $\mathcal{O}(1/(kd))$, where d is the dimensionality of the search space and k is the condition number of the Hessian in the optimum. In contrast, the convergence rate of gradient descent suffers from large k, but is independent of the dimension d.

Metric Learning. Metric adaptation methods like CMA-ES [5,7,22] improve the convergence rate to $\mathcal{O}(1/d)$ by adapting not only the global step size σ but also the full covariance matrix C of the mutation distribution. However, estimating a suitable covariance matrix requires a large number of samples, so that fast progress is made only after $\mathcal{O}(d^2)$ fitness evaluations, which is in itself prohibitive for large d. Also, the algorithm internal cost for storing and updating a full covariance matrix and even for drawing a sample from the distribution is at least of order $\mathcal{O}(d^2)$, which means that modeling the covariance matrix quickly becomes the computational bottleneck, in particular if the fitness function scales linear with d, as it is the case for neural networks.

Several ESs for large-scale optimization have been proposed as a remedy [1,11,20,28,31]. They model only the diagonal of the covariance matrix and/or interactions in an $\mathcal{O}(1)$ to $\mathcal{O}(\log(d))$ dimensional subspace, achieving a time and memory complexity of $\mathcal{O}(d)$ to $\mathcal{O}(d \log(d))$ per sample. The aim is to offer a reasonable compromise between ES-internal and external (fitness) complexity while retaining most of the benefits of full covariance matrix adaptation. The LM-MA-ES algorithm [11] offers the special benefit of adapting the fastest evolving subspace of the covariance matrix with only $\mathcal{O}(d)$ samples, which is a significant speed-up over the $\mathcal{O}(d^2)$ sample cost of full covariance matrix learning.

Noise Handling. Evolution strategies can be severely impaired by noise, in particular when it interferes with step size adaptation. Being randomized algorithms, ESs are capable of tolerating some level of noise with ease. In the easy-to-analyze multiplicative noise model [26], the noise level decays as we approach the optimum and hence, on the sphere function $f(x) = \|x\|^2$, the signal-to-noise ratio (defined as the systematic variance of f due to sampling divided by the variance of random noise) oscillates around a positive constant (provided that step size adaptation works as desired [25], keeping σ roughly proportional to $\|m\|/d$). For strong noise, this ratio is small. Then the ES essentially performs

a random walk, and a non-elitist algorithm may even diverge. Then CSA is endangered to converge prematurely [2]. For more realistic additive noise, the noise variance is (lower bounded by) a positive constant. When converging to the optimum, σ and hence the signal-no-noise-ratio decays to zero. Therefore progress stalls at some distance to the optimum. Thus there exists a principled limitation on the precision to which an optimum can be located. Explicit noise handling mechanisms like [3,14,36] can be employed to increase the precision, and even enable convergence, e.g., by adaptively increasing the population size or the number of independent controller test runs per fitness evaluation. They adaptively increase the population size or the number of simulation runs per fitness evaluation, effectively improving the signal-to-noise ratio. The algorithm parameters can be tuned to avoid premature convergence of CSA. However, the convergence speed is so slow that in practice additive noise imposes a limit on the attainable solution precision, even if the optimal convergence rate is attained [3].

Noise in High Dimensions. There are multiple ways in which optimization with noise and in high dimensions interact. In the best case, adaptation slows down due to reduced information content per sample, which is the case for metric learning. The situation is even worse for step size adaptation: for the noise-free sphere problem, the optimal step size σ is known to be proportional to $\|m\|/d$. Therefore, in the same distance to the optimum and for the same noise strength, noise handling becomes harder in high dimensions. Then the step size can become too small, and CSA can converge prematurely [2].

4 Experiments

Most of the theoretical arguments brought forward in the previous section are of asymptotic nature, while sometimes practice is dominated by constant factors and transient effects. Also, it remains unclear which of the different effects like slow convergence, slow adaptation, and the difficulty of handling noise is a critical factor. In this section, we provide empirical answers to these questions.

Well-established benchmark collections exist in the evolutionary computation domain, in particular for continuous search spaces [15,19]. Typical questions are whether an algorithm can handle non-separable, ill-conditioned, multi-modal, or noisy test functions. However, it is not a priori clear which of these properties are found in typical controller design problems. For example, the optimization landscapes of neural networks are not yet well understood. Closing this gap is far beyond the scope of this paper. Here we pursue a simpler goal, namely to identify the most relevant factors. More concretely, we aim to understand in which situation (dimensionality and noise strength) which algorithm component (as discussed in the previous section) has a significant impact on optimization performance, and which mechanisms fail to pay off.

To this end, we run different series of experiments on established benchmarks from the optimization literature and from the RL domain. We have published code for reproducing all experiments online.[1] For ease of comparison, we use

[1] https://github.com/NiMlr/High-Dim-ES-RL.

the recently proposed MA-ES algorithm [5] adapting the full covariance matrix, which was shown empirically to perform very similar to CMA-ES. This choice is motivated by its closeness to the LM-MA-ES method [11], which learns a low-rank approximation of the covariance matrix. When disabling metric learning entirely in these methods, we obtain a rather simple ES with CSA, which we include in the comparison.

Figure 1 shows the time evolution of the fitness $F_\pi(\theta)$ (Eq. (2)) on three prototypical benchmark problems from the OpenAI Gym environment [10], a collection of RL benchmarks: acrobot, bipedal walker, and robopong. All three controllers π_θ (Eq. (2)) are fully connected deep networks with hidden layer sizes 30-30-10 (acrobot) and 30-30-15-10 (bipedal walker and robopong), giving rise to moderate numbers of weights around 2,000, depending on the task-specific numbers of inputs and controls. It is apparent that in all three cases fitness noise plays a key role.

Figure 2 investigates the scaling of LM-MA-ES and MA-ES with problem dimension on the bipedal walker task. For the small network considered above, MA-ES performs considerably worse than LM-MA-ES, not only in wall clock time (not shown) but also in terms of sample complexity. A similar effect was observed in [11] for the Rosenbrock problem. This indicates that LM-MA-ES can profit from its fast adaptation to the most prominent subspace. However, this effect does not necessarily generalize to other tasks. More importantly, we see (unsurprisingly) that the performance of both algorithms is severely affected as d grows.

In order to gain a better understanding of the effect of fitness noise on high-dimensional controller design, we consider optimization benchmarks. These problems have the advantage that the optimum is known and that the noise strength is under our control. Since we are particularly interested in scalable metric learning, we employ the noisy ellipsoid problem $f(x) = \bar{f}(x) + N(x)$, $\bar{f}(x) = \sqrt{x^T H x}$, with eigenvalues $\lambda_i = k^{\frac{i-1}{d-1}}$ of H, and $N(x)$ is the noise. For the multiplicative case, the range of $N(x)$ is proportional to $\bar{f}(x)$, while for the additive case it is not.

Among the problem parameters we vary

- problem dimension $d \in \{20, 200, 2000, 20000\}$,
- problem conditioning ($k \in \{10^0, 10^2, 10^6\}$ (sphere, benign ellipsoid, standard ellipsoid), and
- noise strength (none, multiplicative with various constants of proportionality, additive).

Figure 3 shows the time evolution of fitness and step size of the different algorithms in these conditions.

The experiments on the noise-free sphere problem show that the speed of optimization decays with increasing dimension, as predicted by theory [25]: halving the distance to the optimum requires $\Theta(d)$ samples. For this reason, within the fixed budget of 10^6 function evaluations, there is less progress in higher dimensions. For $d = 20,000$, the solution quality is still improved by a factor of

about 10^3, which requires the step size to change by the same amount. However, extrapolating our results we see that in extremely high dimensions the algorithm is simply not provided enough generations to make sufficient progress in order to justify step size adaptation. This is in accordance with [6]. A similar effect is observed for metric learning, which takes $\Theta(d^2)$ samples for the full covariance matrix. Even for the still moderate dimension of $d = 2,000$, the adaptation process is not completed within the given budget. Yet, also during the transitional phase where the matrix is not yet fully adapted, MA-ES already has an edge over the simple ES. LM-MA-ES is sometimes better and sometimes worse than MA-ES. It may profit from the significantly smaller number of parameters in the low-rank covariance matrix, which allows for faster adaptation, in particular in high dimensions, where MA-ES does not have enough samples to complete its learning phase. In any case, its much lower internal complexity allows us to scale up LM-MA-ES to much higher dimensions.

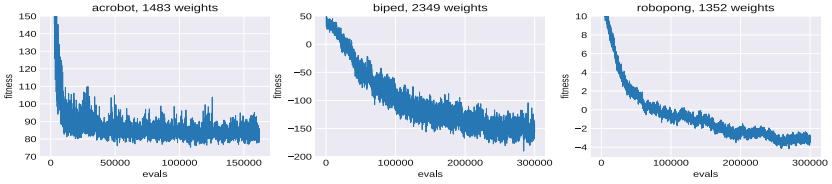


Fig. 1. Evolution of population average fitness for three reinforcement learning tasks with LM-MA-ES, averaged over five runs.

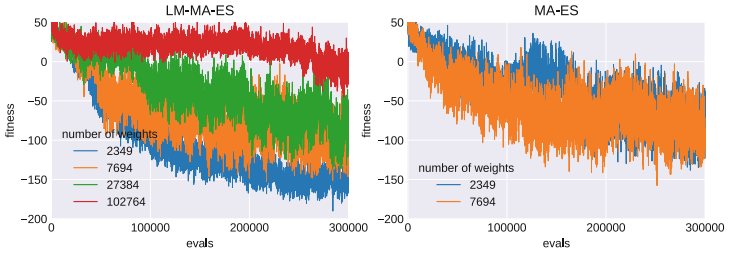


Fig. 2. Evolution of fitness or neural networks with different numbers of weights (different hidden layer sizes), for LM-MA-ES (left) and MA-ES (right) on the bipedal walker task.

In summary, metric adaptation is still useful for problems with a "realistic" dimension of even very detailed controller design problems in engineering, while it is too slow for training neural networks with millions of weights, unless the budget grows at least linear with the number of weights. This in turn requires extremely fast simulations as well as a large amount of computational hardware resources.

Noise has a significant impact on the optimization behavior and on the solution quality. Additive noise implies extremely slow convergence, and indeed we

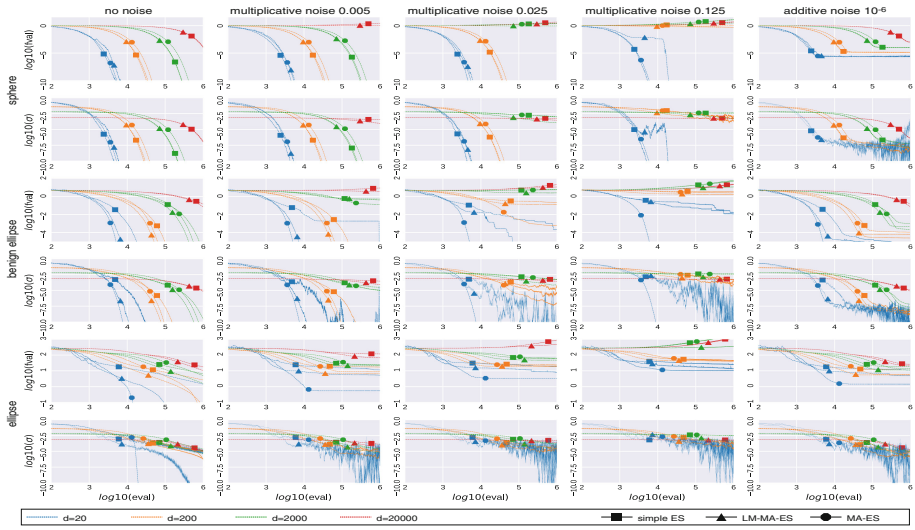


Fig. 3. Evolution of fitness and step size over function evaluations, averaged over five independent runs, for three different algorithms and problems (see the legend for details). Note the logarithmic scale of both axes.

find that all methods stall in this case. Too strong multiplicative noise even results in divergence. A particularly adversarial effect is that the noise strength that can be tolerated is at best inversely proportional to the dimension. This effect nicely shows up in the noisy sphere results. Here, uncertainty handling can help in principle, since it improves the signal-to-noise adaptively to the needs of the algorithm, but at the cost of more function evaluations per generation, which amplifies the effects discussed above.

In the presence of noise, CSA does not seem to work well in low dimensions. In case of high noise, $\log(\sigma)$ performs a random walk. However, this walk is subject to a selection bias away from high values, since they improve the signal-to-noise ratio. Therefore we find extended periods of stalled progress, in particular for $d = 20$, accompanied by a random walk of the (far too small) step size. The effect is unseen in higher dimensions, probably due to the smaller update rate.

We are particularly interested in the interplay between metric adaptation and noise. It turns out that in all cases where CMA helps (non-spherical problems of moderate dimension), i.e., where LM-MA-ES and MA-ES outperform the simple ES, the same holds true for the corresponding noisy problems. We conclude that metric learning still works well, even when faced with noise in high dimensions.

The influence of noise can be controlled and mitigated with uncertainty handling techniques [3,14,36]. This essentially results in curves similar to the leftmost column of Fig. 3, but with slower convergence, depending on the noise strength. In controller design, noise handling can be key to success, in particular if the optimal controller is nearly deterministic, while strong noise is encountered

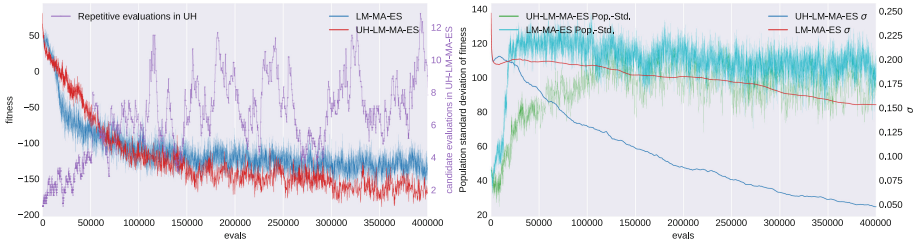


Fig. 4. Fitness and number of re-evaluations (left) step size and standard deviation of fitness (right), averaged over six runs of LM-MA-ES with and without uncertainty handling on the bipedal walker task.

during learning. This is a plausible assumption for the bipedal walker task: at an intermediate stage, the walker falls over randomly depending on minor details of the environment, resulting in high noise variance, while a controller that has learned a stable and robust walking pattern achieves good performance with low variance. Then it is key to handle the early phase by means of uncertainty handling, which enables the ES to enter the late convergence phase eventually. Figure 4 displays such a situation for the benign ellipse with $d = 100,000$ with additive noise applied only for function values above a threshold. LM-MA-ES without uncertainty handling fails, but with uncertainty handling the algorithm finally reaches the noise-free region and then converges quickly.

Figure 5 shows the effect of uncertainty handling. It yields significantly more stable optimization behavior in two ways: 1. it keeps the step size high, avoiding an undesirable decay and hence the danger of premature convergence or of a less-robust population, and 2. it keeps the fitness variance small, which allows the algorithm to reach better fitness in the late fine tuning phase. Interestingly, the ES without uncertainty handling is initially faster. This can be mitigated by tuning the initial step size, which anyway becomes an increasingly important task in high dimensions, for two reasons: adaptation takes long in high dimensions, and even worse, a too small initial step size makes uncertainty handling kick in without need, so that the adaptation takes even longer. The latter might especially be called for on expensive problems commonly found in RL.

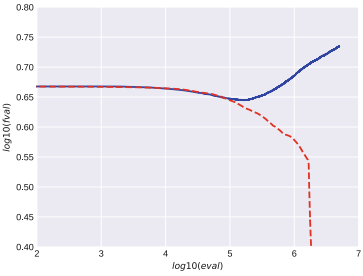


Fig. 5. (UH-) LM-MA-ES on the benign ellipse in $d = 100,000$ with additive noise restricted to $\bar{f}(x) > 3.5$. LM-MA-ES without uncertainty handling (blue curve) diverges while LM-MA-ES with uncertainty handling approaches the optimum (red curve). (Color figure online)

5 Conclusion

We have investigated the utility of different algorithmic mechanisms of evolution strategies for problems with a specific combination of challenges, namely high-dimensional search spaces and fitness noise. The study is motivated by a broad class of problems, namely the design of flexible controllers. Reinforcement learning with neural networks yields some extremely high-dimensional problem instances of this type.

We have argued theoretically and also found empirically that many of the well-established components of state-of-the-art methods like CMA-ES and scalable variants thereof gradually lose their value in high dimensions, unless the number of function evaluations can be scaled up accordingly. This affects the adaptation of the covariance matrix, and in extremely high-dimensional cases also the step size. This somewhat justifies the application of very simple algorithms for training neural networks with millions of weights, see [6].

Additive noise imposes a principled limitation on the solution quality. However, it turns out that adaptation of the search distribution still helps, because it allows for a larger step size and hence a better signal-to-noise ratio. Unsurprisingly, uncertainty handling can be a key technique for robust convergence.

Overall, we find that adaptation of the mutation distribution becomes less valuable in high dimensions because it kicks in only rather late. However, it never harms, and it can help even when dealing with noise in high dimensions. Our results indicate that a scalable modern evolution strategy with step size and efficient metric learning equipped with uncertainty handling is the most promising general-purpose technique for high-dimensional controller design.

References

1. Akimoto, Y., Auger, A., Hansen, N.: Comparison-based natural gradient optimization in high dimension. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 373–380. ACM (2014)
2. Beyer, H.-G., Arnold, D.V.: Qualms regarding the optimality of cumulative path length control in CSA/CMA-evolution strategies. Evol. Comput. **11**(1), 19–28 (2003)
3. Beyer, H.-G., Hellwig, M.: Analysis of the pcCMSA-ES on the noisy ellipsoid model. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 689–696. ACM (2017)
4. Beyer, H.-G., Schwefel, H.-P.: Evolution strategies-a comprehensive introduction. Nat. Comput. **1**(1), 3–52 (2002)
5. Beyer, H.-G., Sendhoff, B.: Simplify your covariance matrix adaptation evolution strategy. IEEE Trans. Evol. Comput. **21**(5), 746–759 (2017). https://ieeexplore.ieee.org/document/7875115/
6. Chrabaszcz, P., Loshchilov, I., Hutter, F.: Back to basics: benchmarking canonical evolution strategies for playing atari. Technical report 1802.08842, arXiv.org (2018)
7. Wierstra, D.: Natural evolution strategies. J. Mach. Learn. Res. **15**(1), 949–980 (2014)

8. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
9. Such, F., et al.: Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. Technical report 1712.06567, arXiv.org (2017)
10. Brockman, G., et al.: OpenAI gym. Technical report 1606.01540, arXiv.org (2016)
11. Loshchilov, I., et al.: Limited-memory matrix adaptation for large scale black-box optimization. Technical report 1705.06693, arXiv.org (2017)
12. Lehman, J., et al.: ES is more than just a traditional finite-difference approximator. Technical report 1712.06568v2, arXiv.org (2017)
13. Plappert, M., et al.: Parameter space noise for exploration. Technical report 1706.01905v2, arXiv.org (2017)
14. Hansen, N., et al.: A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Trans. Evol. Comput. **13**(1), 180–197 (2009)
15. Hansen, N., et al.: COCO: a platform for comparing continuous optimizers in a black-box setting. Technical report 1603.08785, arXiv.org (2016)
16. Geijtenbeek, T., et al.: Flexible muscle-based locomotion for bipedal creatures. ACM Trans. Graph. (TOG) **32**(6), 206 (2013)
17. Salimans, T., et al.: Evolution strategies as a scalable alternative to reinforcement learning. Technical report 1703.03864, arXiv.org (2017)
18. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
19. Li, X., et al.: Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. Gene **7**(33), 8 (2013)
20. Sun, Y., et al.: A linear time natural evolution strategy for non-separable functions. In: Conference Companion on Genetic and Evolutionary Computation. ACM (2013)
21. Hansen, N., Arnold, D.V., Auger, A.: Evolution strategies. In: Kacprzyk, J., Pedrycz, W. (eds.) Springer Handbook of Computational Intelligence, pp. 871–898. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-43505-2_44
22. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution atrategies. Evol. Comput. **9**(2), 159–195 (2001)
23. Heidrich-Meisner, V., Igel, C.: Neuroevolution strategies for episodic reinforcement learning. J. Algorithms **64**(4), 152–168 (2009)
24. Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In: Congress on Evolutionary Computation, vol. 4, pp. 2588–2595 (2003)
25. Jägersküpper, J.: How the (1+1)-ES using isotropic mutations minimizes positive definite quadratic forms. Theor. Comput. Sci. **361**(1), 38–56 (2006)
26. Jebalia, M., Auger, A.: On multiplicative noise models for stochastic search. In: Rudolph, G., Jansen, T., Beume, N., Lucas, S., Poloni, C. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 52–61. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87700-4_6
27. Kawaguchi, K.: Deep learning without poor local minima. In: Advances in Neural Information Processing Systems, pp. 586–594 (2016)
28. Loshchilov, I.: A computationally efficient limited memory CMA-ES for large scale optimization. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 397–404. ACM (2014)
29. Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. J. Artif. Intell. Res. (JAIR) **11**, 241–276 (1999)

30. Rechenberg, I.: Evolutionsstrategie-Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (1973)
31. Ros, R., Hansen, N.: A simple modification in CMA-ES achieving linear time and space complexity. In: Rudolph, G., Jansen, T., Beume, N., Lucas, S., Poloni, C. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 296–305. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87700-4_30
32. Stanley, K., D'Ambrosio, D., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009)
33. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
34. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, vol. 1. MIT press, Cambridge (1998)
35. Teytaud, O., Gelly, S.: General lower bounds for evolutionary algorithms. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 21–31. Springer, Heidelberg (2006). https://doi.org/10.1007/11844297_3
36. https://www.researchgate.net/publication/220743287_Uncertainty_handling_CMA-ES_for_reinforcement_learning

Lamarckian Evolution of Convolutional Neural Networks

Jonas Prellberg(✉) and Oliver Kramer

University of Oldenburg, Oldenburg, Germany
{jonas.prellberg,oliver.kramer}@uni-oldenburg.de

Abstract. Convolutional neural networks belong to the most successful image classifiers, but the adaptation of their network architecture to a particular problem is computationally expensive. We show that an evolutionary algorithm saves training time during the network architecture optimization, if learned network weights are inherited over generations by Lamarckian evolution. Experiments on typical image datasets show similar or significantly better test accuracies and improved convergence speeds compared to two different baselines without weight inheritance. On CIFAR-10 and CIFAR-100 a 75 % improvement in data efficiency is observed.

Keywords: Evolutionary algorithms · Convolutional neural networks
Architecture optimization · Weight inheritance

1 Introduction

Over the last years, deep neural networks and especially convolutional neural networks (CNN) have become state-of-the-art in numerous application domains. Their performance is sensitive to the choice of hyperparameters, such as learning rate or network architecture, which makes hyperparameter optimization an important aspect of applying neural networks to new problems. However, such optimization is computationally expensive due to the lengthy training process that has to be repeated each time a new hyperparameter setting is tested. This downside applies to optimization by hand as well as to automated approaches, such as grid search, random search or evolutionary optimization.

Earlier neuroevolution works [1,5,15,16] optimize network architectures together with the network weights using an evolutionary algorithm (EA). In recent years though, EAs have mostly been applied to optimize network hyperparameters, while the training is performed with backpropagation since it offers increased efficiency for training the large and deep networks prevalent today. The usual procedure is as follows: A genotype encodes the network architecture and corresponding hyperparameters. In a genotype-phenotype process, a network is built from this description and initialized with random weights. Then, several epochs of backpropagation on a training set adjust the network weights.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 424–435, 2018.
https://doi.org/10.1007/978-3-319-99259-4_34

Finally, the network is tested on a validation set and a metric, such as accuracy, is reported as the genotype's fitness.

Instead of randomly initializing the network weights before training, it is also possible to inherit the already learned weights of an ancestor network. This inheritance of acquired traits is a form of lamarckian evolution which, while rejected in biology, can prove useful in artificial evolution. In this work, we show that weight inheritance can drastically increase the data efficiency of an EA that optimizes neural network architectures.

The remainder of this paper is organized as follows: Sect. 2 presents related work about approaches that optimize architectures with EAs or use weight inheritance in their EAs. Section 3 describes the EA that is used in this paper and explains how the mutation with weight inheritance works. In Sect. 4 experimental results are presented and discussed. The paper ends with a conclusion in Sect. 5.

2 Related Work

Lamarckian evolution describes the idea that traits acquired over the lifetime of an individual are inherited to its offspring [14]. While rejected in biology, this approach can be beneficial for artificial evolution when there is a bi-directional mapping between genotype and phenotype. This allows to encode learned behavior back into the genotype and then apply an EA as usual. For example, Parker and Bryant [12] and Ku et al. [11] apply lamarckian evolution to neural networks by directly encoding the network weights in the genotype. This creates a simple one-to-one mapping between genotype and phenotype.

NEAT [16] is a neuroevolution algorithm that allows to grow neural networks starting with a minimal network and expanding it through mutation and principled crossover between the graphs. Because NEAT operates on single graph nodes and edges, it has been most successful on problems that can be solved with small neural networks. However, NEAT has inspired many approaches that try to extend the concept to evolving graphs of higher-level operations, such as convolutions.

Desell [3] uses a variant of NEAT to optimize the structure of a CNN that trains individual networks using backpropagation. An experiment with weight inheritance was conducted but it was not found to decrease the time necessary to train a single network to completion. Fernando et al. [4] use a microbial genetic algorithm to optimize the structure of a DPPN, which is a network that produces weights for a new network. Weight inheritance was found to improve the MSE in an image reconstruction experiment. Verbancsics and Harguess [18] test HyperNEAT [15] as a way to train CNNs for image classification. However, results were mediocre and could be substantially improved using backpropagation.

Kramer [10] uses a $(1+1)-$EA to optimize the hyperparameters defining a convolutional highway network. Suganuma et al. [17] use a modified $(1+\lambda)-$EA to optimize the structure of a CNN using a Cartesian genetic programming encoding scheme. Both approaches use small populations and only employ mutations while still achieving good results.

Weight inheritance is already employed in other works to varying degrees and for various reasons. For example, Jaderberg et al. [8] use an EA to optimize hyperparameters of static networks. It only performs mutations but inherits weights to mutated offspring. The networks are therefore trained to completion over multiple steps with potentially different hyperparameters. If one of the optimized hyperparameters is the learning rate, this effectively trains the network using a dynamic, evolved learning rate schedule. Instead, our goal is to highlight the data efficiency gains that come with weight inheritance.

Real et al. [13] use an EA with a very large population size and an unprecedented amount of computational resources to optimize the structure of a CNN. The method uses only mutation and inherits trained network weights through mutation. Training with backpropagation is performed for about 28 epochs per fitness evaluation on CIFAR-10 and CIFAR-100 and competitive results are reached. Furthermore, weight inheritance is shown to improve the test accuracy that the final network achieves. While their approach is similar to our work, we strive to keep computational demands low with the goal to reduce requirements further with the inclusion of weight inheritance.

Unrelated to evolutionary methods, the Net2Net algorithm [2] is an interesting approach to accelerate the sequential training of multiple related models. Starting from a trained model, it is possible to increase its depth or width while keeping the represented function the same. This is achieved by choosing the new weights in such a way that their effects cancel out. The training of the new model then progresses faster because the initial weights are already very good.

3 Method

To assess the influence of weight inheritance for neural network architecture optimization, design decisions regarding the optimizable hyperparameters and type of EA must be made. The goal is not to achieve state-of-the-art performance or find novel architectures but instead to show the advantages of weight inheritance. Therefore, we choose to optimize a fairly restricted architecture space which, however, is still applicable to many problems. This allows the EA to converge fast enough within our hardware resource constraints to make multiple repetitions of the same experiment feasible for statistical purposes.

The architecture search space is defined by the template presented in Fig. 1. It is made from stacked building blocks that consist of a convolutional layer followed by batch normalization [7] and a ReLU activation. The number of building blocks and the individual number of filters, kernel size and stride of the convolutional layer in each building block are subject to optimization. We statically append global average-pooling, a dense layer and a softmax activation function after the last building block, since experiments are performed specifically on image datasets.

The optimization is performed by a $(1+1)-\text{EA}$. In contrast to evolutionary algorithms with larger populations, the necessary computational resources are modest, but the method is also more prone to getting stuck in local optima in

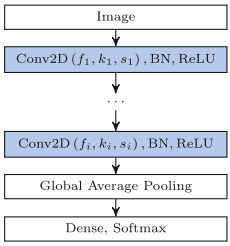


Fig. 1. Graph template defining the network architecture search space. Each building block (shown in blue) has individual hyperparameters filter count f_i, kernel size k_i and stride s_i that have to be optimized by the EA. (Color figure online)

multi-modal problems. To alleviate this, a form of niching is introduced. The evolutionary algorithm and its mutation operator are presented in more detail in the following sections.

3.1 Evolutionary Algorithm

Algorithm 1 presents the $(1+1)-$EA with niching as pseudo-code. An initial network consisting of a single convolutional layer with random filter count, random kernel size and a stride of one is created. This parent network is optimized by the EA as follows: First, a random mutation from the set of possible mutations is applied to the parent to create a child network. Next, the child network's fitness is evaluated. This means the network is trained for e epochs and the validation set accuracy is returned as its fitness. If the child's fitness is greater than the parent's fitness, the child replaces the parent.

Because this algorithm is greedy, it can get stuck in local minima. Therefore, a niching approach adapted from Kramer [10] is implemented. There is a random chance η to follow solutions that are initially worse. In such a case, a child, which has a lower fitness than its parent, is used as the parent network for a recursive call of the same algorithm. During niching, the mutate-evaluate-select-loop is repeated k times. When the last loop iteration ends, the best network found during niching is returned. If this network has a greater fitness than the original parent, it is selected. Otherwise, optimization proceeds with the original parent.

3.2 Mutation Operator

As mentioned before, the number of building blocks and the number of filters, kernel size and stride of each convolutional layer are subject to optimization. For simplicity, all these hyperparameters are chosen from predefined sets:

- Number of building blocks in $\mathbb{N}$
- Filter counts in $\mathcal{F} = \{16, 32, 64, 96, 128, 192, 256\}$
- Kernel size in $\mathcal{K} = \{1, 3, 5\}$
- Stride in $\mathcal{S} = \{1, 2\}$

```
a ← initial network
while termination condition not met do
    b ← mutate (a)
    if fitness (b) > fitness (a) then
        a ← b
    else if random() < η and not yet niching then
        c ← best network after recursion with b as initial network for k
        iterations (niching)
        if fitness (c) > fitness (a) then
            a ← c
        end
    end
end
return a
```

Algorithm 1. $(1 + 1) - \text{EA}$ with niching

Mutations are picked randomly from the list below. Each choice has a relative frequency (indicated by the multiplier in front of the list item) that determines how much more likely it is to be chosen than the mutation with a relative frequency of one. The frequencies have been chosen such that the more granular mutations, which are likely to have a smaller impact on the result, are applied less often in order to effectively use the available computation time.

- $3 \times$ *add block*: Adds a building block at a random position. The contained convolutional layer is initialized with a random filter count, random kernel size and a stride of one.
- $3 \times$ *remove block*: Removes a random building block.
- $2 \times$ *add filters*: Picks a random convolution and sets its filter count to the next greater value in $\mathcal{F}$.
- $2 \times$ *remove filters*: Picks a random convolution and sets its filter count to the next lower value in $\mathcal{F}$.
- $2 \times$ *change kernel size*: Picks a random convolution and randomly draws its kernel size from $\mathcal{K}$.
- $1 \times$ *change stride*: Picks a random convolution and randomly draws its stride from $\mathcal{S}$.

All random choices within each mutation, such as picking a random convolution or kernel size, are drawn uniformly at random from the appropriate set.

The mutation operator is forced to modify the network. A history of all previously evaluated networks is maintained and mutations are repeatedly applied to the parent network until a network is created that has not been evaluated before. Furthermore, only networks with at most three convolutions of stride two are allowed because the image inputs of CIFAR are only 32×32 and each stride-two convolution halves the side lengths.

3.3 Weight Inheritance

Each network is associated with a set of weights that contains, for example, kernels and biases for convolutional layers. When creating the initial parent network, these weights are randomly initialized in an appropriate fashion, e.g. Glorot [6] initialization. However, once a network has been evaluated its weights contain useful, learned values. When the mutation operator is applied, most of these weights are kept intact. The mutations *add block*, *remove block* and *change stride* do not influence existing weights so that all of them can be reused. The additional weights that belong to the convolutional and batch-normalization layers created by *add block* are randomly initialized. However, the mutations *add filters*, *remove filters* and *change kernel size* influence the shapes of some existing weights. For example, since the shape of a convolutional layer's kernel depends on its input and output shape, adding filters to a layer also affects the successive layer. In such cases, the affected weights are randomly reinitialized, while all other weights are reused.

4 Experiments

Training neural networks for image classification typically takes lots of resources. Hence, improving data-efficiency would be of great value. Therefore, we choose to experiment on the standard image benchmarks CIFAR-10 and CIFAR-100.

4.1 Setup

The mutation operator that employs weight inheritance is compared to a mutation operator that randomly reinitializes all network weights after each mutation. Otherwise, the same EA with the same hyperparameters is used on both datasets. The niching rate and depth are set to $\eta = 0.1$ and $k = 5$ respectively.

During each fitness evaluation, a network is trained for e epochs and subsequently its performance is assessed on the validation set. Choosing e is a trade-off between evaluation speed and the accuracy of the fitness assessment. If e is very low, evaluation is fast but networks are not trained to completion. Therefore the reported fitness will usually be lower than what the network could actually achieve given enough training time. Consequently, large but accurate networks have difficulty competing with smaller networks which train faster but might reach a lower final accuracy. If e is very high, these problems vanish but the evaluation takes a long time. Since many evaluations are necessary for large search spaces, this is impractical. Weight inheritance is supposed to offset some of the problems that come with small choices of e. The EA gets a budget of n total training epochs as its termination condition. This allows for a comparison of accuracy in terms of training examples that each experiment has processed. Also, the choices of n and e together influence how many generations, i.e. mutations, are possible within the total training epoch budget.

We propose an EA with weight inheritance and $e = 4$ training epochs per fitness evaluation. Note that 4 epochs is not sufficient to train networks that

work well on CIFAR to completion. The comparison baseline is an EA that does not use weight inheritance with two different epoch settings. The first baseline, which will be called baseline I, also uses 4 training epochs per evaluation in order to allow for a direct comparison. This allows us to show that the algorithm with weight inheritance is more data efficient and has better final accuracy all else being equal. The second baseline, which will be referred to as baseline II, uses $e = 16$ training epochs per evaluation. This significantly longer training time is more in line with the traditional approach of optimizing neural network architectures. It allows us to show that our observations still hold here and we do not simply trade a lower final accuracy for data efficiency. During evolution the best network is saved in regular intervals. After the EA finishes, these checkpoints are trained to completion and evaluated on the test set. We use this to compare test accuracies at one point during the evolution and after the evolution is finished.

The experiments are repeated 20 times with different random seeds to account for variance introduced by the randomness that is inherent to the EA and also the network training. Using the results from these repetitions, we perform significance tests using the one-sided Mann-Whitney U test. It was chosen because the sample sizes are small as each sample requires considerable time to create.

4.2 Training Details

The datasets have been split into a training, validation, and test set which contain 45k, 5k and 10k examples respectively for both CIFAR-10 and CIFAR-100. During a fitness evaluation, backpropagation is performed on the training set for e epochs and the validation set accuracy is reported as the network's fitness. All training phases are performed using a cross-entropy loss, the Adam [9] optimizer, a batch size of 512 and a learning rate of 0.001. Adam's state, i.e. first and second moment estimates, is not inherited during mutation. The test set is only used after all experiments have finished to evaluate saved network checkpoints. These checkpoints are trained to completion using a learning rate schedule: 10^{-3} until epoch 10, 10^{-4} until epoch 20 and 10^{-5} until epoch 30.

4.3 Results

Figure 2 compares three experimental settings on CIFAR-10 and CIFAR-100 with a total epoch budget of 512. The EA with weight inheritance outperforms the comparison baselines that do not use weight inheritance on both datasets. The accuracy plateau is reached more quickly and higher test accuracy is achieved.

For CIFAR-10, weight inheritance experiments reach a mean test accuracy of $85\,\% \pm 2\%$ after only 128 total training epochs. In comparison, baseline I experiments reach a mean test accuracy of $82\,\% \pm 1\%$ after 512 epochs. This means that the EA with weight inheritance achieves significantly ($p < 0.01$) higher accuracy than baseline I in 75 % less total training epochs. Baseline II experiments reach a test accuracy of $85\,\% \pm 3\%$ after 512 epochs. This is slightly, though not significantly, lower than the test accuracy of the weight inheritance experiments. After

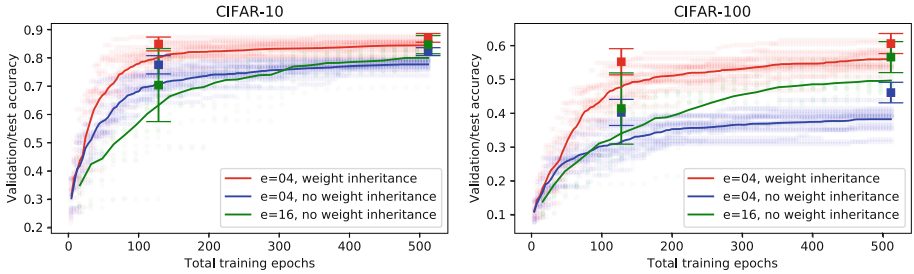


Fig. 2. Comparison of the EA with weight inheritance and $e = 4$ epochs against two baselines without weight inheritance and $e \in \{4, 16\}$ epochs on CIFAR-10 and CIFAR-100. Each dot represents the best validation accuracy achieved so far during an EA run at the respective epoch. Each line runs through the mean of the dots that are from the same experimental setting after the same amount of total epochs. Each box shows the average test accuracy after training the networks to convergence and the boxplot whiskers represent one standard deviation. (Color figure online)

512 epochs, the weight inheritance experiments reach a mean test accuracy of $87\,\% \pm 2\,\%$ which is significantly ($p < 0.01$) higher than baseline II at 512 epochs.

For CIFAR-100 results look very similar. After 128 epochs, the weight inheritance experiments achieve a mean test accuracy of $55\,\% \pm 4\,\%$. In contrast, baseline I experiments reach a significantly ($p < 0.01$) lower mean test accuracy of $46\,\% \pm 3\,\%$ after 512 epochs. Again, this is an improvement using 75 % less total training epochs. Baseline II experiments achieve a (not significantly) higher mean test accuracy of $57\,\% \pm 5\,\%$ after 512 epochs. Running the weight inheritance experiments for all 512 epochs as well results in a mean test accuracy of $61\,\% \pm 3\,\%$ which now is significantly ($p < 0.01$) higher than baseline II at 512 epochs.

In summary, weight inheritance experiments on CIFAR-10 and CIFAR-100 have shown to achieve significantly ($p < 0.01$) higher accuracy using a quarter of the total training epochs when compared to baseline I that uses the same amount of training epochs per fitness evaluation. Furthermore, final accuracy after 512 epochs is also significantly ($p < 0.01$) higher compared to baseline II experiments which benefited from more training epochs per fitness evaluation.

To get an idea how the evolutionary process modifies the genotypes, consider Fig. 3. It shows how the genome length, i.e. the number of building blocks in the corresponding networks, changes over the course of evolution. All EA runs are initialized with a genotype that contains a single building block. During the evolutionary process, increasingly larger genotypes are evaluated as they offer more accuracy than genotypes with fewer building blocks. The weight inheritance experiments and baseline II both settle around an average of 7 building blocks, whereas baseline I networks contain an average of 6 building blocks.

Table 1 lists minimum, mean and maximum test accuracies of the CIFAR experiments for specific checkpoint epochs. When weight inheritance is used, minimum, mean and maximum accuracies are higher than their baseline

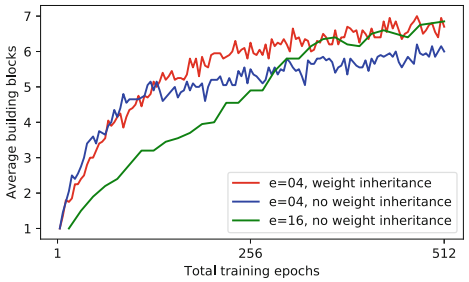


Fig. 3. Average (of all EA runs) number of building blocks in the genome during the optimization process on CIFAR-100

counterparts at all tested checkpoints. None of the results reach state-of-the-art performance, which was, as already pointed out, not the goal of this work. Our best evolved network on CIFAR-100 without data augmentation reaches a test accuracy of 66.1 % after 512 total epochs and required 6×10^{10} FLOPS[1] to find. This takes about 1.5 days on a single Nvidia K40 GPU.

Table 1. Test accuracies at the two checkpoints on CIFAR-10 and CIFAR-100

Settings			128 total epochs			512 total epochs		
Data	Inheritance	e	Min	Mean	Max	Min	Mean	Max
C10	Yes	4	**79.1**	**85.0±2.4**	**89.0**	**83.3**	**87.2±1.5**	**89.3**
	No	4	68.3	77.6±3.2	81.8	78.9	82.3±1.4	84.2
	No	16	32.5	70.4±12.6	85.5	76.6	84.8±3.1	88.9
C100	Yes	4	**47.7**	**55.3±3.8**	**61.1**	**56.1**	**60.7±2.9**	**66.1**
	No	4	31.7	40.2±3.8	46.0	39.8	46.1±3.0	52.2
	No	16	25.9	41.4±10.3	57.7	46.6	56.7±4.5	63.0

Additional experiments with 10 repetitions each have been performed on the MNIST and Fashion-MNIST datasets. The results are shown in Fig. 4. On both datasets, improvements from weight inheritance over its baselines are marginal. This is expected, as both datasets are easy to solve compared to CIFAR and can be learned quickly by small networks. Still, there is no deterioration in performance from using weight inheritance either.

[1] The FLOPS estimate for a single network is based on the FLOPS reported by the TensorFlow profiler to process a single example multiplied by 4 epochs, 98 batches per epoch and batch size 512. The total FLOPS of the EA run is the sum of the FLOPS estimates for all networks that were trained during the optimization.

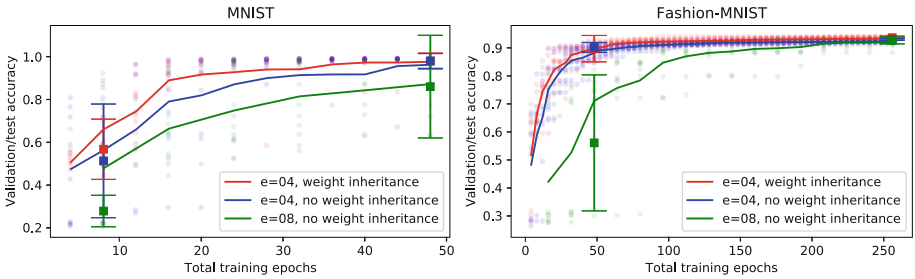


Fig. 4. Comparison of the EA with weight inheritance and $e = 4$ epochs against two baselines without weight inheritance and $e \in \{4, 8\}$ epochs on MNIST and Fashion-MNIST. See Fig. 2 for an explanation of the plot.

4.4 Discussion

The tradeoff between few and many training epochs per fitness evaluation that is explained in Sect. 4.1 has a visible effect in Fig. 2. At the beginning of each experiment, baseline I outperforms baseline II but at some point this relationship inverts. This is because small networks, which require only few epochs to reach good accuracy, are still sufficient to increase the validation set accuracy in the beginning of the experiment. However, at some point larger networks become necessary to further improve the results. These networks require more training time, making it easier for the algorithm that trains networks longer during fitness evaluation to progress. Therefore the green and blue graphs intersect. This happens earlier for CIFAR-100 because it is a harder problem than CIFAR-10.

We have seen weight inheritance experiments consistently outperform their baselines on CIFAR-10 and CIFAR-100 but could not observe a significant difference on the MNIST or Fashion-MNIST datasets. We did not find any instances of our experiments where weight inheritance was harmful, but this need not be the case generally: Just like in our work, most recent neuroevolution publications only use mutation operators and refrain from performing crossover. While this is usually motivated by the difficulty of designing a useful network crossover operator, crossover might also bring problems with regard to weight inheritance. Similarly to choosing a bad initialization before starting the training of a network, building a new network from trained parts of different networks could leave it in a region of the parameter space that is hard to optimize.

5 Conclusion

Evolutionary algorithms show promise as a way to automatically discover appropriate network architectures for new problems, but their usefulness is limited by their enormous computational requirements. Optimizing deep neural network architectures is computationally expensive because networks have to be retrained for each fitness evaluation. Therefore, approaches to lower these requirements are of great value.

We show that an evolutionary algorithm with a weight inheritance scheme generally achieves equal or higher accuracy compared to baselines that do not use weight inheritance and benefit from more training epochs per fitness evaluation. The fitness convergence speed is improved, sometimes making it possible to drastically reduce the number of total training epochs, while achieving test accuracies comparable to the baselines. Specifically, on both CIFAR-10 and CIFAR-100 weight inheritance increases data efficiency by 75 % with comparable test accuracy. The resulting speedup makes evolutionary algorithms a lot more viable for application to neural network architecture optimization even on hard problems. If accuracy is more important than training time, weight inheritance can also lead to a higher final test accuracy in some cases. Most importantly though, there has been no instance where weight inheritance was harmful. All results show either equally good or better results than the baselines. Thus it seems generally advisable to try the inclusion of weight inheritance schemes when mutation operators are used for neural network architecture optimization.

A promising research direction for future work will be to explore the interactions between weight inheritance and crossover operators. Also, further decreasing the time necessary for each training step in the evolutionary process is an important goal. For example, integrating the Net2Net [2] algorithm with the evolutionary algorithm might offer better results than randomly initializing additional new weights and allow for even less training steps.

References

1. Baluja, S.: Evolution of an artificial neural network based autonomous land vehicle controller. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **26**(3), 450–463 (1996)
2. Chen, T., Goodfellow, I., Shlens, J.: Net2net: accelerating learning via knowledge transfer. In: Proceedings of the International Conference on Learning Representations (ICLR 2016) (2016)
3. Desell, T.: Large scale evolution of convolutional neural networks using volunteer computing. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO 2017), pp. 127–128. ACM, New York (2017). https://doi.org/10.1145/3067695.3076002
4. Fernando, C., et al.: Convolution by evolution: differentiable pattern producing networks. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016), pp. 109–116. ACM, New York (2016). https://doi.org/10.1145/2908812.2908890
5. Garcia-Pedrajas, N., Hervas-Martinez, C., Munoz-Perez, J.: Covnet: a cooperative coevolutionary model for evolving artificial neural networks. IEEE Trans. Neural Netw. **14**(3), 575–596 (2003). https://doi.org/10.1109/TNN.2003.810618
6. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 9, pp. 249–256. PMLR, Chia Laguna Resort, Sardinia, 13–15 May 2010. http://proceedings.mlr.press/v9/glorot10a.html
7. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning (ICML 2015), Lille, France, pp. 448–456 (2015)

8. Jaderberg, M., et al.: Population Based Training of Neural Networks. ArXiv e-prints, November 2017
9. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: The International Conference on Learning Representations (ICLR 2015), December 2015
10. Kramer, O.: Evolution of convolutional highway networks. In: Sim, K., Kaufmann, P. (eds.) EvoApplications 2018. LNCS, vol. 10784, pp. 395–404. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77538-8_27
11. Ku, K.W.C., Mak, M.W., Siu, W.C.: A study of the Lamarckian evolution of recurrent neural networks. IEEE Trans. Evol. Comput. **4**(1), 31–42 (2000). https://doi.org/10.1109/4235.843493
12. Parker, M., Bryant, B.D.: Lamarckian neuroevolution for visual control in the quake ii environment. In: 2009 IEEE Congress on Evolutionary Computation, pp. 2630–2637, May 2009. https://doi.org/10.1109/CEC.2009.4983272
13. Real, E., et al.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning (ICML 2017) (2017). https://arxiv.org/abs/1703.01041
14. Sasaki, T., Tokoro, M.: Comparison between Lamarckian and Darwinian evolution on a model using neural networks and genetic algorithms. Knowl. Inf. Syst. **2**(2), 201–222 (2000). https://doi.org/10.1007/s101150050011
15. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009). https://doi.org/10.1162/artl.2009.15.2.15202
16. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). https://doi.org/10.1162/106365602320169811
17. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017), pp. 497–504. ACM, New York (2017). https://doi.org/10.1145/3071178.3071229
18. Verbancsics, P., Harguess, J.: Image classification using generative neuro evolution for deep learning. In: 2015 IEEE Winter Conference on Applications of Computer Vision, pp. 488–493, January 2015. https://doi.org/10.1109/WACV.2015.71

Learning Bayesian Networks with Algebraic Differential Evolution

Marco Baioletti[1(✉)], Alfredo Milani[1,3], and Valentino Santucci[2]

[1] Department of Mathematics and Computer Science,
University of Perugia, Perugia, Italy
{marco.baioletti,alfredo.milani}@unipg.it
[2] Department of Social and Human Sciences,
University for Foreigners of Perugia, Perugia, Italy
valentino.santucci@unistrapg.it
[3] Department of Computer Science, Hong Kong Baptist University,
Kowloon Tong, Hong Kong SAR, China

Abstract. In this paper we introduce DEBN, a novel evolutionary algorithm for learning the structure of a Bayesian Network. DEBN is an instantiation of the Algebraic Differential Evolution which is designed and applied to a particular (product) group whose elements encode all the Bayesian Networks of a given set of random variables. DEBN has been experimentally investigated on a set of standard benchmarks and its effectiveness is compared with BFO-B, a recent and effective bacterial foraging algorithm for Bayesian Network learning. The experimental results show that DEBN largely outperforms BFO-B, thus validating our algebraic approach as a viable solution for learning Bayesian Networks.

Keywords: Bayesian Networks Learning
Algebraic Differential Evolution

1 Introduction and Related Work

A Bayesian Network (BN) [14] is used to represent in a compact and effective way a probability distribution of a set of discrete random variables $X_1, \ldots, X_n$. A BN is composed by two different parts. The qualitative component is a directed acyclic graph (DAG) G in which the nodes are the variables X_i and the edges denote influences among variables. Given a variable X_i, $pa(X_i)$ is the set of parents of X_i and it contains all the variables X_j connected with an incoming edge to X_i. For each variable X_i, the quantitative component contains a conditional probability distribution of X_i with respect to $pa(X_i)$, i.e., the conditional probability $p(X_i = x_i | pa(X_i) = c_j)$ for each value x_i of X_i and for each combination c_j of values for the variables in $pa(X_i)$.

The problem of learning BNs from empirical data has been extensively studied during last years [14]. In particular, the problem of learning the structure

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 436–448, 2018.
https://doi.org/10.1007/978-3-319-99259-4_35

(qualitative part) of the network is well investigated, being the problem of learning the quantitative part much simpler, once the structure is given.

There are three main methodologies to learn the structure of a Bayesian Network. The first approach is to find conditional independence relations through statistical tests and to use them to infer the structure (for instance the presence of arcs), as done in [24]. Another possibility is the constraint-based approach, for instance dynamic programming [14]. Finally, a third approach is to perform a search process into a suitable space in order to find the optimal structure according to a given score metric.

Many score functions have been proposed for this purpose, e.g., K2, BDe, AIC, BIC and MDL scores [14]. In particular, we focus on the K2 and BDe scores that, given a BN with DAG structure G and a dataset D, are respectively defined as follows:

$$K2(G;D) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \left(\frac{(r_i-1)!}{(N_{ij}+r_i-1)!} \cdot \prod_{k=1}^{r_i} N_{ijk}!\right.$$

$$BDe(G;D) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij}+N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk}+N_{ijk})}{\Gamma(N'_{ijk})}$$

where, for each variable X_i, r_i is the cardinality of the domain of X_i, q_i is the number of possible value combinations of $pa(X_i)$, and N_{ijk} is the number of records in D in which X_i takes the k–th value and $pa(X_i)$ take the j–th combination of values. Besides, the BDe parameters N'_{ijk}, for every triple i, j, k, are usually set to $\frac{N'}{q_i r_i}$, where N' is a constant called *equivalent sample size* which, in this paper, as in several other works [14], we set to $N' = 1$. Furthermore, for the sake of computation, the logarithm of the score functions is usually employed.

There have been many attempts to solve the BN learning problem as a combinatorial problem and one of the most studied approach is through evolutionary techniques. Starting from Larrañaga paper [18], which used genetic algorithms, the most used approach is Ant Colony Optimization, since it uses an incremental way of building solutions, making the enforcement of acyclicity constraint easy to manage [10]. Another approach is to employ evolutionary algorithms to produce good orderings among variables, which are then used as input to other DAG construction algorithms, like K2 [11,13,26]. Other evolutionary approaches are based on discrete variants of Particle Swarm Optimization [15,27]. An alternative search space is the Partial DAG space [7], which represent in a compact way an equivalence class of DAGs.

A hybrid approach which combines conditional independence learning with searching for an optimal structure is [25].

One of the best evolutionary approach to this problem is BFO-B [12,28], an application of Bacterial Foraging Optimization technique. BFO-B has been compared to other swarm intelligence and other techniques showing that BFO-B outperforms all its evolutionary and non-evolutionary competitors.

In this paper we present an Algebraic Differential Evolution algorithm [21,22] to solve this problem. Differential Evolution (DE) [19] is widely adopted in opti-

mization problems due to its capacity of self-adapting the search to the fitness landscape at hand. Although DE has been originally proposed for continuous problems, in a previous series of papers [1,2,4,5,20,21], we have introduced an algebraic framework that allows to apply DE to combinatorial problems in which the search space is a finitely generated group.

In particular, in this paper we propose a novel representation for DAGs which allows to see the search space of all DAGs of a fixed vertex sets as a product group. In this way, it is possible to apply Algebraic DE to the BN learning problem in terms of finding the DAG with the maximum score.

Our algorithm, called DEBN, has been tested on some standard benchmarks and compared with BFO-B which, to the best of our knowledge, is the state-of-the-art evolutionary technique for BN learning. The experimental results show that DEBN largely outperforms BFO-B.

2 Differential Evolution

Differential Evolution (DE) [19] is a simple and powerful evolutionary algorithm for optimizing non-linear and even non-differentiable real functions $f : \mathbb{R}^n \to \mathbb{R}$. Hence, DE evolves a population of N real-valued vectors $x_1, \ldots, x_N \in \mathbb{R}^n$ by iteratively applying the three genetic operators: differential mutation, crossover, and selection.

The differential mutation generates a *mutant* y_i for each *target* population individual x_i. Though several mutation schemes have been proposed [19], the original one is denoted by rand/1 and it is computed as

$$y_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3}) \tag{1}$$

where r_1, r_2, r_3 are three random integers in $\{1, \ldots, N\}$ mutually different among them and with respect to i, while $F > 0$ is the scale factor parameter of DE.

For each pair formed by the target individual x_i and the mutant y_i, the crossover generates a *trial* solution z_i by recombining x_i and y_i. The most common variant is the *binomial crossover* [19] which generates z_i according to

$$z_i^{(j)} = \begin{cases} y_i^{(j)} \text{ if } r_{1,j} \leq CR \text{ or } r_2 = j \\ x_i^{(j)} \text{ otherwise} \end{cases} \tag{2}$$

where: CR is the crossover probability (another parameter of DE), $r_{1,j} \in [0, 1]$ is a random number generated for each dimension j, and $r_2 \in \{1, \ldots, n\}$ is randomly generated in order to guarantee that at least one component is inherited from the mutant y_i. Note also, that other crossover schemes are available [8,9,19].

Finally, the most used *selection* operator compares each target individual x_i with the corresponding trials z_i and selects the better between them to enter in the population of the next generation.

3 Algebraic Framework

Here we provide a concise description of the algebraic framework for evolutionary computation previously proposed in [3,21]. In particular, our attention has been mainly focused on ADE, an algebraic version of Differential Evolution, which obtained state-of-the-art performances on the permutation flow-shop scheduling problem [21,22]. Note anyway that the framework is rather general and can be adapted to other evolutionary algorithms [1] and other search spaces.

In principle, our algebraic methodology can be applied to all the combinatorial problems whose search space X forms a finitely generated group with respect to an internal composition $\star$ and a set of generators $H \subseteq X$ [16].

Recall that a group $(X, \star)$ is *finitely generated* if there exists a finite subset $H \subseteq X$, called *generating set*, such that any $x \in X$ can be decomposed as $x = h_1 \star h_2 \star \cdots \star h_l$ for some $h_1, h_2, \ldots, h_l \in H$. We also denote by $|x|$ the length of a minimal decomposition of x in terms of H.

The *Cayley graph* of a finitely generated group is the labeled digraph whose vertexes are the solutions in X and there exists an arc from x to y labeled by $h \in H$ if and only if $y = x \star h$. Moreover, for all $x \in X$, every (shortest) path from the neutral element e to x corresponds to a (minimal) decomposition of x, i.e., if the arc labels in the path are $(h_1, h_2, \ldots, h_l)$, then $x = h_1 \star h_2 \star \cdots \star h_l$.

The Cayley graph has an important geometric interpretation. Indeed, any solution $x \in X$ can be seen both as a *point*, i.e., a vertex in the graph, but also as a *vector* because its decomposition is a sequence of generators, i.e., arcs of a path in the Cayley graph. This dichotomous interpretation allows to define the operations $\oplus, \ominus, \odot$ on X in such a way that they simulate the analogous operations of the Euclidean space.

3.1 Vector-Like Operations

The *addition* $x \oplus y$ is defined as the application of the vector $y \in X$, decomposed as $(h_1, h_2, \ldots, h_l)$, to the point $x \in X$. It can be proved [21] that

$$x \oplus y = x \star y. \tag{3}$$

Given $x, y \in X$ considered as points, their *difference* $y \ominus x$ is the vector $(h_1, h_2, \ldots, h_l)$ which are the labels of a path from x to y. In [21] we proved that

$$y \ominus x = x^{-1} \star y. \tag{4}$$

Given $a \in [0, 1]$ and $x \in X$, the result of the *scalar multiplication of* x *by the scalar* a, denoted by $a \odot x$, is defined as

$$a \odot x = h_1 \star h_2 \star \cdots \star h_k \tag{5}$$

where $(h_1, h_2, \ldots, h_l)$ is a minimal decomposition of x and $k = \lceil a \cdot |x| \rceil$. The operation $\odot$, contrarily to $\oplus$ and $\ominus$, depends on the particular minimal decomposition chosen for x. In general there can be multiple minimal decompositions,

thus $\odot$ is not uniquely defined. However, since we are designing an evolutionary algorithm, we consider a random minimal decomposition of x when computing $a \odot x$.

In the following we describe the groups of the permutations and bit-strings, which will be used later in the paper.

3.2 Permutation Group

The set $\mathcal{S}_n$ of the permutations of $\{1, 2, \ldots, n\}$ forms a group, called *symmetric group*, with respect to the permutation composition $\circ$. Given $\pi, \rho \in \mathcal{S}_n$, their composition $\pi \circ \rho$ is defined as the permutation $(\pi \circ \rho)(j) = \pi(\rho(j))$ for all the indexes $j = 1, \ldots, n$. $\mathcal{S}_n$ is not Abelian (for $n \geq 3$) and its neutral element is the identity permutation ι such that $\iota(j) = j$ for all $j = 1, \ldots, n$.

Among the many generating sets of $\mathcal{S}_n$, the simplest one is the set of simple transpositions

$$ST = \{\sigma_i \in \mathcal{S}_n : 1 \leq i < n\},$$

where σ_i corresponds to an adjacent swap between positions i and $i+1$. Formally: $\sigma_i(i) = i + 1$, $\sigma_i(i + 1) = i$, and $\sigma_i(j) = j$ for $j \in \{1, \ldots, n\} \setminus \{i, i + 1\}$.

A random decomposition algorithm for this generating set is the RandBS procedure, introduced in [21], which produces a random minimal decomposition of a given permutation by requiring $O(n^2)$ computational time. It is worth to notice that the length of a minimal decomposition of $\pi \in \mathcal{S}_n$ is the number of inversions of π.

We will denote by $\oplus_p, \ominus_p, \odot_p$, respectively, the operations $\oplus, \ominus, \odot$ defined for $\mathcal{S}_n$.

3.3 Bit-String Group

The set $\mathbb{B}^m$ of all the m-length bit-strings forms an Abelian group with respect to the bitwise XOR operator $\veebar$. Its neutral element is the all-zeros string 0. Since $x \veebar x = 0$ for all $x \in \mathbb{B}^m$, the inverse of any $x \in \mathbb{B}^m$ is itself.

The most obvious generating set for $\mathbb{B}^m$ is the set

$$U = \{u_i \in \mathbb{B}^m : u_i(i) = 1 \text{ and } u_i(j) = 0 \text{ for } j \neq i\},$$

where $u_i(k)$ indicates the k-th bit of the string u_i.

A random decomposition algorithm for a bit-string b can be easily found by selecting all the indices $i \in \{1, \ldots, m\}$ with $b(i) = 1$ and disposing them into a sequence with a random order. Note that the length of a minimal decomposition of b is just its Hamming weight.

We will denote by $\oplus_b, \ominus_b, \odot_b$, respectively, the operations $\oplus, \ominus, \odot$ defined for $\mathbb{B}^m$. It is important to notice that $\oplus_b$ and $\ominus_b$ coincide and both are commutative.

4 Dual Representation of Bayesian Networks

In this section we introduce the representation of BN structures, i.e., DAGs, and their associated finitely generated group.

A DAG G of n vertices can be represented by a pair (π, b), where $\pi \in \mathcal{S}_n$ and $b \in \mathbb{B}^m$ with $m = \binom{n}{2}$.

The bits of b represent the skeleton of G. Let $C = \{(j,k) : 1 \leq j < k \leq n\}$ be the ordered set of vertex pairs, if the i–th pair of C is (j,k), then there exists in G an arc from X_j to X_k, or vice versa, if and only if $b_i = 1$.

The permutation π determines the direction of the arcs: if $b_i = 1$, then the arc goes from X_j to X_k if j appears before k in π, i.e., $\pi^{-1}(j) < \pi^{-1}(k)$, otherwise the arc goes in the opposite direction. Said in other words, π is a topological order of the variables $X_1, \ldots, X_n$.

One of the most important properties of this representation is that any pair (π, b) represents a DAG. This fact is an apparent advantage of this representation with respect to other forms (for instance the adjacency matrix) where constraint must be used to select which combinations correspond to directed acyclic graph. However, since a DAG can have more than one topological order, our representation is, in general, a many-to-one representation, i.e., there can be multiple pairs (π, b) that represent the same DAG.

The set of all the pairs (π, b), such that $\pi \in \mathcal{S}_n$ and $b \in \mathbb{B}^m$, is the Cartesian product $\mathcal{B} = \mathcal{S}_n \times \mathbb{B}^m$. Importantly, $\mathcal{B}$ can be endowed with the binary operation $*$ defined as

$$(\pi, b) * (\pi', b') = (\pi \circ \pi', b \veebar b') \tag{6}$$

where $\circ$ and $\veebar$ are the group operations for $\mathcal{S}_n$ and $\mathbb{B}^m$, respectively. Therefore $\mathcal{B}$ is a group with respect to $*$, namely the *product group* of $\mathcal{S}_n$ and $\mathbb{B}^m$. Its neutral element is $(\iota, 0)$, while the inverse of (π, b) is (π^{-1}, b).

Addition and subtraction on $\mathcal{B}$ can now be defined as in Eqs. (3) and (4), by using the operation $*$ and its related inverse operator.

In order to define the multiplication of a pair (π, b) by a scalar $a \in [0, 1]$, we have to choose a generating set for $\mathcal{B}$. We describe two ways of defining a generating set for $\mathcal{B}$ starting from the generating sets for $\mathcal{S}_n$ and for $\mathbb{B}^m$.

The *additive* generating set A is defined as

$$A = ST' \cup U'$$

where $ST' = \{(\sigma_i, 0) : i = 1, \ldots, n-1\}$ and $U' = \{(\iota, u_j) : j = 1, \ldots, m\}$. Its cardinality is $|A| = n - 1 + m$. Note that the generators of ST' only influence the permutation part (since the second component of the element of ST' is 0). Conversely, the generators of U' have effect only on the binary part. Using A as generating set, it is easy to prove that $|(\pi, b)| = |\pi| + |b|$.

A randomized decomposition algorithm for A which produces a random minimal decomposition of $(\pi, b) \in \mathcal{B}$ is the following. Given a random minimal decomposition $(\sigma_{h_1}, \ldots, \sigma_{h_L})$ of π and a random minimal decomposition $(u_{k_1}, \ldots, u_{k_M})$ of b, then

– create an empty sequence r of size $L + M$,
– choose L random different indices $1 \leq j_1 < \cdots < j_L \leq L + M$ of r,
– assign $r_{j_v} \leftarrow (\sigma_{h_v}, 0)$ for $v = 1, \ldots, L$,
– fill the M unassigned positions of r with the pairs (ι, u_{k_v}) for $v = 1, \ldots, M$.

The *multiplicative* generating set P is defined as

$$P = (ST \times U) \cup A$$

whose cardinality is $n(m + 1)$. A minimal decomposition of a pair $(\pi, b) \in \mathcal{B}$ in terms of P is much shorter than a minimal decomposition in terms of A, because each generator belonging to $ST \times U$ affects both the permutation and binary part.

A minimal decomposition of $(\pi, b) \in \mathcal{B}$ can be obtained by pairing the minimal decompositions of π and b. In the general case, a certain number of copies of the neutral element have to be added to the shorter of the two minimal decompositions in order to match the length of the longest one. The generators of A (also present in P) are useful for this purpose. Using P as generating set, it can be easily proved that $|(\pi, b)| = \max\{|\pi|, |b|\}$.

A randomized minimal decomposition for $(\pi, b) \in \mathcal{B}$ in terms of P is computed as follows. Given a random minimal decomposition $(\sigma_{h_1}, \ldots, \sigma_{h_L})$ of π and a random minimal decomposition $(u_{k_1}, \ldots, u_{k_M})$ of b, if $L < M$, then

– create an empty sequence r of size M,
– choose L random different indices $1 \leq j_1 < \cdots < j_L < M$ in r,
– assign $r_{j_v} \leftarrow (\sigma_{h_v}, u_{k_{j_v}})$ for $v = 1, \ldots, L$,
– fill the $M - L$ unassigned positions of r with the pairs (ι, u_{k_v}) for $v = 1, \ldots, M$.

The method works in a similar way when $L \geq M$.

5 The Algorithm DEBN

In this section we describe DEBN, the algorithm based on the Algebraic Differential Evolution for learning Bayesian Networks. It has the same structure of a classical DE algorithm: its pseudo-code is depicted in Algorithm 1.

Any population individual x_i is represented by means of the dual representation introduced in Sect. 4, i.e., $x_i = (\pi_i, b_i)$, where $\pi_i \in \mathcal{S}_n$, $b_i \in \mathbb{B}^m$, and $m = \binom{n}{2}$. The individuals are evaluated by means of a BN score function selected by the user. In this work, K2 and BDe have been considered (see Sect. 1).

Each individual $x_i = (\pi_i, b_i)$ is randomly initialized by selecting a permutation π_i uniformly at random on $\mathcal{S}_n$, while each bit of b_i is set to 1 with probability $\frac{2}{n-1}$, thus that the average number of edges in the BN represented by x_i is n.

The discrete differential mutation uses the algebraic operations $\oplus$, $\ominus$, $\odot$ of $\mathcal{B}$. Moreover, in order to mitigate the diversity loss phenomenon, typical in combinatorial spaces, the rand/1 scheme of classical DE has been extended by introducing a random term as follows:

$$y_i = (x_{r_1} \oplus t) \oplus F \odot (x_{r_2} \ominus x_{r_3}), \tag{7}$$

Algorithm 1. DEBN Pseudo-Code

```
1: function DEBN
2:     Initialize and Evaluate the Population
3:     while termination criterion is not met do
4:         for i ← 1 to N do
5:             y_i ← DifferentialMutation(x_i, F, pm)
6:             z_i ← Crossover(x_i, y_i, CR)
7:             Evaluate(z_i)
8:         for i ← 1 to N do
9:             x_i ← selection(x_i, z_i)
10:    return the best BN structure found
```

where, as in Eq. (1), x_{r_1}, x_{r_2}, and x_{r_3} are three random population individual different to each other and with respect to x_i, while $F \in [0, 1]$ is the scale factor parameter.

Furthermore, $t \in \mathcal{B}$ is randomly generated by means of the *pre-mutation probability* $pm \in (0, 1)$ such that $|t| = k$ with probability pm^k. Operatively, t is initialized to the neutral element $(\iota, 0)$, then, during a loop, a random number $r \in [0, 1]$ is generated and, if $r < pm$, a suitable randomly selected generator (from A or P) is applied to t. As soon as $r \geq pm$, the loop is stopped and t is returned.

Two crossover operators are separately applied to the permutation and binary parts of $x_i = (\pi_i, b_i)$ and $y_i = (\pi'_i, b'_i)$, thus obtaining the trial individual $z_i = (\pi''_i, b''_i)$. We have implemented different crossover schemes for the permutation and binary parts. After preliminary tests we decide to use this combination of crossover:

$$\pi'' = CYC(\pi_i, \pi'_i)$$

$$b''_i = BIN(b_i, b'_i, CR)$$

where CYC is the (parameterless) cycle crossover described in [17], and BIN is the usual binomial crossover of DE, as defined in Eq. (2).

The generation is then concluded by applying the 1-to-1 selection scheme of classical DE.

DEBN has also been equipped with a self-adaptive procedure, inspired by the well known jDE method [6], that allows to self-regulate the three parameters pm, F, and CR. Each population individual maintains its own parameter values. Then, independently for each parameter, when mutant and trial are generated, with probability 0.9, they inherit the value of the target population individual, otherwise they randomly sample a new value in the allowed range for the parameter at hand, i.e., $[0.1, 1]$ for F, $[0, 1]$ for CR, and $[0.1, 0.3]$ for pm.

Finally, two implementations of DEBN can be devised, namely DEBN_+, where the operation $\odot$ is defined with respect to the generating set A, and $\text{DEBN}_\times$, which is based on P.

6 Experimental Results

In this section we describe the experimental results obtained with the implementation of DEBN algorithm. Experiments have been conducted using 8 popular BN benchmarks. For each one, we have generated two datasets of different sizes by using the sampling procedure of the *bnlearn* R package [23].

The benchmark names are provided in Table 1, where we also report the dataset sizes, together with the number of nodes and edges of the true networks from which the datasets are generated. The aim of the experimentation is thus to try to the recover the original networks by maximizing the K2 and BDe scores, computed by only looking at the datasets.

Table 1. Datasets

Network	Size1	Size2	#vars	#edges
Alarm	4000	8000	37	46
Asia	1000	5000	8	8
Barley	5000	10000	48	84
Child	2000	5000	20	25
Hailfinder	5000	10000	56	66
Insurance	3000	6000	27	52
Water	4000	8000	32	66
Win95pts	5000	10000	76	112

Three algorithms have been compared: DEBN_+, $\text{DEBN}_\times$, and the recent state-of-the-art evolutionary algorithm BFO-B, which has been implemented by faithfully following the description given in [28]. As indicated by its authors, the BFO-B parameters have been set to $N_s = 4$, $N_{re} = 4$, $N_{ed} = 3$, $S = 80$, $N_c = 30$, and $P_{ed} = 0.1$.

Our DEBN_+ and $\text{DEBN}_\times$ only require to set the population size N. After some preliminary tests (here not reported for the lack of space), we decided to use $N = 50$ for both the variants.

In order to choose a fair termination criterion for DEBN, we have observed that BFO-B performed around 100 000 fitness evaluations in average, so we used this number of evaluations also for DEBN.

All the three algorithms have been run 20 times per dataset using both the considered score metrics. Tables 2 and 3 provide the average and best scores obtained by all the algorithm and considering, respectively, BDe and K2 as score functions. For each dataset, the best average and maximum scores are indicated in, respectively, bold and italic.

Tables 2 and 3 clearly show that our proposals largely outperform BFO-B on almost all the comparisons. The only exceptions are on *water_4000* and *water_8000* where BFO-B obtains a better average K2 score. However, in the

Table 2. Results with BDe score

Dataset	BFO-B		DEBN_{+}		$\text{DEBN}_{\times}$	
	Avg	Max	Avg	Max	Avg	Max
alarm_4000	−43794.98	−43437.00	**−42570.45**	*−42502.57*	−42610.05	−42519.19
alarm_8000	−87576.63	−86859.20	−85234.69	*−85086.80*	**−85209.07**	−85087.68
asia_1000	−2310.45	*−2310.37*	**−2310.37**	*−2310.37*	**−2310.37**	*−2310.37*
asia_5000	−11394.74	−11394.50	**−11394.49**	*−11394.49*	**−11394.49**	*−11394.49*
barley_5000	−268201.94	−264423.00	**−266737.42**	*−262211.07*	−267429.76	−263562.99
barley_10000	−532904.94	−527937.00	−524224.13	*−512050.47*	**−523856.84**	−517022.55
child_2000	−25041.73	−25040.10	−25040.90	*−25040.08*	**−25040.08**	*−25040.08*
child_5000	−61385.42	−61382.90	**−61382.93**	*−61382.93*	**−61382.93**	*−61382.93*
hailfinder_5000	−251569.15	−250809.00	−249938.90	*−249555.91*	**−249935.54**	−249635.38
hailfinder_10000	−503277.10	−499926.00	−497099.00	−496475.76	**−496891.03**	*−496451.81*
insurance_3000	−40884.81	−40718.00	−40472.34	*−40347.91*	**−40454.16**	−40389.54
insurance_6000	−80903.57	−80495.30	−80146.05	−79966.23	**−80109.22**	*−79960.52*
water_4000	−52234.43	−52154.70	**−52068.78**	*−51993.13*	−52071.86	−52001.85
water_8000	−103631.50	−103443.00	**−103320.64**	*−103155.90*	−103352.41	−103168.28
win95pts_5000	−60481.39	−58908.90	−51460.67	−49924.90	**−51348.43**	*−49440.49*
win95pts_10000	−121667.70	−118890.00	−105275.83	*−99497.04*	**−104804.92**	−100368.35

Table 3. Results with $K2$ score

Dataset	BFO-B		DEBN_{+}		$\text{DEBN}_{\times}$	
	Avg	Max	Avg	Max	Avg	Max
alarm_4000	−43802.95	−43429.90	**−42760.22**	*−42694.80*	−42790.03	−42695.92
alarm_8000	−87576.81	−86298.90	**−85451.64**	*−85301.86*	−85573.49	−85318.76
asia_1000	−2289.97	*−2289.94*	**−2289.94**	*−2289.94*	**−2289.94**	*−2289.94*
asia_5000	−11373.64	−11373.50	**−11373.47**	*−11373.47*	**−11373.47**	*−11373.47*
barley_5000	−282797.80	−280376.00	**−278009.54**	*−274887.52*	−279218.10	−275726.47
barley_10000	−558264.08	−551764.00	−538313.54	*−531544.39*	**−536969.49**	−531683.39
child_2000	−25022.86	−25019.60	**−25019.56**	*−25019.56*	−25020.45	*−25019.56*
child_5000	−61366.06	*−61363.50*	**−61363.76**	−61363.55	**−61363.76**	−61363.55
hailfinder_5000	−252976.70	−251587.00	**−250506.92**	*−250204.53*	−250543.04	−250266.94
hailfinder_10000	−506897.80	−504137.00	**−497874.99**	−497546.61	−497917.03	*−497194.28*
insurance_3000	−41273.06	−41150.30	**−40905.16**	−40857.28	−40921.03	*−40823.76*
insurance_6000	−81655.57	−81088.70	**−80868.01**	*−80642.36*	−80905.42	−80707.54
water_4000	**−52451.31**	*−52429.50*	−52487.60	−52433.88	−52490.21	−52449.57
water_8000	**−103793.35**	−103760.00	−103854.43	−103771.33	−103842.66	*−103755.88*
win95pts_5000	−54866.60	−52726.90	**−50442.92**	−48932.97	−50852.83	*−48509.02*
win95pts_10000	−112196.95	−110883.00	**−103897.32**	*−97930.23*	−104792.63	−101290.97

larger BNs *hailfinder* and *win95pts*, the score differences are remarkably large in favor of the DEBN algorithms.

Regarding the comparison between the two DEBNs and considering the BDe metric, $\text{DEBN}_\times$ obtained better average scores, while DEBN_+ shows better peak performances.

On the other end, considering the K2 metric, DEBN_+ is slightly preferable with respect to $\text{DEBN}_\times$ both in terms of average and peak performances.

7 Conclusions and Future Work

In this paper we have described DEBN, an Algebraic Differential Evolution algorithm [21] for learning the structure of a Bayesian Network (BN). DEBN is based on a novel BNs representation based on the algebraic concept of product group, where a DAG is represented by a permutation and a bit-string. Both permutations and bit-strings are finitely generated groups, hence it is possible to apply the principles of Algebraic Differential Evolution.

Two variants of DEBN have been proposed and experimentally compared with BFO-B, one of the best evolutionary algorithms for BN learning. The experimental results clearly show that DEBN largely outperforms BFO-B.

As future lines of research, we will investigate: the use of other generating sets for the permutation part (see [3]), and the application of the DEBN scheme to other problems whose solutions are DAGs.

References

1. Baioletti, M., Milani, A., Santucci, V.: Algebraic particle swarm optimization for the permutations search space. In: Proceedings of 2017 IEEE Congress on Evolutionary Computation (CEC 2017), pp. 1587–1594 (2017). https://doi.org/10.1109/CEC.2017.7969492
2. Baioletti, M., Milani, A., Santucci, V.: Linear ordering optimization with a combinatorial differential evolution. In: Proceedings of 2015 IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2015), pp. 2135–2140 (2015). https://doi.org/10.1109/SMC.2015.373
3. Baioletti, M., Milani, A., Santucci, V.: An extension of algebraic differential evolution for the linear ordering problem with cumulative costs. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 123–133. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_12
4. Baioletti, M., Milani, A., Santucci, V.: Automatic algebraic evolutionary algorithms. In: Pelillo, M., Poli, I., Roli, A., Serra, R., Slanzi, D., Villani, M. (eds.) WIVACE 2017. CCIS, vol. 830, pp. 271–283. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78658-2_20
5. Baioletti, M., Milani, A., Santucci, V.: MOEA/DEP: an algebraic decomposition-based evolutionary algorithm for the multiobjective permutation flowshop scheduling problem. In: Liefooghe, A., López-Ibáñez, M. (eds.) EvoCOP 2018. LNCS, vol. 10782, pp. 132–145. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77449-7_9

6. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans. Evol. Comput. **10**(6), 646–657 (2006)
7. Daly, R., Shen, Q.: Learning Bayesian networks equivalence with ant colony optimization. J. Artif. Intell. Res. **35**, 391–447 (2009)
8. Das, S., Suganthan, P.N.: Differential evolution: a survey of the state-of-the-art. IEEE Trans. Evol. Comput. **15**(1), 4–31 (2011)
9. Das, S., Mullick, S.S., Suganthan, P.N.: Recent advances in differential evolution an updated survey. Swarm Evol. Comput. **27**, 1–30 (2016)
10. de Campos, L.M., Fernández-Luna, J.M., Gámez, J.A., Puerta, J.M.: Ant colony optimization for learning Bayesian networks. Int. J. Approx. Reason. **31**(3), 291–311 (2002)
11. de Campos, L.M., Gámez, J.A., Puerta, J.M.: Learning Bayesian networks by ant colony optimization: searching in two different spaces. Mathw. Soft Comput. **9**(2–3), 251–268 (2002)
12. Ji, J., Yang, C., Liu, J., Liu, J., Yin, B.: A comparative study on swarm intelligence for structure learning of Bayesian networks. Soft Comput. **21**(22), 6713–6738 (2017)
13. Kabli, R., Herrmann, F., McCall, J.: A chain-model genetic algorithm for Bayesian network structure learning. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 1264–1271. ACM (2007)
14. Koller, D., Friedman, N.: Probabilistic Graphical Models Principles and Techniques. MIT Press, Cambridge (2009)
15. Kuo, S.-C., Wang, H.-J.., Wei, H.-Y., Chen, C.-C., Li, S.-T.: Applying MDL in PSO for learning Bayesian networks. In: 2011 IEEE International Conference on Fuzzy Systems (FUZZ), pp. 1587–1592. IEEE (2011)
16. Lang, S.: Algebra, vol. 211. Springer, Heidelberg (2002). https://doi.org/10.1007/978-1-4613-0041-0
17. Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: a review of representations and operators. Artif. Intell. Rev. **13**(2), 129–170 (1999)
18. Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R.H., Kuijpers, C.M.H.: Structure learning of Bayesian network by genetic algorithms: a performance analysis of control parameters. IEEE Trans. Pattern Anal. Mach. Intell. **18**(9), 912–926 (1996)
19. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-31306-0
20. Santucci, V., Baioletti, M., Milani, A.: An algebraic differential evolution for the linear ordering problem. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO 2015), pp. 1479–1480. ACM, New York (2015). https://doi.org/10.1145/2739482.2764693
21. Santucci, V., Baioletti, M., Milani, A.: Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. IEEE Trans. Evol. Comput. **20**(5), 682–694 (2016). https://doi.org/10.1109/TEVC.2015.2507785
22. Santucci, V., Baioletti, M., Milani, A.: Solving permutation flowshop scheduling problems with a discrete differential evolution algorithm. AI Commun. **29**(2), 269–286 (2016). https://doi.org/10.3233/AIC-150695
23. Scutari, M.: Learning Bayesian networks with the bnlearn R package. J. Stat. Softw. **35**(3), 1–22 (2010)

24. Tsamardinos, I., Brown, L.E., Aliferis, C.F.: The max-min hill-climbing Bayesian network structure learning algorithm. Mach. Learn. **65**(1), 31–78 (2006)
25. van Dijk, S., van der Gaag, L.C., Thierens, D.: A skeleton-based approach to learning Bayesian networks from data. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) PKDD 2003. LNCS (LNAI), vol. 2838, pp. 132–143. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39804-2_14
26. Wu, Y., McCall, J.A.W., Corne, D.W.: Two novel ant colony optimization approaches for Bayesian network structure learning. In: IEEE Congress on Evolutionary Computation, pp. 1–7 (2010)
27. Xing-Chen, H., Zheng, Q., Lei, T., Li-Ping, S.: Learning Bayesian networks structures with discrete particle swarm optimization algorithm. In: 2007 IEEE Symposium on Foundations of Computational Intelligence, pp. 47–52 (2007)
28. Yang, C., Ji, J., Liu, J., Liu, J., Yin, B.: Structural learning of Bayesian networks by bacterial foraging optimization. Int. J. Approx. Reason. **69**, 147–167 (2016)

Optimal Neuron Selection and Generalization: NK Ensemble Neural Networks

Darrell Whitley[1(✉)], Renato Tinós[2], and Francisco Chicano[3]

[1] Colorado State University, Fort Collins, CO 80523, USA
whitley@colostate.eu
[2] University of São Paulo, Ribeirão Preto, SP, Brazil
[3] University of Málaga, Málaga, Spain

Abstract. This paper explores how learning can be achieved by turning on and off neurons in a special hidden layer of a neural network. By posing the neuron selection problem as a pseudo-Boolean optimization problem with bounded tree width, an exact global optimum can be obtained to the neuron selection problem in $O(N)$ time. To illustrate the effectiveness of neuron selection, the method is applied to optimizing a modified Echo State Network for two learning problems: (1) Mackey-Glass time series prediction and (2) a reinforcement learning problem using a recurrent neural network. Empirical tests indicate that neuron selection results in rapid learning and, more importantly, improved generalization.

1 Introduction to Optimal Neural Selection

Programmed cell death, also known as neuronal apoptosis, is known to be an important part of normal brain development and mechanisms behind neuronal apoptosis have been extensively studied [12,15]. Along with synaptic pruning [3], neuronal apoptosis helps to shape the size and configuration of different neural processing centers in the brain. This is also thought to represent a very basic form of learning. Thus, it is natural to ask what are the benefits of neuron selection and how might neuron selection be utilized in artificial neural networks.

The proposed method converts a form of the neuron selection problem into a k-bounded pseudo Boolean optimization problem, with the goal of identifying useful combinations of neurons. A k-bounded pseudo-Boolean optimization problem [2] can be expressed in the following form:

$$f(\mathbf{x}) = \sum_{i=1}^{M} f_i(\mathbf{x}) \tag{1}$$

where $\mathbf{x} \in \{0,1\}^N$ is a bit vector, each subfunction f_i can output any real value, and $f_i(\mathbf{x})$ is evaluated using a subset of k bits drawn from the bit vector $\mathbf{x}$. Each subfunction f_i identifies which bits are the correct inputs to f_i. MAX-kSAT is a classic example of a k-bounded Boolean optimization problem, where

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 449–460, 2018.
https://doi.org/10.1007/978-3-319-99259-4_36

each subfunction f_i corresponds to a clause that evaluates to 0 or 1. Spin glass systems and NK Landscapes are also well known k-bounded pseudo Boolean optimization problems.

In this paper, we will restrict our attention to neural networks with a single output neuron that learn a single continuous real valued output; however, the learning method can generalize to multiple outputs. The bit vector $\mathbf{x} \in \{0,1\}^N$ will be used to indicate if a neuron should be turned on or turned off.

In order to create M subfunctions, the single output neuron is converted into an ensemble of M output neurons, all of which attempt to learn the same task. Furthermore, only a subset of other neurons in the neural network (k to be precise) will be allowed to connect to a particular output neurons. Thus, optimizing Eq. 1 results in the selection of a subset of neurons from vector $\mathbf{x}$ that contribute in a positive fashion to an ensemble of M outputs attempting to learn the same task. The neuron selection method proposed here only acts on the set of neurons that are directly connected to an output neuron.

Each subfunction f_i might minimize the mean squared error under supervising learning, or it might maximize a performance metric in the case of reinforcement learning. Because the problem is posed as a k-bounded pseudo-Boolean optimization problem, each of the output neurons (corresponding to a subfunction f_i) receives input from only k other neurons. If k neurons were randomly selected to connect to an output f_i, it would probably be necessary to use a heuristic method to optimize the neuron selection problem. However, there are advantages to choosing a localized and structured pattern when connecting neurons to outputs. In the current paper, neurons are connected to outputs in such a way that the neuron selection problem can be solved in $O(N)$ time using dynamic programming. The resulting solution is globally optimal relative to the starting architecture, and the input vector $\mathbf{x}$. Obviously, different initial architectures would nevertheless yield different results.

We apply this new learning method to two problems. The first problem is the Mackey-Glass time series prediction problem [10]. The second problem is the reinforcement learning problem of balancing two poles on a cart while providing only cart position and the two pole angles as input [20]; this configuration makes it necessary to learn to compute velocity information and makes this classic control problem much more difficult. For both learning problems, we utilize a recurrent neural network in the form of an Echo State Network.

Echo State Networks are one form of *reservoir computing* [13,14]. Reservoir computing networks use a reservoir of sparsely and recurrently connected artifical neurons that have randomly generated weighted connections. Both the input neurons and output neurons are outside of the reservoir. The weights inside of the reservoir of neurons are not adjusted by learning. To determine which neurons are useful, learning is typically used to adjust the weights that connect artificial neurons in the reservoir to the outputs. In our experiments, we use neuron selection to determine which neurons connected to the reservoir are useful.

For the Mackey-Glass problem, neuron selection improves generalization using similar computation time when compared to the standard Echo State

Network. For the two pole balancing problem with no velocity inputs, neuron selection learns more rapidly and produces dramatically better generalization than any other method that has been reported in the literature. It achieves these results with no policy iteration and no back propagation. Neuron selection is the *only* form of learning that is utilized.

Although we use an Echo State Network as the foundation for our experiments, in principle the same technique might be applied to multi-layered perceptrons or deep learning networks. It therefore provides a new means of automatically configuring a neural network architecture to fit a particular problem.

2 Optimization by Dynamic Programming

We will very briefly outline how and when dynamic programming can used to optimize k-bounded pseudo-Boolean functions. We construct a Variable Interaction Graph, G, to model the interaction between variables in a k-bound pseudo-Boolean optimization problem. If two variables x_q and x_j appear together in subfunction f_i then there is an edge between x_q and x_j in G. We next define the concepts of tree width and tree decomposition of a graph [6].

Definition 1. *[6] A tree decomposition of any graph $G(V, E)$ is a pair $D = (S, T)$ where $S = \{X_i, i \in I\}$ is a collection of I subsets of the vertices of G and T is a tree with one node for each subset in the collection S, such that:*

1. $\bigcup_{i \in I} X_i = V$,
2. *for all the edges $(u, w) \in E$ there exists a subset $X_i \in S$ such that both u and w are in X_i,*
3. *for each vertex v, the set of nodes that contain v, $\{i | v \in X_i\}$, form a subtree of T.*

The tree width, denoted by w, of the decomposition $D = (S, T)$ is given by $w = max_{i \in I} (|X_i| - 1)$.

Dynamic programming can be used to find the global optimum of any pseudo-Boolean optimization problem (i.e., Eq. 1) in time $O(2^w N)$, where w is the tree width of graph $G(V, E)$ [4] and N is the number of variables. In effect, a tree decomposition D provides an ordering of the variables and of the subfunctions so that only w variables are *active* at a time. A variable is active if it appears in a subfunction that is currently being probed by dynamic programming. Once a variable is active, it stays active until a global solution is found for all of the subfunctions that utilize that variable. In general, for NP-Hard problems (such as MAXSAT) the runtime cost of dynamic programming is exponential. However, problems where the tree width is bounded by a constant can be solved in linear time. Thus, there is an advantage in creating a neuron selection architecture that limits the size of the tree width of the variable interaction graph. In the next section we show that if the neuron connections are sufficiently localized and regular, the tree width is automatically limited in size.

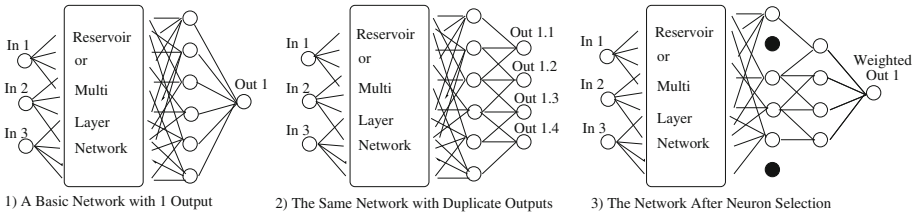


Fig. 1. On the left is a basic network with one output. In the middle figure, the one output neuron is replaced by an ensemble of output neurons, all with the same target output and learning objective. Each output neuron in the ensemble is connected to exactly $k = 3$ neurons in the probe filter layer ($k \ll N$). Normally, there are N neurons in both the probe filter layer and an ensemble of N outputs. Here, to aid visualization, there are 6 neurons in the probe filter layer and only 4 outputs in the ensemble (x_1 is not adjacent to x_6), and $k = 3$. Each output Out 1.i is different in performance because it connects to different neurons in the probe filter layer. The neurons of the probe filter layer are turned on or off by optimizing binary vector $\mathbf{x}$ so as to maximize performance summed across all outputs. In the figure on the right, the black neurons have been turned off, optimizing $\mathbf{x}$ and modifying the architecture. The ensemble of outputs are again collected into a single output weighted by the relative performance of each output in the ensemble.

3 Converting a Neural Network into an NK Landscape

One type of pseudo-Boolean optimization problem that automatically controls for tree width is the *Adjacent NK Landscape* [11,19]. N refers to the number of Boolean variables in vector x, $M = N$ is the number of subfunctions, and $k = K + 1$ is the number of variables that appear in each subfunction f_i. In an Adjacent NK Landscape, Boolean variable x_i appears in subfunction f_i as well as the variables $x_{i+1}, x_{i+2}, \ldots x_{i+K}$. If the Adjacent NK Landscape allows variables to wrap around such that x_1 and x_N are adjacent, then the tree width is $2K$. If variables do not wrap and x_1 and x_N are not adjacent in the Adjacent NK Landscape, then the tree width is K [21].

We will use Fig. 1 to explain how the neuron selection problem can be expressed as a k-bounded pseudo-Boolean optimization problem with bounded tree width. On the left in Fig. 1, we start with a basic neural network with a single output neuron. In this example, there are 3 inputs and 1 output. We assume there is a hidden layer of neurons that feed into the output, or if a reservoir is used, we create a hidden layer of N neurons that feed into the output. We will refer to this layer as the *probe filter* layer. This layer probes the other neurons in the network that feed into this final hidden layer. The probe filter layer must directly connect to the output layer.

We will refer to our architecture as an *NK Ensemble Network*. The *NK Ensemble Network* has an ensemble of N outputs, and each output receives inputs from k neurons in the probe filter layer. All weights in the NK Ensemble Network remain fixed during neuron selection. Only the bit vector $\mathbf{x} \in \{0,1\}^N$ is

optimized. If $x_i = 1$, the i^{th} neuron of the probe filter is turned on, i.e., its activation is used as input for neurons in the output layer connected to it; if $x_i = 0$, the i^{th} neuron is turned off.

We denote the evaluation of the i^{th} output neuron as $f_i(\mathbf{x})$. The subfunction f_i automatically accesses the correct k bits when passed an input of length N. It is also convenient to assume that f_i can take an input of length k or length $N = |\mathbf{x}|$. For example, assume $k = 3$ and that $f_i(\mathbf{x}) = f_i(011)$; this means that $k = 3$ probe filter neurons feed into output neuron i, but the 1st neuron (numbering bits left to right) is currently turned off in $\mathbf{x}$. All of the inputs to f_i must be evaluated once (and only once). Thus, for $k = 3$ we must evaluate $f_i(000), f_i(001), \ldots f_i(111)$. This is done by turning off the correct neurons in the probe filter layer, then doing an evaluation that processes the training data just once, or (e.g., for reinforcement learning) a simulation is used to evaluate the network performance. This means that each subfunction requires 2^k presentations of the training data, or 2^k performance based evaluations. This needs to be done for all N subfunctions.

At most $2^k N$ presentations of the training data are needed to convert the neuron selection problem into an Adjacent NK Landscape. For recurrent networks where the output at time t impacts the input at time $t + 1$, the total number of online evaluations will be exactly $2^k N$; this represents the worst case runtime cost, which is still $O(N)$ for fixed k. For other classes of learning problems this cost might be reduced because subfunction f_i might be evaluated simultaneously and in parallel with other subfunctions (e.g., f_i and f_{i+k} do not interact) using the same presentation of the training data.

Because k is a small constant, each function f_i can be expressed as a lookup table with 2^k entries. Dynamic programming can then be done offline because the neuron selection objective function is now fully captured by the lookup tables. In practice, we have found that the runtime cost of dynamic programming is less than 1% of the entire computation for small k (e.g., $k \leq 6$) and usually takes about the same amount of time as a single feed-forward pass over the training data.

The algorithm for neuron selection follows the 3 illustrations in Fig. 1.

(1) Step 1. Start with a basic network. The weights in the network might be optimized with a weight training algorithm (e.g., back propagation), or weights might be generated randomly. The network must include a probe filter layer, and only neurons in the probe filter layer connect to the output.
(2) Step 2. Assume there are N neurons in the probe filter layer: create an ensemble of N outputs. Let x_i reference the i^{th} neuron in the probe filter. Output neuron "Out 1.i" receives inputs from neurons x_i to x_{i+K}. The performance of output neuron Out1.i is condensed into a single number, P_i, where $f_i(x) = P_i$. This makes is possible to storage each function f_i as a look-up table of size 2^k.
(3) Step 3. Optimize the function: $f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} f_i(\mathbf{x})$ using dynamic programming. Use the optimal solution $\mathbf{x}*$ to select neurons in the probe filter

layer. In this formulation we average over the subfunctions but this has no impact on the form of the optimization problem.

Let z be an input to the neural network. Let $S_{out.i}(z)$ denote the state of ensemble output neuron (e.g. Out 1.i in Fig. 1) after input z is propagated through the network; let $Out_{ensemble}(z)$ denote the weighted output obtained by combining the ensemble:

$$Out_{ensemble}(z) = \frac{1}{N}\sum_{i=1}^{N} \alpha_i S_{out.i}(z) \qquad \text{where} \qquad \alpha_i = \frac{f_i(\mathbf{x}^*)}{\sum_{i=1}^{N} f_i(\mathbf{x}^*)} \tag{2}$$

The weighting vector α is calculated after optimizing the vector $\mathbf{x}$. Thus α_i depends on the performance associated with $f_i(x*)$ and output neurons with better results have higher weights.

4 Experimental Results

All of the experiments in this paper use "Echo State Networks" as a foundation. One motivation for using Echo State Networks is that the neurons in the reservoir have randomly generated weights. Schiller and Steil [17] show that when gradient methods are used to train recurrent neural networks, most of the weight changes occur in the weights that connect to outputs, even if the methods are being used to change all of the weights in the network. We explore the idea that neural networks can be trained using little or no weight optimization.

The term "NK Ensemble Network" will be used to denote networks that have been enhanced by neuron selection.

4.1 Problem One: Mackey-Glass Time Series Prediction

The Mackey-Glass time series problem is a supervised learning problem and a classic benchmark for chaotic time series prediction. The original Echo State Network was successfully applied to this problem [10]. The vector of weights $\mathbf{w_{out}}$ in a given output neuron can be trained by solving a system of linear equations:

$$\mathbf{y_d} = \mathbf{H}\mathbf{w_{out}} \tag{3}$$

where $\mathbf{H}$ is a matrix composed by the inputs of the output layer for each training example and $\mathbf{y_d}$ is the vector of desired values for the output neuron. Equation 3 can be solved [14] by:

$$\mathbf{w_{out}} = (\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y_d} \tag{4}$$

In order to avoid numerical instability, a regularization term can be added to the term inside the parentheses in Eq. 4.

The Echo State Network for this problem has only one input and only one output. These respectively correspond to the points of the time series at instants t and $t + 1$ (the Mackey-Glass time series with delay 17 was used). There are

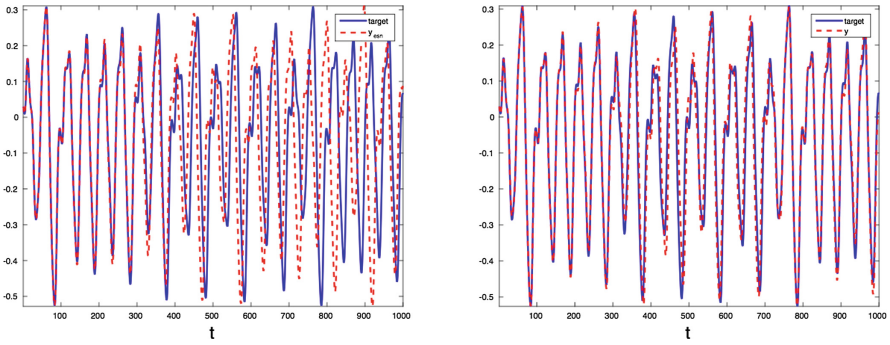


Fig. 2. The leftmost figure illustrates the performance of the standard echo state network for the Mackey-Glass problem. The rightmost figure illustrates the performance of the NK ensemble network. In both figures, the desired output is denoted by the solid blue line, and the actual output is denoted by the dashed red line. The NK echo state network error is 4 times lower than the standard echo state network. (Color figure Online)

two hidden layers between the reservoir and the output. The weights associated with the two hidden layers were obtained using the same weight optimization reported by Jaeger [10]. The inputs of the neurons of the hidden layer 2 receive inputs from 90% of the neurons in hidden layer 1. A bias neuron is used. A linear activation function is used in the neurons of the hidden layer 2 (in order to apply Eq. 4 to train the output weights) and output layer. The neurons of the reservoir use the hyperbolic tangent activation function.

The NK Ensemble Network uses exactly the same standard configuration except an ensemble of N output neurons is used. The second hidden layer functions as the probe filter layer. The weights between the probe filter layer and the ensemble of outputs are generated randomly, then rescaled so that the sum of the weights is equal to 1.

The output neurons of the NK Ensemble Network for inputs z are given by:

$$S_{out.i}(z) = \sum_{q=1}^{N} w^s_{q,i} S_q(z) x_q \tag{5}$$

where $S_q(z)$ is the output of neuron q of the probe filter layer for input z, x_q indicates if the neuron q is turned on (1) or off (0), and the scaled weight $w^s_{q,i}$ is given by:

$$w^s_{q,i} = \frac{w_{q,i}}{\sum_{j=1}^{N} w_{j,i} x(j)} \tag{6}$$

where $w_{q,i}$ are randomly generated in the interval between 0.0 and 1.0.

The weights of the output neurons are then adapted using Eq. 4 (for each output neuron and each combination given by vector $\mathbf{x}$). Finally, look-up tables are generated for each subfunction f_i. Next, the bit vector $\mathbf{x} \in \{0,1\}^N$ is optimized using dynamic programming.

The initial 1000 time steps of the series are used to stabilize the reservoir before the training phase for each learning algorithm. The standard Echo State Network is trained for 2000 time steps of the Mackey-Glass series and tested for additional 300 points. During the test phase, the input of the network at time t is given by the output of the network at time $t-1$.

Learning for the NK Ensemble Network is broken into three phases. In Phase 1, the weights are adapted for 1800 time steps in exactly the same way in which they were for the standard Echo State Network. In Phase 2, the subfunctions f_i are generated. Each subfunction is evaluated for 2^k configurations, and each configuration is evaluated for 200 time steps. This can be thought of as a validation phase where the optimization of the probe filter layer corresponds to a type of model selection. The input of the network at time t is given by the output of the network at time $t-1$. In the validation phase, the terms f_i are computed by the mean squared error for the NK Ensemble Network during 200 time steps. The vector $\mathbf{x}$ is then optimized by the dynamic programming procedure; the cost of the dynamic programming is minimal and less than 1% of the total runtime. In Phase 3, the NK Ensemble Network is tested for generalization.

Both the standard Echo State Network and the NK Ensemble Network were allocated 2000 time steps of the data for learning and 1000 time steps for testing. (The 2000 steps for the NK Ensemble network includes the time needs for dynamic programming.) Both networks used exactly the same reservoir. Both networks were trained and tested using a sample size of 30. During testing, generalization was measured by the function:

$$g = \frac{1}{1 - e_{mse}}$$

where e_{mse} is the mean squared error. $g = 1.0$ represents perfect generalization.

Both networks yield reasonably good prediction during the first 300 steps of testing. However, an examination of Fig. 2 shows that the standard Echo State Network yields poorer performance after time step 300 during the testing phase: the predictions become increasingly worse with time. The NK Ensemble Network continues to make good predictions across all of the testing phase. Overall, the error of the standard Echo State Network is 4 times larger than the error of the NK Ensemble Network. Thus, the NK Ensemble Network is able to improve generalization with little or no additional training cost.

4.2 Problem Two: Double Pole Balancing Without Velocity Inputs

The NK Ensemble Network is next tested on the double pole balancing problem without velocity information [20]. No back propagation was used. No policy iteration was used. The only form of learning was neuron selection. All of the weights in the network were generated randomly.

The reservoir utilizes 60 neurons, with recurrent connections between neurons. Each neuron in the reservoir has recurrent connections to 10% of the neurons in the reservoir. All weights and bias of the NK Ensemble Network are

fixed, being randomly generated between $[-0.6, 0.6]$. After the initialization, the recurrent weights in each reservoir are scaled with a spectral radius equal to 0.95. All neurons use the hyperbolic tangent function as the activation function.

When no velocity information is provided, this problem is difficult; it has also been widely studied [5,7–9,18]. The 3 inputs to the artificial neural network at step t are the scaled cart position and the angles of the two poles:

$$\mathbf{u}(t) = [p_c(t)/p_c^{max}, \quad \theta_1(t)/\theta_1^{max}, \quad \theta_2(t)/\theta_2^{max}]^{\mathrm{T}}$$

where $p_c(t)$ is the cart position, $\theta_i(t)$ is the angle of the i-th pole, and p_c^{max} and θ_i^{max} are the maximum allowed values used to scale the inputs between -1 and $+1$. All neurons use the hyperbolic tangent function with outputs between -1 and $+1$ as the sigmoidal squashing function.

The following objective function has been used by a number of researchers [5,8,9,18].

$$f = t/t_{max} + 9f_{stable}$$

$$f_{stable} = \begin{cases} 0, & \text{if } t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t}(|x_c(i)|+|\dot{x_c}(i)|+|\theta_1(i)|+|\dot{\theta}_1(i)|)}, & \text{otherwise,} \end{cases}$$

where t is the number of time steps that the system is successfully controlled (up to a limit of $t_{max} = 1000$ steps).

The output in the problem posed in this paper is continuous, allowing for greater control. The force (in Newtons) applied to the cart at iteration t when evaluating the i-th output neuron is given by:

$$action(t) = 10S_{out.i}(\mathbf{u}(t)) \tag{7}$$

The state of the neuron selection vector $\mathbf{x}$ is included in the calculation of S_i (Eq. 5). The track length is given by $p_c \in [-2.4, 2.4]$ meters; beyond this range the cart crashes into the ends of the track. The system must keep both poles within $\theta_i \in [-36, 36]$ degrees of vertical. The function f_1 indicates how long the cart and pole system has avoided a failed state (where a pole falls, or the cart crashes). An overall evaluation greater than 1.0 generally means that the system avoided failure for t_{max} time. However, because $t_{max} = 1000$ is small, a bang-bang control strategy might be learned so that even if the controller avoids failure for t_{max} time steps, the system will become increasingly unstable and eventually fail when the system is run for more than t_{max} time steps. The second function f_{stable} indicates the stability of the system during the last 100 time steps if $t \geq 100$. A higher value of f_{stable} means that the system is staying close to the ideal state: close to the center of the track, with small pole angles close to vertical (zero), and with low velocities.

During learning, the system always starts from the state $p_c(0) = \theta_2(0) = \dot{p_c}(0) = \dot{\theta}_1(0) = \dot{\theta}_2(0) = 0$ and $\theta_1(0) = 4.5°$. The mass of cart is 1 kg, the mass of pole 1 is 0.1 kg, the mass of pole 2 is 0.01 kg: length of pole 1 equal to 1 m,

length of pole 1 equal to 0.1 m, coefficient of friction of the cart on the track is 0.0005, the coefficient of friction of the poles equal to 0.000002 [5]. The 4th order Runge-Kutta method with integration step equal to 0.01 was used.

4.3 Comparative Results

Learning was successful 100% of the time across all experiments. To test generalization, the final network was evaluated 625 times, each time with different initial settings for cart position, cart velocity, pole 1 angle, and pole 1 velocity. The angle and velocity for pole 2 are set to zero. The combination of five different initial settings for each variable is considered: 5, 25, 50, 75, and 95% of a reduced range of the variables. With 5 settings and 4 variables, $5^4 = 625$. The evaluation of the generalization test counts the number of positions from which the system is successfully controlled for 1000 steps. This test of generalization has been widely used for the last 20 years [5,7–9].

In Table 1, we report results for the NK Ensemble Network for several different configurations, as well as previously published results. There appears to be no new significant results since 2008.

Gomez, Schmidhumber and Miikkulainen [8] have shown that a wide range of standard reinforcement learning methods do not work well on the problem of balancing two poles on a cart given no velocity information. They used Q-learning with a Multi-Layer Perceptron that mapped state-action pairs to Q-values. They also compared to methods such as Sarsa(λ) with Case Based Function Approximators as well as Sarsa(λ) with a Cerebellar Model Articulation Controller [16]. They concluded these methods were less effective and less efficient compared to neuroevolution based methods such as NEAT [18], ESP [7] and CoSyNE [8]. In this paper and the above studies, a continuous output is learned. The most recent reinforcement work on pole balancing [1] looked at a discrete "bang-bang" controller and provided velocity information as inputs; this work also did not test for generalization.

The NK Ensemble Networks included networks with N = 20 and N = 100, and K = 2, 3, 4, 5. Using just 320 evaluations, the NK Ensemble Networks with $N = 20$ and $K = 3$ yields an average generalization of 304 successes from the 625 possible start states. This level of generalization is similar to the best results previously reported in the literature as reported in Table 1. Increasing N and K improved generalization at the cost of additional evaluations. The best generalization was achieved by setting $N = 100$ and $K = 4$ and then selecting only the "Top 20" best output neurons (based on α_i from Eq. 2) to be included in the ensemble. This was done at no additional runtime cost. This configuration used 3200 evaluations, but the NK Ensemble Network was able to successfully balance the double pole from 490 of the 625 start states on average with a relatively low standard deviation. These generalization results greatly improve on results previously reported in the literature. A runtime analysis shows that 99% of the runtime was spent on feedforward evaluations of the neural network; less than 1% of the time was spent on the dynamic programming optimization.

Table 1. Evaluation results for the NK ensemble network with different values of N and K. The results are also compared to other results in the literature.

Algorithm		Evaluations	Generalization
CE 1996, reference [9]		840,000	300
ESP 1999, reference [7]		169,000	289
ESP 2008, reference [8]		26,342	Not Given
NEAT 2002, reference [18]		33,184	286
NEAT 2008, reference [8]		6,929	Not Given
CoSyNE 2008, reference [8]		3,416	Not Given
NK Ensemble Network,	N=20, K=2	160	229 ±160 s.d.
NK Ensemble Network,	N=20, K=3	320	304 ±154 s.d.
NK Ensemble Network,	N=20, K=4	640	321 ±151 s.d.
NK Ensemble Network,	N=20, K=5	1,280	377 ±126 s.d.
NK Ensemble Network,	N=100, K=2	800	323 ±115 s.d.
NK Ensemble Network,	N=100, K=3	1,600	396 ±108 s.d.
NK Ensemble Network,	N=100, K=4	3,200	437 ±83 s.d.
NK Ensemble Network, "Top 20"	N=100, K=2	800	450 ±79 s.d.
NK Ensemble Network, "Top 20"	N=100, K=3	1,600	478 ±56 s.d.
NK Ensemble Network, "Top 20"	N=100, K=4	3,200	490 ±48 s.d.

5 Conclusions

This paper explores the idea that learning can be achieved by turning on and turning off neurons in an artificial neural system. By posing the neuron selection problem as a pseudo-Boolean optimization problem with bounded tree width, an exact global optimum can be obtained to the neuron selection problem in $O(N)$ time. In this paper neuron selection is empirically evaluated when used in combination with the Echo State Network. However, the method could be used with other multi-layer networks. It should also be noted that the NK Ensemble Network does not require significant tuning to achieve the "right network configuration" in order to learn.

On the Mackey-Glass time series prediction problem the NK Ensemble Network improved generalization and reduced variance across runs compared to the standard Echo State Network.

The NK Ensemble Network is able to learn the control task of balancing two poles on a fixed track with no velocity information. Learning was 100% successful. No back propagation was used. No policy iteration was used. All of the weights in the network were generated randomly. The only form of learning was neuron selection. Learning was much faster compared to other algorithms. But more important, generalization dramatically improved as N and K were increased, and variance in the generalization results decreased.

References

1. Anderson, C., Elliott, D.: Faster reinforcement learning after pretraining deep networks to predict state dynamics. In: International Joint Conference on Neural Networks (2015)

2. Boros, E., Hammer, P.: Pseudo-boolean programming revisited. Discrete Appl. Math. **123**(1), 155–225 (2002)
3. Chechik, G., Meilijson, I., Ruppin, E.: Neuronal regulation: a mechanism for synaptic pruning during brain maturation. Neural Comput. **11**(8), 2061–2080 (1999)
4. Crama, Y., Hansen, P., Jaumard, B.: The basic algorithm for pseudo-boolean programming revisited. Discrete Appl. Math. **29**(2–3), 171–185 (1990)
5. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with analog genetic encoding. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 671–680. Springer, Heidelberg (2006). https://doi.org/10.1007/11844297_68
6. Gao, Y., Culberson, J.: On the treewidth of NK landscapes. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 948–954. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_106
7. Gomez, F., Miikkulainen, R.: Solving non-Markovian control tasks with neuroevolution. In: IJCAI. Morgan Kaufmann (1999)
8. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. J. Mach. Learn. Res. **9**, 937–965 (2008)
9. Gruau, F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding. In: Genetic Programming Conference. Morgan Kaufmann (1996)
10. Jaeger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004)
11. Kauffman, S.A.: The Origins of Order: Self-organization and Selection in Evolution. Oxford University Press, Oxford (1993)
12. Kristiansen, M., Ham, J.: Programmed cell death during neuronal development: the sympathetic neuron model. Cell Death Differ. (Nature Publishing Group) **21**(7), 1025–1035 (2014)
13. Lukoševičius, M.: A practical guide to applying echo state networks. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 7700, pp. 659–686. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35289-8_36
14. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**(3), 127–149 (2009)
15. Roth, K.A., D'Sa, C.: Apoptosis and brain development. Ment. Retard. Dev. Disabil. Res. Rev. **7**, 261–266 (2001)
16. Santamaria, J., Sutton, R., Ram, A.: Experiments with reinforcement learning in problems with continuous state and actions spaces. Adapt. Behav. **6**(2), 163–217 (1998)
17. Schiller, U.D., Steil, J.J.: Analyzing the weight dynamics of recurrent learning algorithms. Neurocomputing **63**, 5–23 (2005)
18. Stanley, K., Miikkulainen, R.: Efficient reinforcement learning through evolving neural network topologies. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 569–577, Morgan Kaufmann (2002)
19. Tomassini, M., Verel, S., Ochoa, G.: Complex-network analysis of combinatorial spaces: the NK landscape case. Phys. Rev. E **78**, 066114 (2008)
20. Wieland, A.P.: Evolving neural network controllers for unstable systems. In: Proceedings of the 1991 International Joint Conference on Neural Networks (IJCNN), vol. 2, pp. 667–673. IEEE (1991)
21. Wright, A.H., Thompson, R.K., Zhang, J.: The computational complexity of N-K fitness functions. IEEE Trans. Evolut. Comput. **4**(4), 373–379 (2000)

What Are the Limits of Evolutionary Induction of Decision Trees?

Krzysztof Jurczuk(✉), Daniel Reska, and Marek Kretowski

Faculty of Computer Science, Bialystok University of Technology,
Wiejska 45a, 15-351 Bialystok, Poland
{k.jurczuk,d.reska,m.kretowski}@pb.edu.pl

Abstract. For typical assessment of applying machine learning or data mining techniques, accuracy and interpretability are usually the most important elements. However, when the analyst is faced with real contemporary big data problems, scalability and efficiency become crucial factors. Parallel and distributed processing support is often an indispensable component of operational solutions.

In the paper, we investigate the applicability of evolutionary induction of decision trees to large-scale data. We focus on the existing Global Decision Tree system, which searches the tree structure and tests in one run of an evolutionary algorithm. Evolved individuals are not encoded, so the specialized genetic operators and their application schemes are used. As in most evolutionary data mining systems, every fitness evaluation needs processing the whole training dataset. For high-dimensional datasets, this operation is very time consuming and to overcome this deficiency, two acceleration solutions, based on the most promising, latest approaches (NVIDIA CUDA and Apache Spark) are presented. The fitness calculations are delegated, while the core evolution is unchanged. In the experimental part, among others, we identify what are dataset dimensions which can be efficiently processed in the fixed time interval.

Keywords: Evolutionary data mining · Decision trees
Parallel and distributed computing · Spark · GPU · CUDA

1 Introduction

Decision trees [13] are one of the most popular forms of knowledge, which can be automatically discovered from the learning dataset. Typical induction algorithm is based on the well-known top-down approach [20]. Such a greedy heuristics, based on the classical divide and conquer schema, proved to be really fast and accurate. On the other hand, it can be easily shown that the resulting trees, even after post-pruning, are very often overgrown and not stable [17].

More global induction methods, especially based on the evolutionary approaches [2], have emerged recently as interesting alternatives. In this type of algorithms, a tree structure, all tests in non-terminal nodes and all predictions in leaves are searched simultaneously. Global methods are clearly more

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 461–473, 2018.
https://doi.org/10.1007/978-3-319-99259-4_37

computationally complex, but the generated decision trees can be significantly simpler, without sacrificing the prediction quality. For large-scale data, however, the potential gains from applying the evolutionary approach may be unachievable, as the population-based and iterative induction can be simply too slow. Moreover, as in many evolutionary data mining algorithms, the whole training dataset should reside in memory, since it is extensively reexamined. As a result, memory constraints may influence the applicability of such methods for larger datasets.

It is clear that the possible success of evolutionary induction of decision trees for large-scale data depends on the availability, easiness of use and costs of use of (parallel or distributed) acceleration solutions. In this paper, we discuss a case study of boosting one existing evolutionary data mining system: Global Decision Tree (GDT) [6,15]. It enables induction of various variants of decision trees, but here only univariate classification trees are considered. The investigations are focussed on economically reasonable approaches, thus, we restrict the hardware used to a small computing cluster or a single graphics processor unit (GPU) accelerator. Two novel parallel and distributed processing solutions are analyzed. The first solution applies general-purpose computation on GPUs (GPGPU) and it is based on NVIDIA CUDA [21] framework. The second one is devoted to computing clusters and it is based on the Apache Spark [24] engine. The limits and constraints imposed by the studied acceleration techniques are identified.

The rest of the paper is organized as follows. First, two acceleration techniques are briefly introduced and their recent applications are mentioned. Then, the original GDT system is presented. In Sect. 3 two considered extensions are detailed. Experimental results are described and discussed in Sect. 4. The paper is concluded and possible future works are sketched in the last section.

1.1 GPU, CUDA

GPUs of modern graphics cards are equipped with hundreds/thousands of small, energy-efficient computing units (GPU cores). Each GPU core, though smaller, simpler and slower than a CPU core, is tuned to be especially efficient at the basic mathematical operations. This simplicity allows many more GPU cores to be crammed into a single chip. Moreover, current GPU architectures are approaching terabytes per second memory bandwidth that, coupled with many computational units, creates an ideal device for handling multiple tasks in parallel and managing workloads efficiently. Thus, not only graphics applications but also GPGPU have gained in popularity [23].

Compute Unified Device Architecture (CUDA) [21] is a programming interface and parallel platform that has revolutionized GPGPU. Although there are some alternatives (like OpenCL), CUDA is the most widespread platform. In CUDA, a GPU is considered as a co-processor to a CPU. It means that a part of CPU's tasks can be delegated to the GPU and be processed by thousands of threads in parallel concurrently to the CPU operations. From a programming perspective, CPU calls a kernel that is a function run on the GPU. Then, many threads are created to run the function. The threads are hierarchically

grouped into thread blocks, which are in turn arranged on a grid. The CUDA GPU memory also has a hierarchical structure [21].

GPGPU is recently widely applied in many computational intelligence methods [4,9,11]. Application of GPUs in evolutionary data mining usually focuses on boosting the performance of the evolutionary process which is relatively slow due to high computational complexity, especially for the large-scale data [5,12].

1.2 Apache Spark

Apache Spark [24] is an open-source distributed computing engine for large-scale data processing and one of the most widely used tools in the ever-growing Big Data ecosystem. Spark architecture is based on a concept of Resilient Distributed Dataset (RDD) - an immutable distributed data structure that provides fault tolerance and can be processed in parallel using high-level APIs.

The main advantages of Spark are its in-memory computing capabilities for iterative algorithms and interactive data exploration. Spark processes data in distributed shared memory model, preferably in the RAM of the cluster nodes. Furthermore, Spark offers a much broader set of high-level functional-style data operators that simplify the implementation of distributed applications.

One of the earliest applications of Spark to evolutionary algorithms was proposed by Deng et al. [7], where the population in a differential evolution method is treated as an RDD and only the fitness evaluation is distributed to workers. Teijeiro et al. [22] also described a parallel differential evolution, focusing on individual's mutation, in both master-slave and island models. As for evolutionary data mining approaches using Spark, fuzzy rule-based classifiers were proposed in [8,18]. In [10] the authors tried to scale a genetic programming solution for symbolic regression and proposed a fitness evaluation service based on Spark.

2 Global Decision Tree System

The GDT system [6,15] enables induction of several types of decision trees, depending among others on the type of a predictive task to be solved (classification or regression), the permitted test types in nodes (univariate or oblique) and the prediction types in leaves (single value or model), etc. All variants of the algorithm share the same typical evolutionary process [16] with an unstructured, fixed size population (default population size: 64 individuals) and a generational selection (ranking linear selection and elitist strategy are applied). In this study, to facilitate understanding and to eliminate less important details, the authors focus only on the simplest classification binary trees with tests based on continuous-valued features and without missing data.

2.1 Representation, Initialisation and Termination

Tree-based representation is well-known in genetic programming, where first attempts at evolving a decision tree were presented by Koza [14]. Following these

ideas, in the GDT system, decision trees are not specially encoded and they are processed in their actual form. In non-terminal nodes, typical inequality tests with two outcomes are used, but only precalculated candidate thresholds are considered as potential splits.[1]

An initial individual is created by applying a simple top-down algorithm to randomly chosen small sub-samples of the original training data (default: 10% of the training dataset, but not more than 500 objects), which provides a high degree of heterogeneity of the initial population and is not computationally complex. Among objects located in the considered node, two objects from different classes (so-called *mixed dipole*) are randomly chosen. An effective test that separates these two objects into subtrees is randomly created, taking into account only attributes with different feature values. The recursive partitioning is finished when all training objects in a node are characterized by the same class or the number of objects in a node is lower than the predefined value (default value: 5) or the maximum tree depth is reached (default value: 10). Finally, the resulting tree is post-pruned based on the fitness function.

Evolution terminates when the fitness of the best individual in the population does not improve during the fixed number of generations (default: 1 000) or the maximum number of generations is reached (default value: 1 000).

2.2 Genetic Operators

In the GDT system, there are two specialized genetic operators corresponding to classical mutation applied to a single individual (default probability: 0.8), or to crossover that recombines two individuals (default probability: 0.2).

A mutation operator begins by randomly choosing the node type (equal probability of selecting a leaf node or an internal node), but if the mutation of one type is not possible, the other type is chosen. The ranked list of nodes of the selected type is created, and a mechanism analogous to the ranking linear selection is applied to decide which node will be affected. In case of internal nodes, the ranking takes into account both location (level) of the node in the tree and the reclassification accuracy of each node, whereas for leaves only the second factor is considered. It should be noticed that a modification of a test in a root node affects the whole tree and can have a large impact. On the other hand, mutating an internal node in the lower parts of the tree has only a local impact. As a result, nodes on higher levels of the tree are mutated with lower probability and among nodes on the same level, the reclassification quality is used to sort them. Less accurate leaves are mutated with higher probability and homogenous leaves (all training instances from the same class) are not mutated at all.

There are a few possible mutation variants, which can be performed on internal nodes:

– a test can be modified by shifting a threshold value;

[1] A candidate threshold for the given attribute is defined as the midpoint between such a successive pair of objects in the sequence sorted by the increasing value of the attribute, in which the objects are characterized by different classes.

- a test can be replaced by another test existing in a tree or by a new one. New tests can be created based on randomly chosen dipoles (like in initial population) or locally searched according to some optimality criteria (this can be called memetic extension);
- one subtree can be replaced by another subtree from that node;
- a node can be pruned into a leaf.

Considering a mutation of a leaf node, the range of variants is more modest: a leaf can be just transformed into a subtree.

A crossover operator begins by randomly selecting two trees (and nodes in each of them) that will be affected. There are a few variants of recombination:

- exchange subtrees, branches or only tests associated with nodes (if possible); such an exchange can be purely random or can use a mixed dipole as a guide;
- transfer subtrees asymmetrically where the subtree of the first/second individual is replaced by a new one that was duplicated from the second/first individual. The replaced subtree starts in the node denoted as a receiver, and the duplicated subtree starts in the node denoted as a donor. It is worth to note that different preferences could be used for choosing donor and receiver sites: the receiver node should have a high classification error because it is replaced by the donor node that should have a small value of classification error as it is duplicated.

For both genetic operators each time a choice of operator variant is random, but only valid variants are considered. The default probability distribution of variants is uniform.

2.3 Fitness Function

In most of the data mining system, the first and the most important objective is to find the predictor with the highest classification quality. The main problem with such an objective is that there is no possibility to measure classifier performance in advance. We could only estimate the quality on a given dataset and typically one can only estimate the classifier performance on the training dataset. However, it is well known, that due to the over-fitting problem, a classifier which perfectly reclassifies the training dataset usually performs much worse on unseen objects. The second objective, which is often indicated, is devoted to the classifier simplicity and it can be expressed by the number of nodes. And hopefully, putting emphasis also on a classifier simplicity could be a good way to prevent the over-fitting.

In the GDT system, many forms of the single-objective or multi-objective fitness function are available. As, in this paper, only univariate classification trees are considered, the simplest weighted form of the fitness function is used:

$$Fitness(T) = Accuracy(T) - \alpha * Size(T), \tag{1}$$

where $Accuracy(T)$ represents the classification quality of the tree T estimated on the training dataset, $Size(T)$ is the number of nodes in T and α is the user-supplied parameter (default value: 0.001). The second part of the equation works

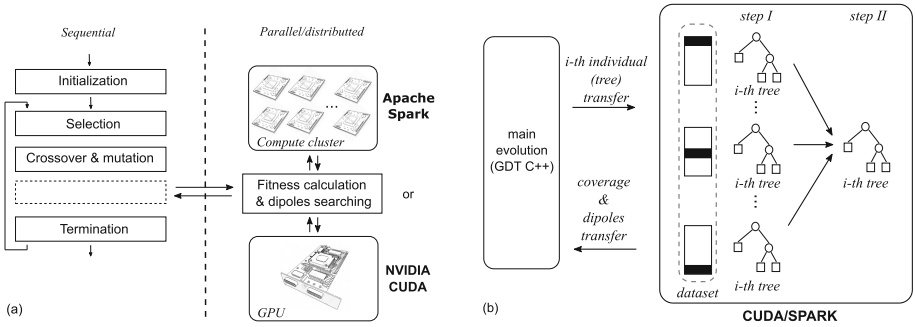


Fig. 1. Evolutionary induction accelerations: (a) general idea, (b) more details concerning processesing an individual in parallel by blocks/workers (in case of CUDA/SPARK acceleration) (step I) as well as reducing/merging results (step II).

as a penalty term and helps to mitigate the over-fitting problem. A similar solution can be identified in the well-known cost-complexity pruning from the CART system [3]. It should be at least mentioned that the value of the α parameter can be usually tuned up for a given dataset, especially if the perfect reclassification cannot be expected, but the parameter tuning is far outside the paper's scope.

3 Boosted GDT Versions

The general idea of GDT accelerations is illustrated in Fig. 1(a). The following operations: initialization of the population as well as selection of the individuals remain unchanged compared to original GDT system. The reason why these initial steps are not accelerated is that the initial population is created only once on a small fraction of the dataset. In the evolutionary loop, also other relatively fast operations like genetic operations (without individual evaluation) are run in a sequential manner. After successful application of crossover or mutation, there is a need to evaluate the individuals. It is the most time-consuming operation since all objects in the training dataset need to be passed through the tree starting from the root node to an appropriate leaf. Thus, this operation is isolated and accelerated by one of the two solutions: CUDA- or Spark-based.

3.1 CUDA Based Acceleration

A GPU-based solution begins by sending the whole dataset to the GPU [12]. This CPU to GPU transfer is done only once and the data is saved in the allocated space in the global memory. Thus, all objects of the dataset are accessible for all threads at any time.

The CPU controls the evolutionary induction. The GPU is called to perform calculations when there is a need to evaluate an individual after successful crossover and/or mutation. At first, the affected individual is sent to the GPU

(Fig. 1(b)). Then, the CPU asks the GPU to take on some of its work. Two kernel functions are called. The first kernel is called to propagate objects from the tree root to the leaves. Next, the second kernel function merges information about the objects' location in the leaves, calculates class distributions and classification errors and finally propagates them from the leaves toward the tree root. The obtained tree statistics (like coverage, errors) as well as dipoles are sent back to the CPU that uses them to update the affected individual.

The first kernel function uses the data decomposition strategy (step I in Fig. 1(b)). At first, the whole dataset is spread into smaller parts that are processed by different GPU blocks. Next, in each block, the assigned objects are further spread over the threads. Each GPU block makes a copy of the evaluated individual that is loaded into the shared memory. This way the threads process the same individual in parallel but handle different chunks of the data.

At the end of the first kernel function, in each tree leaf the number of the objects of each class that reach that particular leaf is stored. However, these values are spread over GPU blocks. Thus, the second kernel function is called (step II in Fig. 1(b)). It merges information from multiple copies of the individual allocated in each GPU block. This operation sums the counters from copies of the individual, and the total number of objects of each class in each tree leaf is obtained. Finally, in the second kernel function reclassification errors in each leaf are calculated. Then, all gathered information: class distribution and errors are propagated from the leaves towards the root node.

To improve the algorithm's performance, the CPU does not have a direct access to the objects that fall into particular nodes of the tree. The propagation of the instances is performed only on the GPU (in contrast to the sequential version). However, some variants of the mutation operator require (object) dipoles to construct a new test in an internal node. This is why the GPU also provides the CPU with two objects of each class in each tree node. In the first kernel, such objects are randomly selected from the objects that reach particular leaves. In the second kernel function, when the multiple copies of the tree are merged, among the available objects again two objected are randomly selected. The CPU using these two objects (of each class in each tree nodes provided by the GPU) can quickly and easily constitute the desired dipoles.

3.2 Spark Based Acceleration

The proposed Apache Spark-based acceleration relies on the distribution of the dataset over a Spark cluster and parallelization of its processing, while the rest of the evolution is unaffected in principle and is realized sequentially.

The Spark-based approach uses a multi-process architecture: while the original GDT system is a native C++ application, Spark is written in the Scala language and its processes run on the Java Virtual Machine (JVM). The Spark processes consist of a single Driver that dispatches the work to a multiple Workers that run on the cluster worker nodes. Both the Spark Driver and GDT applications are running on the same machine and utilise named pipes mechanism for inter-process communication. As a result, the core evolution is performed in

GDT process (C++), whereas the distributed object propagation and dipoles searching procedures, re-implemented in Java, are realized by Spark (JVM).

The proposed approach is based on an implementation described in [19], which was modified and optimized to accommodate big datasets processing. The method starts with the loading of the training dataset, which is processed line-by-line and transformed into an RDD of objects representing packages of observations (1 000 obs. in a package by default). This "packing" operation is highly beneficial from the memory usage standpoint, as it reduces the number of objects in RDD for the given dataset, minimizing the overhead of RDD data structures. Next, the observation RDD is split into a number of partitions that are then cached in the cluster memory.

To prevent data skew, the dataset should be split into partitions of even size and uniformly distributed over the nodes. In our solution, each observation package is randomly assigned to a group with a numeric ID, where the group number equals the number of partitions. The partitions number depends on the data size, with the usual range of 1 to 4 partitions per single worker CPU core.

During the evolution, the GDT process sends a request with a single tree data to the Spark Driver. During the induction, all observations are passed through the transferred decision tree and distributions of classes and dipoles in its leaves are obtained. The parallel processing is realized by typical pair of `map-reduce` operations evoked on the grouped RDD (see Fig. 1(b)). Each dataset partition group emits a locally processed copy of the tree (`map(group)` $\rightarrow$ `tree`) and the local trees are then reduced into a final result (`reduce(tree1, tree2)` $\rightarrow$ `tree3`). During the reduction, the class distributions are simply merged, while the dipoles are reduced implicitly by selecting the dipoles from one of the trees. Finally, the error calculations and propagation of classes and dipoles in the final tree are performed and the results are sent back to the GDT process, where the overall accuracy is estimated. The process ends when the last tree is processed.

4 Experiments

Experimental validation was performed on an artificially generated dataset called *chess* with two 2 real-values attributes and objects arranged on a 3 $\times$ 3 chessboard (Fig. 2). It is a dataset for which moderate sized decision trees are induced. We used the synthetic dataset to scale it freely, unlike real-life datasets. We examined various numbers of objects, from hundreds of thousands to a few billions. All presented results correspond to averages of 5–10 runs and were obtained with a default set of parameters from the sequential version of the GDT system [6]. As we are focused in this paper only on size and time performance of the GDT system, results for the classification accuracy are not included. However, for the tested dataset, the GDT system managed to induce trees with optimal structures and accuracy about 99% [15].

GPU experiments were performed on a workstation equipped with Intel Xeon E5-2620 v4 (20 MB Cache, 2.10 GHz), 256 GB RAM, and running Ubuntu 16.04. The sequential algorithm was implemented in C++ and compiled with gcc 5.4.0.

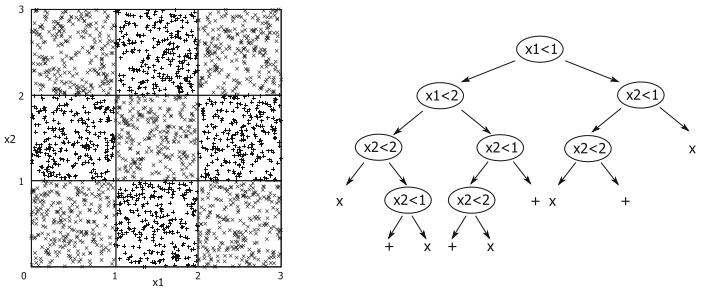


Fig. 2. Examples of analyzed *chess3x3* dataset variant and the corresponding ideal structure classification tree.

The GPU-parallelization was implemented in CUDA-C and compiled by nvcc CUDA 7.5/8.0 [1] (single-precision arithmetic). We tested two NVIDIA GPU cards: *(i)* GeForce GTX 780 (3 GB memory, 2 304 cores) and *(ii)* Pascal P100 (12 GB memory, 3 584 cores). The first GPU card is the consumer line GeForce GPU, while the second one is the professional-level GPU accelerator that currently costs about 5 000 $ (almost 10 times more than the first one).

Apache Spark (version 2.2.0) was deployed on a cluster of 18 workstations with a quad-core Intel Xeon E3-1270 3.4 GHz CPU, 16 GB RAM, running Ubuntu 16.04 and connected by a Gigabit Ethernet network. 16 worker nodes were used by Spark executors, one node was dedicated to Spark Master and HDFS NameNode (Hadoop 2.7.3) and the last node was running Spark Driver and GDT C++ processes. The experiments were performed on 4, 8 and 16 workers, which corresponds, respectively, to 16, 32 and 64 CPU cores in total.

We are interested in estimating the size of the dataset which can be processed on the given platform/hardware in fixed amount of time: 1 min, 1 h and 1 day (Tables 1 and 2). Table 1 concerns the GPU-accelerated solution. We see that both GPU cards provide a significant boost in training dataset processing. Pascal P100 is able to handle 1 million of objects in 1 min. In 1 h, the size of the processed dataset increases to nearly 100 millions of objects.

Comparing GPUs, we see that a cheaper one (GTX 780) gives about twice worse results than Pascal P100. However, this performance difference decreases with the increase of the dataset size. In all cases, for the maximum datasets that can be stored in each GPU memory, the induction does not last longer than 1 day. The time of processing those maximum datasets is included in Table 3.

The scale of the performance improvement is more visible when comparing the sequential and GPU-accelerated versions of the GDT system. For example, as regards 1 h, results show that the GPU-supported version is able to handle the dataset greater by two orders of magnitude (200 000 objects by the sequential version vs 84 000 000 objects with a support of Pascal P100). Comparing results of the GPU-based acceleration and OpenMP parallelization using eight CPU cores, similarly, we see that the first solution wins.

Table 1. The maximum size of the *chess3x3* dataset variant which can be completely processed in the given period of time by GPU-accelerated GDT system. In addition, results for a sequential CPU version as well as an OpenMP parallelization using eight CPU cores are provided.

GPU card/period	1 min	1 h	1 day
GTX 780	530 000	57 000 000	256 000 000*
Pascal P100	966 000	84 000 000	1 033 000 000*
Sequential CPU	1 200	275 000	3 500 000
OpenMP (8 CPU cores)	45 000	970 000	14 000 000

*The processing time of the maximum datasets (that can be stored in each GPU memory) was shorter than 1 day, see Table 3.

Table 2. The maximum size of the *chess3x3* dataset variant which can be completely processed in the given period of time by Spark-accelerated GDT system.

Period number of workers	1 h	1 day
4	13 000 000	500 000 000
8	20 000 000	1 250 000 000
16	35 000 000	2 500 000 000

Table 3. The maximum size of the *chess3x3* dataset which can be completely processed by the given platform/hardware. Processing time is also included.

Solution		Dataset size	Time
GPU	GTX 780	256 000 000	5 h
	Pascal P100	1 033 000 000	17 h
Spark	4 workers	900 000 000	33 h
	8 workers	1 850 000 000	35.5 h
	16 workers	3 900 000 000	38 h

Concerning the Spark-based acceleration, we see that Spark deployed on all 16 nodes can process a dataset of 35 millions objects in 1 h period (Table 2). It is less than the best Pascal P100 GPU result of 84 millions, but within the same order of magnitude. Furthermore, the 1 min execution time cannot be achieved due to the framework and networking overhead. The algorithm usually processes about 80 thousand trees during its execution and the overhead of Spark can be from 6 to 8 milliseconds for every tree in smaller datasets processed in 1 h. This gives about 9 min of total overhead for the entire algorithm. In the 24 h period, however, the datasets are significantly bigger and the framework impact is not as noticeable. In the end, Spark can process 2.5 billion observations in one day, outperforming the GPUs due to their memory limitations.

We are also interested in verifying how big datasets are able to be processed in the given platform/hardware, taking into account available memory restrictions (see Table 3). Concerning GPU cards, the dataset size is strictly limited by their global memory sizes. Since GTX 780 is equipped with 4 times less memory than Pascal P100, it is able to process about 4 times smaller datasets. Pascal P100 handles over than 1 billion objects in less than 1 day. Because of very long computation time, we did not even try to process these datasets either by the

sequential version or OpenMP-based parallelization. Each year, NVIDIA releases new GPUs with faster and larger memory, thus, it is only a matter of short time when GPU cards with 24, 48, etc. GB of memory appear.

Spark, given its Big Data processing capabilities, excels with bigger datasets. The maximum dataset processed on the entire cluster has 3.9 billions objects (see Table 2), which corresponds to 78 GB of raw text data and 96 GB in memory-cached RDD. The results show datasets that can be processed on the cluster in stable and efficient manner, without swapping, excessive JVM garbage collection and with the dataset fully cached in the memory. Technically it is possible to configure the RDD to "spill" into disk storage if it does not fit completely in the memory, but this mode of operation results in drastic performance degradation (reading from memory vs reading from disk). Objects in RDD can also be serialized for space efficiency, but this process also comes with a performance penalty due to necessary deserialization. We observed over ten-fold slowdown with serialized objects, but achieved up to 3:1 compression ratio.

The main advantage of Spark is its capability to easily scale with the size of the cluster. The results in Table 2 show that the size of the dataset can basically double with twice the number of nodes. The same application can also run without any modification on a cluster with potentially thousands of nodes.

5 Conclusions

In this paper, we investigate the applicability of evolutionary induction of decision trees for large-scale data. We show that boosted solutions are able to process really large-scale data, even up to billions of objects. It is clear that the CUDA-based acceleration is generally faster but limited by the size of the GPU memory. On the other hand, the Spark-based solution is preferable if a dataset becomes huge, in our case exceeds one billion of objects. Moreover, an unmodified Spark solution can be easily scaled up just by adding more hardware to the cluster.

In this work, we focus on the dataset dimension expressed as a number of objects. In future works, we would also like to investigate the influence of the number of features. This could be especially interesting for genomic data where the number of features is often large. We also plan to extend the GPU-based solution into a framework where one can easily add more GPUs to distribute dataset over them and push the data size limit.

Acknowledgments. This work was supported by the grant S/WI/2/18 from BUT founded by Polish Ministry of Science and Higher Education.

References

1. NVIDIA Developer Zone - CUDA Toolkit Documentation (2018). https://docs.nvidia.com/cuda/cuda-c-programming-guide/
2. Barros, R.C., Basgalupp, M.P., De Carvalho, A.C., Freitas, A.A.: A survey of evolutionary algorithms for decision-tree induction. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **42**(3), 291–312 (2012)

3. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. CRC Press, Boca Raton (1984)
4. Cano, A.: A survey on graphic processing unit computing for large-scale data mining. WIREs: Data Min. Knowl. Discov. **8**(1), e1232 (2018)
5. Chitty, D.: Improving the performance of GPU-based genetic programming through exploitation of on-chip memory. Soft Comput. **20**(2), 661–680 (2016)
6. Czajkowski, M., Kretowski, M.: Evolutionary induction of global model trees with specialized operators and memetic extensions. Inf. Sci. **288**, 153–173 (2014)
7. Deng, C., Tan, X., Dong, X., Tan, Y.: A parallel version of differential evolution based on resilient distributed datasets model. In: Gong, M., Pan, L., Song, T., Tang, K., Zhang, X. (eds.) BIC-TA 2015. CCIS, vol. 562, pp. 84–93. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-49014-3_8
8. Ferranti, A., Marcelloni, F., Segatori, A., Antonelli, M., Ducange, P.: A distributed approach to multi-objective evolutionary generation of fuzzy rule-based classifiers from big data. Inf. Sci. **415–416**, 319–340 (2017)
9. Fonseca, A., Cabral, B.: Prototyping a GPGPU neural network for deep-learning big data analysis. Big Data Res. **8**, 50–56 (2017)
10. Funika, W., Koperek, P.: Towards a scalable distributed fitness evaluation service. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) PPAM 2015. LNCS, vol. 9573, pp. 493–502. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32149-3_46
11. Jinjing, L., Qingkui, C., Bocheng, L.: Classification and disease probability prediction via machine learning programming based on multi-gpu cluster mapreduce system. J. Supercomput. **73**(5), 1782–1809 (2017)
12. Jurczuk, K., Czajkowski, M., Kretowski, M.: Evolutionary induction of a decision tree for large-scale data: a GPU-based approach. Soft Comput. **21**(24), 7363–7379 (2017)
13. Kotsiantis, S.B.: Decision trees: a recent overview. Artif. Intell. Rev. **39**(4), 261–283 (2013)
14. Koza, J.R.: Concept formation and decision tree induction using the genetic programming paradigm. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 124–128. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0029742
15. Kretowski, M., Grzes, M.: Evolutionary induction of mixed decision trees. Int. J. Data Warehous. Min. (IJDWM) **3**(4), 68–82 (2007)
16. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Heidelberg (1996). https://doi.org/10.1007/978-3-662-03315-9
17. Murthy, S.K.: Automatic construction of decision trees from data: a multi-disciplinary survey. Data Min. Knowl. Discov. **2**(4), 345–389 (1998)
18. Pulgar-Rubio, F.J., Rivera-Rivas, A.J., Pérez-Godoy, M.D., González, P., Carmona, C.J., del Jesus, M.J.: MEFASD-BD: multi-objective evolutionary fuzzy algorithm for subgroup discovery in big data environments - a MapReduce solutioon. Knowl.-Based Syst. **117**, 70–78 (2017)
19. Reska, D., Jurczuk, K., Kretowski, M.: Evolutionary induction of classification trees on spark. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds.) ICAISC 2018. LNCS (LNAI), vol. 10841, pp. 514–523. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91253-0_48
20. Rokach, L., Maimon, O.: Top-down induction of decision trees classifiers-a survey. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **35**(4), 476–487 (2005)
21. Storti, D., Yurtoglu, M.: CUDA for Engineers : An Introduction to High-Performance Parallel Computing. Addison-Wesley, New York (2016)

22. Teijeiro, D., Pardo, X.C., González, P., Banga, J.R., Doallo, R.: Implementing parallel differential evolution on spark. In: Squillero, G., Burelli, P. (eds.) EvoApplications 2016. LNCS, vol. 9598, pp. 75–90. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31153-1_6
23. Yuen, D., Wang, L., Chi, X., Johnsson, L., Ge, W., Shi, Y.: GPU Solutions to Multi-scale Problems in Science and Engineering. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-16405-7
24. Zaharia, M.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016)

Tutorials and Workshops at PPSN 2018

Tutorials at PPSN 2018

Gisele Lobo Pappa[1(✉)], Michael T. M. Emmerich[2], Ana Bazzan[3], Will Browne[4], Kalyanmoy Deb[5], Carola Doerr[6], Marko Đurasević[7], Michael G. Epitropakis[8], Saemundur O. Haraldsson[9], Domagoj Jakobovic[7], Pascal Kerschke[10], Krzysztof Krawiec[11], Per Kristian Lehre[12], Xiaodong Li[13], Andrei Lissovoi[14], Pekka Malo[15], Luis Martí[16], Yi Mei[4], Juan J. Merelo[17], Julian F. Miller[18], Alberto Moraglio[19], Antonio J. Nebro[20], Su Nguyen[21], Gabriela Ochoa[9], Pietro Oliveto[14], Stjepan Picek[22], Nelishia Pillay[23], Mike Preuss[10], Marc Schoenauer[24], Roman Senkerik[25], Ankur Sinha[26], Ofer Shir[27,28], Dirk Sudholt[14], Darrell Whitley[29], Mark Wineberg[30], John Woodward[31], and Mengjie Zhang[4]

[1] Federal University of Minas Gerais, Belo Horizonte, Brazil
glpappa@dcc.ufmg.br
[2] Leiden University, Leiden, Netherlands
[3] Federal University of Rio Grande do Sul, Porto Alegre, Brazil
[4] Victoria University of Wellington, Wellington, New Zealand
[5] Michigan State University, East Lansing, USA
[6] Sorbonne University, Paris, France
[7] University of Zagreb, Zagreb, Croatia
[8] University of Patras, Patras, Greece
[9] University of Stirling, Stirling, UK
[10] University of Müenster, Müenster, Germany
[11] Poznan University of Technology, Poznan, Poland
[12] University of Birmingham, Birmingham, UK
[13] RMIT University, Bundoora, Australia
[14] University of Sheffield, Sheffield, UK
[15] Aalto University, Helsinki, Finland
[16] Universidade Federal Fluminense, Niterói, Brazil
[17] University of Granada, Granada, Spain
[18] University of York, York, UK
[19] University of Exeter, Exeter, UK
[20] University of Malaga, Malaga, Spain
[21] La Trobe University, Melbourne, Australia
[22] TU Delft, Delft, Netherlands
[23] University of Pretoria, Pretoria, South Africa
[24] Institute for Research in Computer Science and Control, Ile de France, France
[25] Tomas Bata University, Zlín, Czech Republic
[26] Indian Institute of Management Ahmedabad, Ahmedabad, India
[27] Tel-Hai College, Tel Hai, Israel
[28] Migal-Galilee Research Institute, Kiryat Shmona, Israel
[29] Colorado State University, Fort Collins, USA
[30] University of Guelph, Guelph, Canada
[31] Queen Mary University of London, London, UK

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 477–489, 2018.
https://doi.org/10.1007/978-3-319-99259-4_38

Abstract. PPSN 2018 features a total of 23 free tutorials covering a broad range of topics in evolutionary computation and related areas. From theory and methods to applications and computer implementations, and from introductory to advanced, the PPSN 2018 tutorial program offers participants the opportunity to learn more about both well-established and ongoing research in this field.

1 Welcome from the Tutorial Chairs

PPSN 2018 received a surprisingly large number of high-quality tutorial proposals, from which 23 tutorials were selected for presentation at the conference. All tutorials are scheduled on the first two days of the conference, which are exclusively reserved for tutorial presentations and workshops.

The PPSN 2018 tutorial program features a wide range of topics, from theory to applications and computer implementations. Two hands-on tutorials cover the implementation of evolutionary algorithms (EAs) in the cloud and in the jMetal framework. Other tutorials focus on applications in security (cryptology, anomaly and intrusion detection), software engineering (genetic improvement), and dynamic scheduling.

In addition to introductory tutorials on mathematical programming, two genetic programming (GP) variants, learning classifier systems, and multiagent systems, a number of advanced techniques also deserve attention. Adaptive parameter choices, deterministic search operator design in Gray Box Optimization, and automated algorithm design with hyper-heuristics are covered, as well as algorithms for bi-level and multi-modal optimization.

Finally, there is a group of tutorials oriented towards the modelling, analysis and visualization of search-space structures, fitness landscapes, and evolutionary algorithm processes and their dynamics. Three tutorials on the theoretical analysis of population-based EAs, parallel EAs, and GP complete the programme.

We invite all PPSN participants to explore the wide range of topics discussed in the selected tutorials, and wish them an enjoyable conference!

Gisele L. Pappa and Michael T. M. Emmerich
PPSN 2018 Tutorial Chairs.

2 Tutorial Abstracts

2.1 Adaptive Parameter Choices in Evolutionary Computation

Tutorial Speaker: Carola Doerr, *Sorbonne University (France).*

Tutorial Abstract: Evolutionary algorithms and other popular black-box optimization techniques are highly parametrized algorithms. To run these algorithms, we typically need to decide upon their population sizes, mutation strengths, crossover rates, selective pressure, etc. This parametrization allows to adjust the behavior of the algorithms to the problem at hand. The chosen

parameter values can have a decisive influence on performance. We thus need to select them with care.

Unfortunately, the identification of good parameter values still is one of the most challenging tasks in evolutionary computation. What complicates the parameter selection problem is the observation that different parameter values can be optimal in different stages of the optimization process. In the beginning of an optimization process, for example, one may want to allow for more exploration, while later on we may prefer a more focused search ("exploitation"). This observation calls for adaptive parameter choices, which automatically adjust the parameter values to the current state of the optimization process.

Adaptive parameter choices are today standard in continuous optimization. Quite surprisingly, however, this is not the case in discrete optimization, where they play only a rather marginal role. A paradigm change towards a more systematic use of non-static parameter choices is much needed. This tutorial aims to contribute to this goal, by providing an in-depth discussion of online parameter selection techniques. We survey both experimental and theoretical results, which demonstrate the unexploited potential of non-static parameter choices.

2.2 Applications of Genetic Programming in Dynamic Scheduling

Tutorial Speakers: Domagoj Jakobovic and Marko Đuraseviæ, *University of Zagreb, Croatia*, Yi Mei and Mengjie Zhang, *Victoria University of Wellington (New Zealand)* and Su Nguyen, *La Trobe University (Australia).*

Tutorial Abstract: Scheduling problems are encountered in many real-world situations and scenarios. In real world, the problem is often dynamic, and unpredicted new jobs arrive in real time. To solve scheduling problems under dynamic conditions various problem-specific heuristics, called dispatching rules, have been designed. However, manually designing such heuristics is a difficult and lengthy process. Therefore, a great deal of research is focused on automatically designing new scheduling heuristics. Genetic programming is usually the method of choice for generating new dispatching rules, since in numerous occasions it generated good dispatching rules for various difficult scheduling environments. The tutorial will cover recent developments in the automatic generation of dispatching rules, as well as outline several new research directions in this field, such as multi-objective heuristic generation, application of ensemble learning methods, construction of surrogate models, etc. The tutorial will help interested researchers to acquire an overview of this emerging and interesting research area and to understand the key ideas and challenges for future studies.

2.3 A Small World Hidden in Evolutionary Computation Techniques

Tutorial Speaker: Roman Senkerik, *Tomas Bata University (Czech Republic).*

Tutorial Abstract: This tutorial represents an insight into an attractive open research task, which is a novel method for visualizing the dynamics of evolutionary and swarm-based algorithms in the form of networks. The idea is based on the similarity in interactions between individuals in the metaheuristics algorithms and for example, users of social networks, linking between web pages, etc. The population is visualized as an evolving complex network that exhibits non-trivial features. The features like clustering, centralities, communities, and many more, offer a clear description of the population under evaluation. This tutorial shows the differences between the types of complex networks used, variations in building complex networks to capture the population dynamics of evolutionary algorithms or communication inside swarm-based algorithms, investigation on the time development of the network. It also shows several successful utilization of complex networks attributes for the performance improvements through the adaptive population as well as parameter/strategy control, further the possibility of controlling the evolution through complex network features.

2.4 Bio-inspired Approaches to Anomaly and Intrusion Detection

Tutorial Speakers: Luis Martí, *Universidade Federal Fluminense (Brazil)* and Marc Schoenauer, *Institute for Research in Computer Science and Control (France).*

Tutorial Abstract: Intrusion detection systems (IDSs) have gained a substantial attention because of its high-impact safety and security applications. Two main approaches are used when building those systems: (i) misuse-based and (ii) anomaly-based detection. While the former focuses on detecting attacks that follow a known pattern or signature, the latter is interested in building a model representing the system's nominal behavior while assuming all deviated activities to be anomalous or intrusions, and, therefore provide a more robust solution. Bio-inspired approaches have been proposed to address the problem of anomaly-based intrusion detection, with artificial immune systems (AISs) being the most recognizable approach of all. However, recent developments in the area of single and multi-criterion evolutionary computing, adversarial co-evolutionary modeling and simulation have served a foundation for novel and better performing bio-inspired IDS that have yielded competitive results.

This tutorial will present the anomaly detection topic, its peculiarities and revise the current state of the art on this topic, departing from classical machine learning approaches, presenting the current state-of-the-art methods and analyze how and why those methods have been shown to outperform many of the currently established approaches.

2.5 Cartesian Genetic Programming

Tutorial Speaker: Julian F. Miller, *University of York (UK).*

Tutorial Abstract: Cartesian Genetic Programming (CGP) is a well-known and respected form of Genetic Programming. It uses a very simple integer address-based genetic representation of a program in the form of a directed graph. In a number of studies, CGP has been shown to be comparatively efficient to other GP techniques. The classical form of CGP has undergone a number of developments which have made it more useful, efficient and flexible in various ways. These include self-modifying CGP (SMCGP), cyclic connections (recurrent-CGP), encoding artificial neural networks and automatically defined functions (modular CGP). SMCGP uses functions that cause the evolved programs to change themselves as a function of time. Recurrent-CGP allows evolution to create programs which contain cyclic, as well as acyclic, connections. CGP encoded artificial neural networks represent a powerful training method for neural networks. CGP has been applied successfully to a variety of real-world problems, such as digital circuit design, visual object recognition and classification.

2.6 Cloud-y Evolutionary Algorithms

Tutorial Speaker: J.J. Merelo, *University of Granada (Spain).*

Tutorial Abstract: This tutorial will describe how cloud computing is a new paradigm that changes the way applications are designed and deployed, and how it can be put to use in a scientific computing environment. It will be a practical tutorial with examples and tools that are used nowadays by companies and institutions. The examples used will be taken from evolutionary computation, although its application is widespread.

2.7 Computational Complexity Analysis of Genetic Programming

Tutorial Speaker: Pietro Oliveto and Andrei Lissovoi, *University of Sheffield (UK).*

Tutorial Abstract: Genetic Programming is an evolutionary computation paradigm that aims to evolve computer programs. Compared to the great number of successful applications of GP that have been reported, the theoretical understanding of its underlying working principles lags far behind. In particular, the identification of which classes of computer programs can be provably evolved efficiently via GP has progressed slowly compared to the understanding of the performance of traditional evolutionary algorithms (EAs) for function optimisation. The main reason for the slow progress is that the analysis of GP systems is considerably more involved due to the variable length of programs compared to the fixed solution representation used in EAs and because understanding candidate program quality over all possible inputs is unfeasible. Nevertheless, nowadays it is possible to analyse the time and space complexity of GP algorithms for evolving proper programs with input/output relationships where

the fitness of candidate solutions is evaluated by comparing their accuracy on input/output samples of a polynomially-sized training set (e.g., Boolean Functions). In this tutorial, we give an overview of the recent results outlining the techniques used and the challenges involved.

2.8 Evolutionary Algorithms and Hyper-Heuristics

Tutorial Speaker: Nelishia Pillay, *University of Pretoria (South Africa).*

Tutorial Abstract: Evolutionary algorithms have played a pivotal role in the advancement of hyper-heuristics. The aim of the tutorial is to firstly provide an introduction to evolutionary algorithm hyper-heuristics. The tutorial will examine each of the four categories of hyper-heuristics, namely, selection constructive, selection perturbative, generation constructive and generation perturbative, showing how evolutionary algorithms can be used for each type of hyper-heuristic. A case study will be presented for each type of hyper-heuristic. The EvoHyp library will be used to demonstrate the implementation of evolutionary algorithm hyper-heuristics for the case studies. Challenges in the implementation of evolutionary algorithm hyper-heuristics will be highlighted. An emerging research direction is using hyper-heuristics for the automated design of computational intelligence techniques. The tutorial will look at the synergistic relationship between evolutionary algorithms and hyper-heuristics in this area. The tutorial will end with a discussion session on future directions in evolutionary algorithms and hyper-heuristics.

2.9 Evolutionary Bilevel Optimization (EBO): An Emerging Area for Research and Application in EC

Tutorial Speakers: Kalyanmoy Deb, *Michigan State University (USA)*, Ankur Sinha, *Indian Institute of Management Ahmedabad (India)*, and Pekka Malo, *Aalto University (Finland).*

Tutorial Abstract: Many practical optimization problems are better posed as hierarchical optimization problems in which different optimization tasks are put into different levels. The simplest of these hierarchical problems is known as "Bilevel" optimization problems which contain two levels of optimization tasks in a nested manner. A solution at the upper level is considered feasible only if it is optimal to the corresponding lower level problem. These problems are too complex to be solved using classical optimization methods simply due to the "nestedness" of one optimization task into another. Evolutionary algorithms provide amenable ways to address such problems due to their flexibility and ability to handle constrained search spaces efficiently. In this tutorial, we will introduce principles of bilevel optimization for both single and multiple objectives, and

discuss the difficulties in solving such problems in general. A number of applications of evolutionary bilevel optimization (EBO) will also be highlighted. A recent review on EBO is available in Sinha et al. (2018).

Sinha, A., Malo, P., and Deb, K. (2018). A Review on Bilevel Optimization: From Classical to Evolutionary Approaches and Applications. IEEE Transactions on Evolutionary Computation, Vol 22, No. 2, pp. 276–295.

2.10 Evolutionary Computation and Machine Learning in Cryptology

Tutorial Speaker: Stjepan Picek, *TU Delft (Netherlands).*

Tutorial Abstract: Evolutionary Computation (EC) has been successfully applied to various real-world problems. One domain rich with difficult problems is cryptology. This tutorial starts with a brief introduction on cryptology intended for general audience. Next, we examine several topics from cryptology that are successfully tackled up to now with EC and discuss why those topics are suitable to apply EC. We discuss the choice of appropriate EC techniques (GA, GP, CGP, ES, multi-objective optimization) for various problems and evaluate on the importance of that choice. We discuss the gap between the crypto community and EC community and what does it mean for the results. By doing that, we give a special emphasis on the perspective that cryptology can represent a source of interesting benchmark problems for EC. We finish with a more general overview on artificial intelligence applications in security. This tutorial will present live demos of EC in action when dealing with cryptology problems.

2.11 Exploratory Landscape Analysis

Tutorial Speakers: Pascal Kerschke and Mike Preuss, *University of Müenster (Germany).*

Tutorial Abstract: Exploratory Landscape Analysis (ELA) has been conceived as an automated approach for characterizing optimization problems by extracting – not necessarily intuitively understandable – landscape features, based on a rather small initial sample from the underlying optimization problem. Within this tutorial we will introduce the general concept of (automated) algorithm selection, which is one of the main use cases of ELA, followed by a presentation of examples from different optimization domains, in which ELA has successfully been used to improve algorithm selection processes. After presenting the general idea of ELA and providing a detailed overview of its status quo (including recently published extensions), we will show how ELA can improve our understanding of (a) the characteristics of different problem landscapes, and (b) the behavior of optimization algorithms, which are executed on these problems.

The remainder of the tutorial will be used for an interactive live-demo, in which our participants will perform ELA on some continuous optimization problems.

2.12 Genetic Improvement: Taking Real-World Source Code and Improving It Using Genetic Programming

Tutorial Speakers: John Woodward, *Queen Mary University of London (UK)* and Saemundur O. Haraldsson, *University of Stirling (UK).*

Tutorial Abstract: Genetic Programming (GP) has been around for 25 years. Genetic Improvement (GI) is new. GI evolves source code, rather than a simulation of code. In other words, GI operates directly on Java or C, for example, whereas GP operates on a tiny instruction set defined by the function set and terminal set. Another fundamental difference is that GI starts with real-world software, whereas GP typically tries to evolve programs from scratch. These differences may not seem important; however this subtle difference opens new possibilities for research. Furthermore we can optimize the physical properties of code such as power consumption, size of code, bandwidth, and other non-functional properties, including execution time.

This tutorial is of interest to people with a GP background interested in applying their techniques to real source code, and software practitioners interested in using automated techniques to improve software. We will not assume prior knowledge of GP.

2.13 Introduction to Statistical Modeling of EC Systems and Experiments: A Visual Approach

Tutorial Speaker: Mark Wineberg, *University of Guelph (Canada).*

Tutorial Abstract: This tutorial is a follow-up to the statistics tutorial given at GECCO. While still at an introductory level (attendance of the GECCO tutorial is not assumed), the material covered is typically found in more upper year undergraduate courses on statistical modeling. We will move beyond simple comparisons using T tests, to the understanding and modeling of the underlying behaviors of various factors that may affect a system, even when those behaviors are obscured by the noise encountered in a stochastic system. Topics include: factor models, two factor linear regression confidence bands, multiple regression, polynomial regression, the relationship to GP-style symbolic regression, post-hoc analysis, and possibly one-way and multi-way ANOVA, time permitting. As with the GECCO tutorial, this tutorial takes a very visual approach to statistics; relying on graphics and animation to provide an intuitive understanding of the subject, instead of the traditional equations, which cater to only the initiated.

2.14 Learning Classifier Systems as Learning Cognitive Systems

Tutorial Speaker: Will Browne, *Victoria University of Wellington (New Zealand).*

Tutorial Abstract: Learning classifier systems (LCSs) are an often overlooked class of rule-based machine learning algorithms with a unique and flexible set of features that sets them apart from other strategies. The original LCS from 40 years ago, CS-1, stood for Cognitive System - One. Subsequently, LCSs have become powerful evolutionary machine learning techniques, but there is still much to be gained by exploring their cognitive systems capabilities. The tutorial will begin with a gentle introduction to LCSs based on the recent textbook 'Introduction to Learning Classifier Systems', which is co-authored by the presenter. The second part of this tutorial will show examples of Learning Classifier Systems in terms of cognitive systems. In-depth understanding will be provided regarding improved perception, representation, transfer learning and embodied LCSs. Examples will be discussed of solving previously intractable problems in a human-like manner as well as unique applications in robotics.

2.15 Mathematical Programming as a Complement to Bio-inspired Optimization

Tutorial Speaker: Ofer Shir, *Tel-Hai College and Migal-Galilee Research Institute (Israel).*

Tutorial Abstract: Global optimization of complex models has been for several decades approached by means of formal algorithms as well as Mathematical Programming (MP), and simultaneously has been treated by a wide range of dedicated heuristics - where nature-inspired approaches are placed. These two branches complement each other, yet practically studied under two independent CS disciplines. The claim that education within the scope of problem-solving from nature should encompass basic MP is untenable at present times, and this tutorial aims at bridging the gap for our scholars and students. The tutorial comprises two parts. The first part presents the fundamentals of MP. It overviews mathematical optimization in light of convex optimization versus combinatorial optimization. It discusses some of the theoretical aspects, such as polyhedra and the duality theorem. The second part focuses on MP in practice, particularly on modeling, and covers selected algorithms: Simplex, Ellipsoid, and Branch-and-Bound. The tutorial is planned for all PPSN participants, assuming no prior knowledge in mathematical optimization.

2.16 Multiagent Systems and Agent-Based Modeling and Simulation

Tutorial Speaker: Ana Bazzan, *Federal University of Rio Grande do Sul (Brazil).*

Tutorial Abstract: Multiagent systems (MAS) and agent-based modeling and simulation (ABMS) deal with social interactions among intelligent actors (agents). These two disciplines study neither just physical systems nor agents in isolation, but the agent as part of a social space.The goal of this tutorial is twofold: (a) about half of the time will cover basic material about MAS and ABMS (since this simulation paradigm is highly used by the computational intelligence community), and hence provide the audience a sense of the basic principles; and (b) about half of the time will cover the most recent advances in MAS, including the highly relevant topic of multiagent learning, one of the obvious interfaces between these two communities, as well evolutionary game theory.

2.17 Multi-objective Optimization with the jMetal Framework

Tutorial Speaker: Antonio J. Nebro, *University of Malaga (Spain).*

Tutorial Abstract: jMetal is a Java-based framework for multi-objective optimization with metaheuristics which has become popular in many disciplines (engineering, economics, bioinformatics, etc.). The journal paper describing jMetal has more than 775 citations according to Google Scholar, and it has been used by research groups, industry and academia. In this tutorial, we give a practical overview of the main jMetal components (algorithms, encodings, problems, operators, experiments, quality indicators), focusing on how to configure and run some of the included metaheuristics and also on how to incorporate new solution representations and problems. We give examples of classical algorithms but also more modern techniques, including preference-based metaheuristics. Special attention will be paid to the definition of experimental studies to statistically assess the performance of algorithms. The main goal is that the attendants can replicate all the examples presented, and the material needed to follow the tutorial will be available in a public repository (https://github.com/jMetal/PPSN2018Tutorial).

2.18 Next Generation Genetic Algorithms

Tutorial Speaker: Darrell Whitley, *Colorado State University (USA).*

Tutorial Abstract: New developments in Gray Box Optimization makes it possible to construct new forms of Genetic Algorithms that do not use random mutation or random recombination. Instead, for certain classes of NP Hard problems, it is possible to exactly compute the location of improving moves in constant time. In some domains, this makes random mutation obsolete. Deterministic "Partition Crossover" can be applied to optimization problems such as MAXSAT and the Traveling Salesman Problem. Partition Crossover locally decomposes a recombination graph into q subgraphs in $O(n)$ time. It can then identify the best of 2^q possible offspring. If the parents are local optima, the

offspring are guaranteed to be locally optimal in the largest hyperplane subspace containing both parents. Local decomposition has also been used to solve multiply constrained scheduling problems with unto 1 billion variables.

The book chapter "Next Generation Genetic Algorithms" will accompany the tutorial.

2.19 Runtime Analysis of Population-Based Evolutionary Algorithms

Tutorial Speaker: Per Kristian Lehre, *University of Birmingham (UK).*

Tutorial Abstract: Populations are at the heart of evolutionary algorithms (EAs). They provide the genetic variation which selection acts upon. A complete picture of EAs can only be obtained if we understand their population dynamics. A rich theory on runtime analysis of EAs has been developed over the last 20 years. This theory provides insights into how the performance of EAs depends on their parameter settings and the characteristics of the underlying fitness landscapes. Early studies were mostly concerned with EAs without populations, such as the $(1+1)$ EA. This tutorial introduces recent techniques that enable runtime analysis of EAs with realistic populations. To illustrate the application of these techniques, we consider fundamental questions such as: When are populations necessary for efficient optimisation? What is the appropriate balance between exploration and exploitation and how does this depend on relationships between mutation and selection rates? What determines an EA's tolerance for uncertainty?

2.20 Semantic Genetic Programming

Tutorial Speakers: Alberto Moraglio, *University of Exeter (UK)* and Krzysztof Krawiec, *Poznan University of Technology (Poland).*

Tutorial Abstract: Semantic genetic programming is a recent, rapidly growing trend in Genetic Programming (GP) that aims at opening the 'black box' of the evaluation function and make explicit use of more information on program behavior in the search. In the most common scenario of evaluating a GP program on a set of input-output examples (fitness cases), the semantic approach characterizes program with a vector of outputs rather than a single scalar value (fitness). The past research on semantic GP has demonstrated that the additional information obtained in this way facilitates designing more effective search operators. In particular, exploiting the geometric properties of the resulting semantic space leads to search operators with attractive properties, which have provably better theoretical characteristics than conventional GP operators. This in turn leads to dramatic improvements in experimental comparisons. The aim of the tutorial is to give a comprehensive overview of semantic methods in genetic programming,

illustrate in an accessible way a formal geometric framework for program semantics to design provably good mutation and crossover operators for traditional GP problem domains, and to analyze rigorously their performance (runtime analysis). A number of realworld applications of this framework will be also presented. Other promising emerging approaches to semantics in GP will be reviewed. In particular, the recent developments in the behavioral programming, which aims at characterizing the entire program behavior (and not only program outputs) will be covered as well. Current challenges and future trends in semantic GP will be identified and discussed.

Efficient implementation of semantic search operators may be challenging. We will illustrate very efficient, concise and elegant implementations of these operators, which are available for download from the web.

2.21 The Cartography of Computational Search Spaces

Tutorial Speaker: Gabriela Ochoa, *University of Stirling (UK).*

Tutorial Abstract: The performance of heuristic search algorithms crucially depends on the underlying fitness landscape structure. Most fitness landscapes analysis techniques study their local structure; there is a lack of tools to study instead their global structure, which is known to impact algorithms' performance. This tutorial will describe local optima networks (LONs), a model of fitness landscapes suited to analyse their global structure by bringing tools from complex networks. LONs provide new insight into the structural organisation and the connectivity pattern of a search space. We will cover the relevant definitions, extraction methodologies, metrics, and visualisation techniques to thoroughly characterise the global structure of computational search spaces. We will consider the landscapes induced by both evolutionary and local-search algorithms and will show results on combinatorial problems (binary and permutation spaces) as well as on computer program search spaces. An interactive demo will allow attendees to analyse and visualise realistic computational search spaces.

2.22 The Most Recent Advances on Multi-Modal Optimization

Tutorial Speakers: Michael G. Epitropakis, *University of Patras (Greece)*, Mike Preuss, *University of Müenster (Germany)*, and Xiaodong Li, *RMIT University (Australia).*

Tutorial Abstract: Multi-Modal optimization (MMO) is currently undergoing many changes, and becoming established as an active research area that collects approaches from various domains of operational research, swarm intelligence and evolutionary computation. Typically MMO strives for delivering multiple optimal (or close to optimal) solutions in a single optimization run. This tutorial will cover several scenarios and list currently employed and potentially available

performance measures. Furthermore, many state-of-the-art as well as more classic MMO methods are compared and put into a rough taxonomy. We will also discuss recent relevant competitions and their results and outline the possible future developments in this area. In brief, the tutorial will cover the following topics (in syllabus form):

- Multi-modal optimization - what for?
- Niching? Biological inspiration and optimization reality
- Towards theory: high-level modelling
- Suggested taxonomy and critical review of methods (with Demos)
- Spotlight: Clustering, multiobjectivization, surrogates and archives
- 'Measuring and different scenarios'
- Developing challenging competition benchmark function sets
- Discussion on current competition results, and available software
- Expected future developments.

2.23 Theory of Parallel Evolutionary Algorithms

Tutorial Speaker: Dirk Sudholt, *University of Sheffield (UK).*

Tutorial Abstract: Evolutionary algorithms (EAs) have given rise to many parallel variants, fuelled by the rapidly increasing number of CPU cores and the ready availability of computation power through GPUs and cloud computing. A very popular approach is to parallelize evolution in island models, or coarse-grained EAs, by evolving different populations on different processors. These populations run independently most of the time, but they periodically communicate genetic information to coordinate search. Many applications have shown that island models can speed up computation time significantly, and that parallel populations can further increase solution diversity. However, there is little understanding of when and why island models perform well, and what impact fundamental parameters have on performance. This tutorial will give an overview of recent theoretical results on the runtime of parallel evolutionary algorithms. These results give insight into the fundamental working principles of parallel EAs, assess the impact of parameters and design choices on performance, and contribute to the design of more effective parallel EAs.

Workshops at PPSN 2018

Robin Purshouse[1], Christine Zarges[2(✉)], Sylvain Cussat-Blanc[3], Michael G. Epitropakis[4], Marcus Gallagher[5], Thomas Jansen[2], Pascal Kerschke[6], Xiaodong Li[7], Fernando G. Lobo[8], Julian Miller[9], Pietro S. Oliveto[1], Mike Preuss[6], Giovanni Squillero[10], Alberto Tonda[11], Markus Wagner[12], Thomas Weise[13], Dennis Wilson[3], Borys Wróbel[14], and Aleš Zamuda[15]

[1] University of Sheffield, Sheffield, UK
[2] Aberystwyth University, Aberystwyth, UK
c.zarges@aber.ac.uk
[3] University of Toulouse, Toulouse, France
[4] Lancaster University, Lancaster, UK
[5] University of Queensland, Brisbane, Australia
[6] University of Münster, Münster, Germany
[7] RMIT University, Melbourne, Australia
[8] University of Algarve, Faro, Portugal
[9] University of York, York, UK
[10] Politecnico di Torino, Torino, Italy
[11] National Institute of Agronomic Research, Thiverval-Grignon, France
[12] University of Adelaide, Adelaide, Australia
[13] Hefei University, Hefei, China
[14] Adam Mickiewicz University, Poznań, Poland
[15] University of Maribor, Maribor, Slovenia

Abstract. This article provides an overview of the 6 workshops held in conjunction with PPSN 2018 in Coimbra, Portugal. For each workshop, we list title, organizers, aim and scope as well as the accepted contributions.

1 Welcome from the Workshop Chairs

Workshops are an integral part of the conference series on Parallel Problem Solving From Nature (PPSN). They are intended as forums for presenting and discussing emerging approaches or critical reflections within a subfield. They provide an excellent opportunity to meet people with similar interests, to be exposed to cutting-edge research, and to exchange ideas in an informal setting.

About a year before the main conference, the organizing committee invited proposals for workshops to be held in conjunction with PPSN 2018. The organizers of an accepted workshop are responsible for its format, coordination, publicity, and technical program. Interactive sessions are encouraged. Most workshop contributions are short position papers or abstracts rather than full papers.

For PPSN 2018, the workshop chairs considered 8 high-quality workshop proposals and selected 6 of them for inclusion in the conference program based

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 490–497, 2018.
https://doi.org/10.1007/978-3-319-99259-4_39

on scheduling constraints and synergy with other workshops and tutorials. All workshops were half-day workshops and were held during the first two days of the conference, carefully scheduled alongside the accepted tutorial program.

Benchmarking represented a key theme for many workshop sessions, with topics including on-going advances in benchmarks and methods for multimodal problems, choice of functions to include in standardized benchmarking studies, and use of machine learning problems for benchmarking. A further workshop sought to consider how practice-based considerations can influence the development and benchmarking of optimization algorithms. Hybridization of machine learning with evolutionary computation was also a theme, alongside a considered look at how to incorporate developmental processes observed in nature into artificial neural networks. Two of the workshops considered aspects of bridging the gap between theory and practice in nature-inspired optimization and were associated with COST Action CA15140 'Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice' (ImAppNIO)[1,2].

Summaries for all workshops are presented in the next section. The total number of 31 contributions, as well as overview presentations and panel discussions, demonstrate the sustained popularity of the workshop format and represent a significant contribution to the overall PPSN conference program. Workshop organizers were based at 15 different institutions from 9 countries in Asia, Australia, and Europe and accepted presentations from a diverse set of authors from 20 countries in America, Asia, Australia, and Europe.

We hope that attendees have enjoyed interesting, thought-provoking and fruitful discussions in the workshops at PPSN 2018!

Robin Purshouse and Christine Zarges
PPSN 2018 Workshop Chairs

2 The Six Workshops

We summarize the content of the workshops at PPSN 2018 as follows:

1. Advances in Multimodal Optimization (Sect. 2.1)
2. Black-Box Discrete Optimization Benchmarking (BB-DOB) (Sect. 2.2)
3. Bridging the Gap Between Theory and Practice in Nature-Inspired Optimization (Sect. 2.3)
4. Developmental Neural Networks (Sect. 2.4)
5. Evolutionary Machine Learning (Sect. 2.5)
6. Investigating Optimization Problems from Machine Learning and Data Analysis (Sect. 2.6)

[1] http://imappnio.dcs.aber.ac.uk
[2] http://www.cost.eu/COST_Actions/ca/CA15140

2.1 Advances in Multimodal Optimization

Organizers:

- Mike Preuss, University of Münster, Münster, Germany
- Michael G. Epitropakis, Lancaster University, Lancaster, United Kingdom
- Xiaodong Li, RMIT University, Melbourne, Australia

URL: http://www.epitropakis.co.uk/ppsn2018-niching/

Aim and Scope: The workshop attempts to bring together researchers from evolutionary computation and related areas who are interested in Multi-modal Optimization. This is a currently forming field, and we aim for a highly interactive and productive meeting that makes a step forward towards defining it. The Workshop provides a unique opportunity to review the advances in the current state-of-the-art in the field of Niching methods. Further discussion will deal with several experimental/theoretical scenarios, performance measures, real-world and benchmark problem sets and outline the possible future developments in this area.

List of Accepted Contributions

Authors	Title
M. Epitropakis, X. Li, M. Preuss	Current State of Multimodal Optimization
H. Ishibuchi	Multi-Modal Multi-Objective Optimization: Test Problems, Algorithms and Performance Indicators
P. Kerschke	Exploiting a Problem's Multimodality for Improved Multi-Objective Optimization

2.2 Black-Box Discrete Optimization Benchmarking (BB-DOB)

Organizers:

- Pietro S. Oliveto, University of Sheffield, Sheffield, United Kingdom
- Markus Wagner, University of Adelaide, Adelaide, Australia
- Thomas Weise, Hefei University, Hefei, China
- Borys Wróbel, Adam Mickiewicz University, Poznań, Poland
- Aleŝ Zamuda, University of Maribor, Maribor, Slovenia

URL: http://iao.hfuu.edu.cn/bbdob-ppsn18

Aim and Scope: The aim of BB-DOB is to set up a process that will allow to achieve a standard methodology for the benchmarking of black-box optimization algorithms in discrete and combinatorial search spaces. Our long-term aim is to produce:

1. a well-motivated benchmark function testbed
2. an experimental set-up
3. generation of data output for post-processing and
4. presentation of the results in graphs and tables

In this workshop we encourage a discussion concerning which functions should be included in the benchmarking testbed (i.e., point (1) above). The functions should capture the difficulties of combinatorial optimization problems in practice but at the same time be comprehensible such that algorithm behaviors can be interpreted according to the performance on a given benchmark problem. The desired search behavior should be clear and algorithm deficiencies understood in depth. This understanding should lead to the design of improved algorithms. Ideally (not necessarily for all), the benchmark functions should be scalable with the problem size and non-trivial in the black-box optimization sense (the function may be shifted such that the global optimum may be any point). This workshop is organized in connection with and partly based upon work from COST Action CA15140 'Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice', supported by COST (European Cooperation in Science and Technology, see footnotes 1 and 2).

List of Accepted Contributions

Authors	**Title**
V. Jacimovic	Consensus on Non-Euclidean Manifolds Over Complex Networks: Optimization Problems and Benchmark Functions
Q. Yang, J. Zou, G. Ruan, S. Yang, and J. Zheng	A Dynamic Preference-Based Evolutionary Multi-Objective Optimization Benchmark Based on Reference Point?
S. Wasik, M. Antczak, J. Badura, and A. Laskowski	Optil.io: Online Platform for Benchmarking Optimization Algorithms
A. Zamuda, G. Hrovat, E. Lloret, M. Nicolau, and C. Zarges	Examples Implementing Black-Box Discrete Optimization Benchmarking Survey for BB-DOB@GECCO and BB-DOB@PPSN
S. Raggl	Discrete Real-world Problems in a Black-Box Optimization Benchmark
P. Kerschke, J. Bossek, and H. Trautmann	Analyzing the Impact of Performance Indicator Parameterizations on the Assessment of Algorithm Performances
O. M. Shir, C. Doerr, and T. Bäck	Compiling a Benchmarking Test-Suite for Combinatorial Black-Box Optimization: A Position Statement
H. Wang, F. Ye, C. Doerr, S. van Rijn, and T. Bäck	IOHProfiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics

2.3 Bridging the Gap Between Theory and Practice in Nature-Inspired Optimization

Organizers:

- Fernando G. Lobo, University of Algarve, Faro, Portugal
- Thomas Jansen, Aberystwyth University, Aberystwyth, United Kingdom

URL: http://fernandolobo.info/ppsn2018workshop/

Aim and Scope: Nature-inspired search and optimization heuristics have been used for decades to solve practical problems across different domains. Alongside, the theoretical understanding of them has been improving substantially, providing better understanding of what they can and cannot do in terms of solution quality and runtime. In spite of much improvement from the theoretical perspective, there is a large gap between theoretical foundations and practical applications. Theory and practice reinforce each other. Theory is driven by the need to improve understanding of challenges observed in practice. Likewise, practical applications can benefit from insights and guidelines derived from theory.

The workshop seeks to bring together researchers interested in the debate on how to narrow the gap between theory and practice. It is organized in connection with and partly based upon work from COST Action CA15140 'Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice', supported by COST (European Cooperation in Science and Technology, see footnotes 1 and 2). We hope that this debate will improve the current state of the field.

List of Accepted Contributions

Authors	Title
C. M. Fonseca	It's All About the Problem: Some Thoughts on Nature-Inspired Solver Software Development
E. Tuba, C. M. Fonseca, and P. Machado	Towards a Combinatorial Optimization API for Nature-Inspired Optimization Algorithms
C. Doerr	Towards a More Practice-Aware Evaluation of Iterative Optimization Heuristics
F. Lobo	Interplay Between Theory and Practice

2.4 Developmental Neural Networks

Organizers:

- Dennis Wilson, University of Toulouse, Toulouse, France
- Julian Miller, University of York, York, United Kingdom

– Sylvain Cussat-Blanc, University of Toulouse, Toulouse, France

URL: https://www.irit.fr/devonn/

Aim and Scope: In nature, brains are built through a process of biological development in which many aspects of the network of neurons and connections change are shaped by external information received through sensory organs. Biological development mechanisms such as axon guidance and dendrite pruning have been shown to rely on neural activity. Despite this, most artificial neural network (ANN) models do not include developmental mechanisms and regard learning as the adjustment of connection weights, while some that do use development restrain it to a period before the ANN is used. It is worthwhile to understand the cognitive functions offered by development and to investigate the fundamental questions raised by artificial neural development. In this workshop, we will explore existing and future approaches that aim to incorporate development into ANNs. Invited speakers will present their work with neural networks, both artificial and biological, in the context of development. Accepted submissions on contemporary work in this field will be presented and we will hold an open discussion on the topic.

List of Accepted Contributions

Authors	Title
S. Pautot	Keynote Presentation on Neurosciences
J. F. Miller, D. G. Wilson, and S. Cussat-Blanc	Evolving Programs That Build Neural Networks for Multiple Problems
J. Stork, M. Zaefferer, and T. Bartz-Beielstein	Distance-Based Kernels for Surrogate Model-Based Neuroevolution
Y. Ying, A. Rose, A. Siddique, and W. N. Browne	Minimum Requirements for an Artificial Rat
D. G. Wilson, S. Cussat-Blanc, and H. Luga	A Gene Regulatory Network Model for Axon Guidance

2.5 Evolutionary Machine Learning

Organizers:

– Giovanni Squillero, Politecnico di Torino, Torino, Italy
– Alberto Tonda, National Institute of Agronomic Research, Thiverval-Grignon, France

URL: https://evolearning.github.io/ppsn18/

Aim and Scope: Evolutionary machine-learning (EML) could be easily defined as a crossbreed between the fields of evolutionary computation (EC) and machine

learning (ML). However, as the 'obvious connection' between the processes of learning and evolution has been pointed out by Turing back in 1950, to avoid a blatant pleonasm, the term is mostly used referring to the integration of well-established EC techniques and canonical ML frameworks. A first line of research ascribable to EML predates the recent ML windfall and focuses on using EC algorithms to optimize frameworks: it included remarkable studies in the 1990s, such as the attempts to determine optimal topologies for an artificial neural network using a genetic algorithm. The other way around, a line tackling the use of ML techniques to boost EC algorithms appeared before 2000. More recently, scholars are proposing truly hybrid approaches, where EC algorithms are deeply embedded in frameworks performing ML tasks.

List of Accepted Contributions

Authors	**Title**
I. Bonnici, A. Gouaïch, and F. Michel	Eco-Evolutionary Search in a Metamorph Learner
B. Doerr, C. Doerr, and J. Yang	Provably Efficient Search Heuristics by Learning-Inspired Parameter Control
S. Al-Maliki, E. Lutton, F. Boué, and F. Vidal	MRI Gastric Images Processing Using a Multiobjective Fly Algorithm
E. Medvet, A. Bartoli, A. Ansuini, and F. Tarlao	Observing the Population Dynamics in GE by Means of the Intrinsic Dimension
Y. Nojima, S. Sakai, N. Masuyama, and H. Ishibuchi	Multiobjective Evolutionary Classifier Design Using Class Scores by a Deep Convolutional Neural Network
C. Pageau, A. Blot, H. H. Hoos, M.-E. Kessaci, and L. Jourdan	Automatic Design of a Dynamic Multi-Objective Local Search Algorithm
R. Denysiuk, R. Pinto, M. F. Costa, L. Costa, and A. Gaspar-Cunha	Feature Selection Using Multiobjective Evolutionary Algorithms
D. G. Wilson, K. Harrington, S. Cussat-Blanc, and H. Luga	Evolving Differentiable Gene Regulatory Networks

2.6 Investigating Optimization Problems from Machine Learning and Data Analysis

Organizers:

- Marcus Gallagher, University of Queensland, Brisbane, Australia
- Mike Preuss, University of Münster, Münster, Germany
- Pascal Kerschke, University of Münster, Münster, Germany

URL: https://sites.google.com/view/optml-ppsn18/home

Aim and Scope: In continuous black-box optimization, there are a number of benchmark problem sets and competitions. However, the focus has mainly been on the performance and comparison of algorithms on artificial problems. The aim of this workshop is to instead make a set of optimization problems the centre of focus, bringing together researchers to discuss and develop deeper insights into the structure and difficulty of the problem set, as well as experimental methodology (including algorithms). Several problem classes (and specific problem instances) from the area of machine learning and data analysis were proposed in advance of the workshop submission deadline. Submission of brief papers that show new insights into the problems, for example via exploratory landscape analysis, algorithm performance (with a focus on 'why') or analysis of the quality/diversity of solutions present in the problem instances are invited.

List of Accepted Contributions

Authors	**Title**
M. Gallagher and S. Saleem	Exploratory Landscape Analysis of the MLDA Problem Set
S. H. Zhan and M. A. Muñoz	Recurrence Quantification Analysis of the State Space Trajectories of Black-Box Continuous Optimization Algorithms
M. Gallagher, S. Saleem, S. Van Ryt, and Y. Qiao	Evaluating Algorithm Performance on the MLDA Problem Set

Author Index