BIOINSPIRED OPTIMIZATION METHODS AND THEIR APPLICATIONS

BIOINSPIRED OPTIMIZATION METHODS AND THEIR APPLICATIONS

Proceedings of the Seventh International Conference on Bioinspired Optimization Methods and their Applications, BIOMA 2016

18–20 May 2016, Bled, Slovenia

Edited by

GREGOR PAPA
Jožef Stefan Institute, Ljubljana

MARJAN MERNIK
University of Maribor

Jožef Stefan Institute
Ljubljana, Slovenia

Editors: Gregor Papa, Marjan Mernik

Cover design by Studio Design Demšar, Škofja Loka
Logo design by Gregor Papa
Printed by Tiskarna Artelj, Ljubljana
Published by Jožef Stefan Institute, Ljubljana, Slovenia

Typesetting by Jurij Šilc in `kapproc`-based LaTeX style with kind permission from Kluwer Academic Publishers

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004.02(082)
005.519.1(082)
510.5(082)

INTERNATIONAL Conference on Bioinspired Optimization Methods and their Applications (7 ; 2016 ; Bled)
Bioinspired optimization methods and their applications : proceedings of the Seventh International Conference on Bioinspired Optimization Methods and their Applications - BIOMA 2016, 18-20 May 2016, Bled, Slovenia / edited by Gregor Papa, Marjan Mernik. - Ljubljana : Jožef Stefan Institute, 2016

ISBN 978-961-264-093-4
1. Gl. stv. nasl. 2. Papa, Gregor

284600576

Contents

Preface

Bioinspired optimization methods is an umbrella term for stochastic optimization methods that are based on principles or models of biological systems. This class of methods, such as evolutionary algorithms and swarm intelligence algorithms, are nowadays indispensable for solving complex optimization problems in science, engineering and business.

This volume contains recent theoretical and practical contributions to the field of bioinspired optimization presented at the Seventh International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2016), held in Bled, 18–20 May 2016. The purpose of biennial BIOMA conferences is to provide a forum for presentation and discussion of the latest theoretical and applied results in bioinspired optimization methods and their applications. It is organized since 2004 by Jožef Stefan Institute. During these years BIOMA became a respectful conference with well known lively discussion among researchers and practitioners working in the field of bioinspired computation.

This year we received 25 papers and each paper received three reviews. The reviews were performed by 30 members of the international program committee. In the reviewing procedure, 17 papers were selected for presentation at the conference, which is 68% of the submissions. Together with two invited contributions the conference proceedings contain 19 papers by 45 (co)authors from 11 countries.

The first BIOMA 2016 keynote talk, "A Survey of Model-Based Methods for Global Optimization" given by prof. dr. Thomas Bartz-Beielstein from TH Köln – Technology, Arts, Sciences, Germany, focuses on fundamental aspects of surrogate-model based optimization and recent advances in this field. The second keynote talk, "Parallel Multi-Objective Evolutionary Algorithms" given by prof. dr. El-Ghazali Talbi from University Lille 1, France, provides a unified taxonomy of parallel multi-objective evolutionary algorithms.

Theoretical contributions presented at the conference include sensitivity analysis, parameter control, statistical comparison analysis, impact of quality indicators of multi-objective evolutionary algorithms, ensem-

bles of surrogates, adaptive approaches, worst case estimation based on prediction intervals, and other algorithms' improvements. Reports on applications come from domains such as network community structure detection, dynamic vehicle routing, optimization of a manufacturing quality-control procedure, optimization of water distribution networks, and optimization of a robust gas sensor.

The BIOMA 2016 conference is supported by the Slovenian Research Agency and its research programs. Technical sponsors of the conference are the World Federation of Soft Computing, the Slovenian Artificial Intelligence Society, and the Jožef Stefan Institute. This conference is part of a project that has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 692286.

In conclusion, we hope that all the aforementioned papers will provide readers with some glimpse of research presented at BIOMA 2016. We are grateful to the conference sponsors, members of the program and organizing committees, the keynote speakers, paper presenters and other participants for contributing their parts to the conference. We wish you an inspiring meeting and a pleasant stay in Bled.

Ljubljana, 5 May 2016

Gregor Papa and Marjan Mernik

Program Committee

Gregor Papa, *Chair*, Jožef Stefan Institute, Ljubljana, Slovenia
Marjan Mernik, *Chair*, University of Maribor, Slovenia

Thomas Bartz-Beielstein, TH Köln – Technology, Arts, Sciences, Germany
Christian Blum, University of the Basque Country, San Sebastian, Spain
Janez Brest, University of Maribor, Slovenia
Dirk Büche, MAN Diesel & Turbo Schweiz AG, Zürich, Switzerland
Carlos A. Coello Coello, CINVESTAV-IPN, Mexico
Carlos Cotta, Universidad de Malaga, Spain
Rolf Drechsler, University of Bremen/DFKI, Germany
Bogdan Filipič, Jožef Stefan Institute, Ljubljana, Slovenia
Peter Korošec, Jožef Stefan Institute, Ljubljana, Slovenia
Barbara Koroušić Seljak, Jožef Stefan Institute, Ljubljana, Slovenia
Shih-Hsi "Alex" Liu, California State University, Fresno, USA
Goran Martinović, Josip Juraj Strossmayer University of Osijek, Croatia
JJ Merelo, University of Granada, Spain
Edmondo Minisci, University of Strathclyde, Glasgow, UK
Nalini N, Nitte Meenakshi Institute of Technology, Bangalore, India
Nadia Nedjah, State university of Rio de Janeiro, Brasil
Frank Neumann, The University of Adelaide, Australia
Borut Robič, University of Ljubljana, Slovenia
Franciszek Seredynski, Cardinal Stefana Wyszynski University, Warsaw, Poland
Jurij Šilc, Jožef Stefan Institute, Ljubljana, Slovenia
El-Ghazali Talbi, University Lille 1, France
Jim Tørresen, University of Oslo, Norway
Tea Tušar, Jožef Stefan Institute, Ljubljana, Slovenia

Organizing Committee

Contributing Authors

Hernán Aguirre received his Ph.D. degree in Systems Development Engineering from Shinshu University, Japan, in 2003. He is currently an associate professor at Shinshu University, Faculty of Engineering. His research interests include computational intelligence, multi- and many-objective evolutionary optimization, design innovation, and sustainability. He is a member of IEEE, IEICE, and IPSJ.

Dražen Bajer received his M.Sc. degree in Computer Engineering from the Faculty of Electrical Engineering, Josip Juraj Strossmayer University of Osijek, Croatia, in 2010. He is currently a teaching and research assistant at the Faculty of Electrical Engineering, Josip Juraj Strossmayer University of Osijek, where he is also pursuing the Ph.D. degree. His research interests are computational intelligence methods and their applications, and unsupervised classification. He is an IEEE graduate student member.

Thomas Bartz-Beielstein received his Ph.D. degree in Computer Science from the Technical University of Dortmund, Germany, in 2005. He is currently a professor at the Technische Hochschule Köln, Faculty of Computer Science and Engineering Science. His research interests include simulation, optimization, and statistical analysis of complex real-world problems. He is a member of several associations in the field of evolutionary computing, e.g., ACM SIG "Genetic and Evolutionary Computation", VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik (Fachausschuss Computational Intelligence), EU/ME working group on Metaheuristics, and the International Society on Multiple Criteria Decision Making.

Petr Bujok received his Ph.D. degree in Parallel Evolutionary Algorithms from the University of Ostrava, Czech Republic, in 2013. He is

currently an assistant professor at the University of Ostrava, Faculty of Science. His research interests include evolutionary algorithms, differential evolution, parallel models, global optimization problems and computational statistics. He is a member of Czech Statistical Society.

Matej Črepinšek received his Ph.D. degree in Computer Science from the University of Maribor, Slovenia, in 2005. He is currently a teaching assistant at the University of Maribor, Faculty of Electrical Engineering and Computer Science. His research interests include grammar-based systems, grammatical inference, programming languages, compilers and evolutionary computations.

Tatjana Davidović received her Ph.D. from the Mathematical Department at the University of Belgrade, Serbia, in 2006. She is a research associate professor at the Mathematical Institute of the Serbian Academy of Sciences and Arts. She is also an associate professor for the doctoral courses on Parallel Programming, Meta-heuristics, and Optimization at the Faculty of Technical Sciences, University of Novi Sad, Serbia. Her main research interests include parallel computing, scheduling, combinatorial optimization, mathematical programming, meta-heuristics. She is a member of the Editorial Board for International Journal for Traffic and Transport Engineering (IJTTE) and the reviewer for several international journals.

Tome Eftimov received his B.Sc.Eng. and M.Sc.Eng. degree in Electrical Engineering and Computer Science from the University of Ss. Cyril and Methodius, Skopje, Macedonia, in 2011 and 2013, respectively. He is currently working toward the Ph.D. degree at the Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia. His research interests include machine learning, data mining, text mining, sematic web, statistics, and knowledge extraction.

Michael T. M. Emmerich received his Doctorate degree in the Natural Sciences from the University of Dortmund, Germany, in 2005. He is currently associate professor at Leiden University, Faculty of Science, The Netherlands. He is also an associated professor at the University of Ijui, Brazil. His research interests include multicriteria decision analysis, multi-objective optimization, Gaussian processes, sustainable design, and computational drug discovery. He is a member of the International Society on Multicriteria Decision Making.

Oliver Flasch received his Ph.D. degree in Computer Science at the TU Dortmund University, Germany, in 2015. He is currently a postdoc at Cologne university of Applied Sciences. His research interests include scalable methods for the automatic discovery of mathematical models from data (symbolic regression via genetic programming). He is the CEO of the young startup sourcewerk.

Bogdan Filipič received his Ph.D. degree in Computer Science from the University of Ljubljana, Slovenia, in 1993. He is now a senior researcher and head of Computational Intelligence Group at the Department of Intelligent Systems of the Jožef Stefan Institute, Ljubljana, and an associate professor of Computer and Information Science at the University of Ljubljana. His research interests include stochastic optimization, evolutionary computation and intelligent data analysis. He published over 40 papers in international scientific journals and was a principle investigator in national and international projects dealing with production processes optimization, energy efficiency, and cultural heritage preservation. He serves as a member of editorial boards of several scientific journals and a programme committee member for the Genetic and Evolutionary Computation Conference (GECCO) and Congress on Evolutionary Computation (CEC). He was the general chair of the 13th International Conference on Parallel Problem Solving from Nature (PPSN 2014) and tutorial speaker at CEC 2014 and 2015.

Andreas Fischbach received his Diploma degree in Computer Science at the TU Dortmund University, Germany, in 2009. He is currently a Ph.D. student at the Cologne University of Applied Sciences. His research interest include design of experiments and modern optimization techniques such as sequential parameter optimization

Martina Friese received her Diploma degree in Computer Science from the University of Dortmund, Germany, in 2009. She is currently a doctorate student at the University of Leiden, Faculty of Science. She is also research assistant at the Cologne University of Applied Sciences. Her research interests include Global Optimization, Surrogate Models and Ensemble Methods.

Klemen Gantar received his B.Sc. degree in Computer and Information Science from the University of Ljubljana, Slovenia, in 2015. He is

now a masters student at the same University. His research interests are in evolutionary computation, data mining, and computer vision.

Noémi Gaskó received her Ph.D. degree in Computer Science from the Babeş-Bolyai University of Cluj Napoca, Romania, in 2011. She is currently a lecturer at the Babeş-Bolyai University, Faculty of Mathematics and Computer Science. Her research interests include computational game theory and evolutionary algorithms.

Niko Herakovič received his Ph.D. degree in at the RWTH Aachen, Germany, in 1995. He is currently a professor and a head of Chair of Manufacturing Technologies and Systems as well as a Head of the Laboratory for Handling, Assembly and Pneumatics at the Faculty of Mechanical Engineering, University of Ljubljana. In recent years he has been working on several basic and applied research and industrial projects related to production systems, computer vision, fluid power and mechatronics. His research interests include optimization of production system with the emphasis on handling and assembly systems as well as logistics, high dynamic and low energy consumption fluid power piezo valves, digital hydraulics, simulation, robot vision, robot applications etc. More than 8 years he has spent in industry and has headed or participated in over 50 R&D national and international projects in basic, applied/development and especially industrial research in Slovenia, Austria and Germany. He has over 260 scientific, professional and other publications, over 70 of them with a review.

Jakub Janoštík received his M.Sc. degree in Informatics from the Tomas Bata University in Zlin, Czech Republic, in 2014. He is currently a Ph.D. student at the Tomas Bata University in Zlin, Faculty of Applied Informatics. His research interests include evolutionary algorithms, data-mining, complex networks, big data.

Tatjana Jakšić Krüger is a research assistant at the Mathematical Institute of the Serbian Academy of Sciences and Arts and a Ph.D. student at the Faculty of Technical Sciences, University of Novi Sad, Serbia. Her research topics include the development, analysis, and parallelization of nature inspired meta-heuristics.

Peter Korošec received his Ph.D. degree from the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, in 2006. Since 2002

he has been a researcher at the Jožef Stefan Institute, Ljubljana. He is presently a researcher at the Computer Systems Department and an associate professor at the University of Primorska, Faculty of Mathematics, Natural Sciences and Informational Technologies, Koper. His current areas of research include combinatorial and numerical metaheuristic optimization and parallel/distributed computing.

Barbara Koroušić Seljak received her Ph.D. degree in Computer Science and Informatics from the University of Ljubljana, Slovenia, in 1997. She works as a senior researcher at the Computer Systems Department, Jožef Stefan Institute, and as an assistant professor at the Jožef Stefan International Postgraduate School. Her research work is related to real-time systems, heuristic optimization, and software modeling, in particular of e-health systems. Her work is published in international journals and conference proceedings. She has led and participated in several national and European projects. She is a member of the EuroFIR nonprofit association and the Slovenian Society for Clinical Nutrition and Metabolism.

Sebastian Krey received his Diploma degree in Statistics at the TU Dortmund University, Germany, in 2008. He is currently a Ph.D. student at the same university and a researcher at Cologne University of Applied Sciences. His research interest include supervised and unsupervised machine learning, design of experiments and computational statistics.

Rodica Ioana Lung received her Ph.D. degree in Computer Science from the Babeş-Bolyai University of Cluj Napoca, Romania, in 2006. She is currently an associate professor at the Department of Statistics, Forecasts, Mathematics in the same university. Her research interests include evolutionary optimization, complex networks, and game theory.

Jacek Mańdziuk received his Ph.D. degree in Applied Mathematics from the Warsaw University of Technology, Poland, in 1993 and D.Sc. degree in Computer Science from the Polish Academy of Sciences in 2000 and the title of Professor Titular in 2011. He is currently a professor at the Warsaw University of Technology, Faculty of Mathematics and Information Science. He has also been a visiting professor at the Nanyang Technological University, Singapore. His research interests include application of CI and AI to games, dynamic optimization, human-machine cooperation, financial modeling, and development of general-purpose

human-like learning and problem-solving methods. He is an Associate Editor of the IEEE Transactions on Computational Intelligence and AI in Games and an Editorial Board Member of the International Journal on Advances in Intelligent Systems.

Magdalene Marinaki received a Diploma in Production Engineering and Management from the Technical University of Crete, Greece and a Ph.D. from the same University. Currently she serves as a labotary educational staff in the Technical University of Crete, Chania, Greece. Her research interests focus on computational methods in optimization problems, on nature inspired methods, on supply chain management, on metaheuristics algorithms on data mining and on optimal control. She is the author of two books and more than forty articles in international scientific journals and books. She has more than fifty presentations in international and national scientific conferences. She has worked as a researcher in more than 10 European projects.

Yannis Marinakis received a Diploma in Production Engineering and Management from the Technical University of Crete, Greece, in 1999 and a Ph.D., from the same University, in 2005. He is currently an assistant professor in the Technical University of Crete, Chania, Greece. His research interests focus on computational methods in optimization problems, on nature inspired methods, on supply chain management, on operations research, on game theory and on metaheuristic algorithms. He teaches the following courses: Combinatorial Optimization, Game Theory, Design and Optimization in Supply Chain Management, Evolutionary Algorithms and Large Scale Optimization (postgraduate). He is the author of three books and more than one hundred papers in international scientific journals and books (Computers and Operations Research, European Journal of Operational Research, Journal of Global Optimization, Computational Optimization and Applications, Journal of Combinatorial Optimization, Expert Systems with Applications, Annals of Operations Research, Applied Soft Computing). He has more than forty presentations in international and national scientific conferences. He has participated in a number of research projects as principal investigator and as a researcher. He has been the supervisor of one Ph.D., 25 master theses and of 40 diploma theses.

Goran Martinović received his Ph.D. degree in Computer Science from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, in 2004. He is currently a professor at the Faculty of

Electrical Engineering, Josip Juraj Strossmayer University of Osijek. His research interests include distributed computer systems, fault tolerant systems, real-time systems, artificial intelligence, and medical informatics. He is a member of the IEEE, ACM, KOREMA, and IEEE SMC Technical Committee on Distributed Intelligent Systems.

Marjan Mernik received his Ph.D. degree in Computer Science from the University of Maribor, Slovenia, in 1998. He is currently a professor at the University of Maribor, Faculty of Electrical Engineering and Computer Science. He is also a visiting professor at the University of Alabama at Birmingham, USA. His research interests include evolutionary computations, domain-specific languages, and grammar-based systems. He is a member of ACM, IEEE, and EAPLS. He is the Editor-In-Chief of Computer Languages, Systems and Structures and the Associate Editor of Applied Soft Computing.

Taishi Ohno is a master student at Shinshu University, Japan. His research interests include multi-objective evolutionary optimization and design optimization. He is a member of IEICE.

Michał Okulewicz received his M.Sc. degree in Computer Science from Warsaw University of Technology, Poland, in 2011. He is currently a research assistant at the Warsaw University of Technology, Faculty of Mathematics and Information Science. His research interests include applications of artificial intelligence, swarm optimization and machine learning algorithms.

Michal Pluháček received his Ph.D. degree in Informatics from the Tomas Bata University in Zlin, Czech Republic, in 2016. He is currently a researcher at the Tomas Bata University in Zlin, Faculty of Applied Informatics. His research interests include swarm algorithms and intelligence, randomization in evolutionary computational techniques, intelligent systems.

Radka Polakova received her Ph.D. degree in Adaptation in Differential Evolution Algorithm from the University of Ostrava, Czech Republic in 2014. She is currently a junior researcher at the University of Ostrava, Institute of Research and Applications of Fuzzy Modeling. Her research interests include global optimization, evolutionary algorithms, differential evolution algorithm, and computational statistics.

Miha Ravber received his M.Sc. degree in Computer Science from the University of Maribor, Slovenia, in 2015. He is currently a junior researcher at the University of Maribor, Faculty of Electrical Engineering and Computer Science. His research interests include evolutionary computation, multiobjective optimization, bio-inspired computing.

Margarita Alejandra Rebolledo Coy received her Master degree in Automation and IT at the Cologne University of Applied Sciences, Germany, in 2013. She is currently a Ph.D. student at the same university. Her research interest include design of experiment, Bayesian data modeling and data analysis and optimization.

Manousos Rigakis received a Bachelor in Applied Mathematics from the University of Crete, Greece, in 2012. He is currently a M.Sc. candidate in the Technical University of Crete. His research interests focus on game theory, on computational methods in optimization problems, on nature inspired methods and on metaheuristic algorithms.

Jörg Stork received his Master degree in Automation and IT at the Cologne University of Applied Sciences, Germany, in 2013. He is currently a Ph.D. student at the same university. His research interest include surrogate modeling, metaheuristics and optimization.

Mihai Suciu received his Ph.D. degree in Computer Science from Babeş-Bolyai University, Romania, in 2013. He is currently a teaching assistant at Babeş-Bolyai University, Faculty of Mathematics and Computer Science. His research interests include computational evolutionary algorithms and game theory.

Roman Šenkeřík received his Ph.D. degree in Technical Cybernetics from the Tomas Bata University in Zlin, Czech Republic, in 2008. He is currently an associated professor at the Tomas Bata University in Zlin, Faculty of Applied Informatics. His research interests include interdisciplinary applications of evolutionary computation, modification and development of evolutionary and swarm based algorithms, theory of chaos, emergence and complexity.

Kiyoharu Tagawa received his Dr.Eng. degree from the Kobe University, Kobe, Japan, in 1993. He is currently a professor at the Kindai University, Higashi-Osaka, Japan, School of Science and Engineering. His

research interests include evolutionary algorithm, complex system optimization, multi-objective optimization, and their application for real-world problems. He is a member of IEEE.

El-Ghazali Talbi received the Master and Ph.D. degrees in Computer Science from the Institut National Polytechnique de Grenoble, France. He is a full professor at the University of Lille and the head of DOLPHIN research group from both the Lille's Computer Science laboratory (LIFL, Université Lille 1, CNRS) and INRIA Lille Nord Europe. His current research interests are in the field of multi-objective optimization, parallel algorithms, metaheuristics, combinatorial optimization, cluster and cloud computing, hybrid and cooperative optimization, and applications to logistics/transportation, bioinformatics and networks. He has to his credit more than 150 international publications including journal papers, book chapters and conferences proceedings.

Kiyoshi Tanaka received his Dr. Eng. degree from Keio University, Japan, in 1992. He is currently a professor at Shinshu University, Faculty of Engineering. His research interests include image and video processing, information hiding, evolutionary computation, chaos & fractals, and their applications. He is a member of IEEE, IEICE, IPSJ and IIEEJ.

Dimitra Trachanatzi received a Diploma in Production Engineering and Management from the Technical University of Crete, Greece. She is currently a M.Sc. candidate in the Technical University of Crete. Her research interests focus on game theory, on computational methods in optimization problems, supply chain management, on nature inspired methods and on metaheuristic algorithms.

Tea Tušar is a postdoc in the DOLPHIN team at INRIA Lille - Nord Europe, France. She received her Ph.D. degree in Information and Communication Technologies from the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, in 2014. Before joining INRIA in 2015, she has worked at the Department of Intelligent Systems at the Jožef Stefan Institute since 2004, first as a research assistant and later as a postdoc. Her research interests include evolutionary algorithms for single- and multi-objective optimization with emphasis on visualizing and benchmarking their results and applying them to real-world problems. She served as a workshops co-chair at PPSN 2014 and co-organized GECCOs Student Workshop in years 2013–2015.

Josef Tvrdík received his Ph.D. degree in Theory of Chemical Engineering from the Institute of Chemical Technology in Prague, Czech Republic, in 1979. He is currently an associated professor at the University of Ostrava, Faculty of Science. His research interests include stochastic algorithms for global optimization and their applications in computational statistics, biomedical applications of statistics and statistical software. He is a member of International Association for Statistical Computing, Czech Mathematical Society and Czech Statistical Society.

Adam Viktorin received his M.Sc. degree in Informatics from the Tomas Bata University in Zlin, Czech Republic, in 2015. He is currently a Ph.D. student at the Tomas Bata University in Zlin, Faculty of Applied Informatics. His research interests include evolutionary algorithms, data-mining, information retrieval.

Hugo Zupan obtained his M.Sc. in Mechanical Engineering from University of Ljubljana, Faculty of Mechanical Engineering, Slovenia in 2013. He is currently a researcher in Laboratory for Handling, Assembly and Pneumatics at Faculty of Mechanical Engineering, University of Ljubljana, Slovenia. He obtained his M.Sc. after the defense of his diploma thesis he has been employed in Laboratory for Handling, Assembly and Pneumatics. He has started his Ph.D. studies and is employed as young researcher in the year 2013. His main research and expertise field is logistic of resources in the production process, modeling and simulation. He is responsible for implementation simulation models in industrially based projects. The implemented simulation models solve practical logistical problems of companies in the field of production and assembly, including capacity of intermediate storages and flow of material in storages.

Janez Žerovnik received his Ph.D. degree in Computer Science from the University of Ljubljana, Slovenia, in 1992 and his Ph.D. degree in Mathematics from the Technical University Graz, Austria, in 1994. He is currently professor mathematics at University of Ljubljana and a part time researcher at the Institute of mathematics, physics and mechanics, Ljubljana, Slovenia. He was a research fellow at Montanuniversitaet Leoben, École Normale Supérieure Lyon, France, and Royal Holloway, University of London, U.K. He has published more than 100 papers in refereed journals and more than 100 papers in proceedings of scientific conferences. He is a member of editorial boards of Ars Mathematica Contemporanea, ISRN Discrete Mathematics, and Central European

Journal of Operations Research, and served as guest editor of special issues of Central European Journal of Operations Research (twice), Discussiones Mathematicae. Graph Theory, Theoretical Computer Science, and several conference proceedings including Structural information and communication complexity: revised selected papers, (Lecture Notes in Computer Science, vol. 5869). His main research interests are graph theory and optimization, and more general discrete mathematics with applications in theoretical computer science, chemical graph theory and operational research.

I

INVITED CONTRIBUTIONS

A SURVEY OF MODEL-BASED METHODS FOR GLOBAL OPTIMIZATION

Thomas Bartz-Beielstein
SPOTSeven Lab, TH Köln, Gummersbach, Germany.

spotseven.de
thomas.bartz-beielstein@th-koeln.de

Abstract This article describes model-based methods for global optimization. After introducing the global optimization framework, modeling approaches for stochastic algorithms are presented. We differentiate between models that use a distribution and models that use an explicit surrogate model. Fundamental aspects of and recent advances in surrogate-model based optimization are discussed. Strategies for selecting and evaluating surrogates are presented. The article concludes with a description of key features of two state-of-the-art surrogate model based algorithms, namely the evolvability learning of surrogates (EvoLS) algorithm and the sequential parameter optimization (SPO).

Keywords: Global optimization, Surrogate model.

1. Introduction

Model-based optimization (MBO) plays a prominent role in todays modeling, simulation, and optimization processes. It can be considered as the most efficient technique for expensive and time-demanding real-world optimization problems. Especially in the engineering domain, MBO is an important practice. Recent advances in computer science, statistics, and engineering in combination with progress in high-performance computing provide tools for handling problems, which were considered unsolvable only a few decades ago. This article presents a survey of MBO for global optimization.

Global optimization (GO) can be categorized based on different criteria. For example, the properties of problems to be solved (continuous versus combinatorial, linear versus nonlinear, convex versus multimodal, etc.) can be used. This article presents an algorithmic view on global optimization, i.e., properties of algorithms that search for new solutions are considered.

The term GO will be used in this article for algorithms that are trying to find and explore global optimal solutions with complex, multimodal objective functions [50]. Global optimization problems are difficult to solve, because nearly no structural information (e.g., number of local extrema) is available. Global optimization problems belong to the class of *black-box functions*, i.e., functions for which the analytic form is unknown. Note, the class of black-box functions contains also functions that are easy to solve, e.g., convex functions, which are not discussed in the following. This article focuses on difficult black-box functions.

Consider the *optimization problem* given by

$$\text{Minimize: } f(\mathbf{x}) \quad \text{subject to } \mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is referred to as the *objective function* and $\mathbf{x}_l$ and $\mathbf{x}_u$ denote the lower and upper bounds of the search space (region of interest), respectively. This setting arises in many real-world systems when the explicit form of the objective function f is not readily available, e.g., if the user has no access to the source code of a simulator.

This survey covers *stochastic (random) search algorithms*, deterministic GO algorithms are not further discussed. Random and stochastic search will be used synonymously in the remainder of this article.

An iterative search algorithm that uses a stochastic procedure to generate the next iterate is referred to as a *stochastic search algorithm*. The next iterate can be a candidate solution to the GO or a probabilistic model, where solutions can be drawn from. Stochastic search algorithms are considered robust and easy to implement, because they do not depend on any structural information of the objective function such as gradient information or convexity. This feature is one of the main reasons for the popularity of stochastic search in the domain of GO. Stochastic search algorithms can further be categorized as *instance-based* or *model-based* algorithms [71]. Furthermore, there are basically two model-based approaches: (a) *distribution-based* models and (b) *surrogate* models. We consider four important representatives of surrogate model based optimization: (i) *Multi-fidelity metamodeling* uses several models of the same real system and plays an important role in CFD/FEM based simulation and optimization. (ii) *Evolutionary surrogate based optimization* extends the traditional EA framework, and (iii) *Ensemble surrogate based optimization* combines two or more different surrogate models.

So far, we have obtained the GO categorization (or taxonomy) based on algorithms as shown in Fig. 1.

The remainder of this article is structured as follows. After introducing instance-based stochastic search algorithms (category [2.1]), Section 2 describes modeling approaches for stochastic algorithms, i.e., it

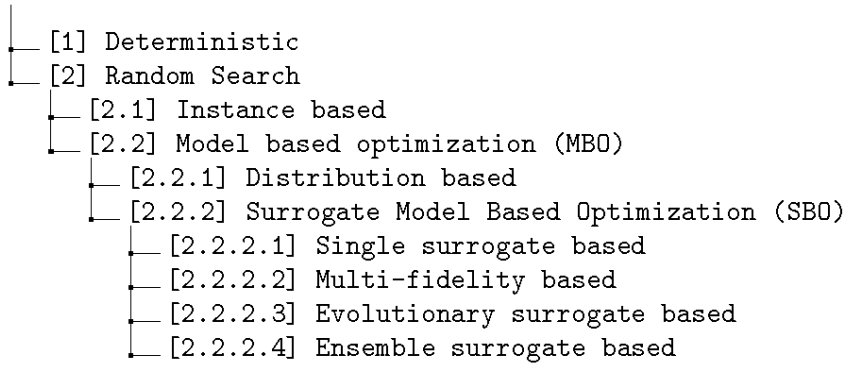


Figure 1: Taxonomy of model-based approaches in GO

refers to category [2.2.]. This category will be referred to as *model-based optimization* (MBO).

We differentiate between models, which use a distribution ([2.2.1]) and models that use an explicit surrogate model ([2.2.2]). Model-based optimization is the first choice for many optimization problems in industry. Section 3 describes typical applications, illustrating the practical relevance of MBO. Fundamental aspects of and recent advances in surrogate-model based optimization are discussed in Section 4. Strategies for selecting and evaluating surrogates are presented in Section 5. Two MBO algorithms, namely EvoLS and SPO, are presented in Section 6. Finally, a summary and an outlook are given in Section 7.

2. Stochastic Search Algorithms

2.1 Instance-Based Algorithms

Instance-based algorithms ([2.1]) maintain a *single solution*, $\mathbf{x}$, or *population*, $P(t)$, of candidate solutions. The iteration or time step is denoted as t. The construction of new candidate solutions depends explicitly on the previously generated solutions. Simulated annealing [36], evolutionary algorithms (EAs) [4], and tabu search [19] are prominent representatives of this category. The key elements of instance-based algorithms are shown in Algorithm 1.

2.2 MBO: Model-Based Algorithms

Model-based optimization algorithms ([2.2]) generate a population of new candidate solutions $P'(t)$ by sampling from a model (or a distri-

Algorithm 1 Instance-based Algorithm

1: $t = 0$. SetInitialPopulation(P)
2: Evaluate(P).
3: **while** not TerminationCriterion() **do**
4: Generate a set of new candidate solutions P'(t) according to a specified random mechanism.
5: Update the current population P(t+1) based on population P(t) and candidate solutions in P'(t).
6: Evaluate($P(t + 1)$).
7: $t = t + 1$.
8: **end while**

bution). The model (distribution) reflects structural properties of the underlying true function f. They are based on the idea that by adapting the model (or the distribution), the search is directed into regions with improved solutions.

One of the key ideas in MBO is the replacement of expensive, high fidelity, fine grained function evaluations, $f(\mathbf{x})$, with evaluations, $\hat{f}(\mathbf{x})$, of an adequate cheap, low fidelity, coarse grained model, M. After presenting typical examples in Section 3, two different approaches for generating cheap models will be presented in Section 4.

3. Applications of MBO

Simulation-based design of complex engineering problems, e.g., structural design of vehicles, use *computational fluid dynamics* (CFD) and *finite element modeling* (FEM) methods. The solvers require a large number of computer simulations to guarantee an exact solution. Hence, this is one of the most popular and successful application areas for MBO. There are two variants of MBO in this field of application: (i) metamodel (category [2.2.2.1]) and (ii) multi-fidelity approximation (category [2.2.2.2]) approaches. The former approach uses *one or several different* metamodels, whereas the latter uses several instances with different parameterizations of the *same* metamodel.

3.1 Metamodels

There are several publications that describe metamodeling approaches in aerospace design. The development of effective numerical methods for managing the use of approximation concepts in optimization for a 31-variable helicopter rotor design, which was part of a collaboration between Boeing, IBM, and Rice University, is described by Booker et

al. [7, 8]. Giannakoglou [18] discusses an aerodynamic shape design problem. Queipo et al. [51] present a multi-objective optimal design of a liquid rocket injector and discuss fundamental problems that arise in MBO. A surrogate-assisted evolutionary optimization framework, which is applied to an airfoil shape optimization problem using computational fluid dynamic is presented in [70]. Forrester and Keane [17] describe recent advances of MBO in aerospace design.

The design of ship propellers in the field of ship propulsion technology is described by Emmerich and Hundemer [14]. The authors model the features of a propeller design as a function of its resulting efficiency, torque coefficients, thrust coefficients, and cavitation. An implementation of a first-order potential-based panel method is used to calculate the hydrodynamic performance of a given propeller.

Li et al. [44] describe the optimization of feature detectors in ultrasound images. They present a study of *radial basis function networks* (RBFN) for metamodeling in heterogeneous, i.e., mixed-integer, parameter spaces.

Although the application of metamodeling techniques has progressed remarkably in the past last decades, the question remains "How far have we really come?" This issue is addressed in [59].

3.2 Multi-Fidelity Approximation

In addition to metamodels, *multi-fidelity metamodeling* methods have been developed. Multi-fidelity metamodeling uses several models of the same real system, where each model has its own degree of detail representing the real process. A typical example is the use of several simulation models with different grid sizes in FEM [26].

Sun et al. [60] describe a multi-fidelity optimization approach for sheet metal forming process. Further examples of multi-fidelity metamodeling are presented in [63]. The authors analyze the performance of Kriging [33] when multi-fidelity gradient data is introduced along with multi-fidelity function data to approximate black-box simulations.

Koziel et al. [39] present a methodology for fast multi-objective antenna optimization with *co-Kriging*. Co-Kriging is an extension of Kriging, which uses the correlations between the models of various fidelities, so that cheap- and expensive simulation data can be combined into one metamodel [15, 35]. Co-Kriging-based sequential design strategies are presented by Le Gratiet and Cannamela [43]. The authors simulate a spherical tank under internal pressure. Further applications from the water industry are published by Razavi et al [53]. Tuo et al. [62] proposed

a finite-element analysis with its mesh density as the tuning parameter. A problem in casting simulation is used to illustrate this approach.

Kleijnen [37] presents an overview of the most recent approaches in simulation practice. The book covers multi-fidelity metamodeling as well.

4. Key Elements of MBO

This section describes two different MBO approaches: (i) distribution based ([2.2.1]) and (ii) surrogate-model based optimization ([2.2.2.]).

4.1 Distribution-Based Approaches

If the metamodel is a distribution, the most basic form of an MBO can be implemented as shown in Algorithm 2:

Algorithm 2 Distribution-based Algorithm

1: $t = 0$. Let $p(t)$ be a probability distribution.
2: **while** not TerminationCriterion() **do**
3: Randomly generate a population of candidate solutions $P(t)$ from $p(t)$.
4: Evaluate($P(t)$).
5: Update the distribution using population (samples) $P(t)$ to generate a new distribution $p(t+1)$.
6: $t = t + 1$.
7: **end while**

Distribution-based algorithms generate a sequence of iterates (probability distributions) $\{p(t)\}$ with the hope that

$$p(t) \to p^* \text{ as } t \to \infty,$$

where p^* is a limiting distribution, which assigns most of its probability mass to the set of optimal solutions. So it is the probability distribution (as opposed to candidate solutions as in instance-based algorithms) that is propagated from one iteration to the next.

Estimation of distribution algorithms (EDA) are popular distribution-based algorithms, which became popular in the field of evolutionary algorithms [41]. Variation operators such as mutation and recombination, which modify candidate solutions (so-called individuals in EA), were replaced by a distribution based procedure: the new population of candidate solutions is generated according to the probability distribution induced or estimated from the promising candidate solution from the

current population. Larraaga and Lozano [41] review different ways of using probabilistic models as EDA instantiations.

Although distribution-based approaches play an important role in GO, they will not be discussed further in this article. The reader is referred to [24]. The authors discuss advantages and outline many of the different types of EDAs. In addition, Hu et al. [25] present recent approaches and a unified view on distribution-based approaches. We will concentrate on surrogate model-based approaches, which have their origin in statistical design and analysis of experiments, especially in response surface methodology.

4.2 Surrogate Model-Based Approaches

In general, surrogates are used, when the outcome of a process cannot be directly measured. Surrogates imitate the behavior of the real model as closely as possible, while being computationally cheaper to evaluate. The surrogate model is also known as a response surface, metamodel, approximation, coarse grained, or simply the *cheap* model. Simple surrogate models are constructed using a data-driven approach. They can be refined by integrating additional points or domain knowledge, e.g., constraints, into the surrogate.

A minimalistic *surrogate model-based optimization* (SBO) algorithm is shown in Algorithm 3. A wide range of surrogates was applied in the last

Algorithm 3 Surrogate Model Based Optimization (SBO) Algorithm

1: $t = 0$. SetInitialPopulation($P(t)$)
2: Evaluate($P(t)$)
3: **while** not TerminationCriterion() **do**
4: Use $P(t)$ to build a cheap model $M(t)$
5: $P'(t+1)$ = GlobalSearch($M(t)$)
6: Evaluate($P'(t+1)$)
7: $P(t+1) \subseteq P(t) + P'(t+1)$
8: $t = t + 1$
9: **end while**

decades. Classical regression models such as *polynomial regression* or response surface methodology [9], *support vector machines* (SVM) [65], artificial neural networks [72], radial basis functions [49], or *Gaussian process* (GP) models, which are sometimes referred to as *design and analysis of computer experiments* or Kriging [2, 11, 38, 56, 57] are the most prominent approaches. Forrester et al. [16] present a comprehensive introduction to SBO with several examples. Table 1 in [66] lists

popular metamodeling techniques and the related components such as experimental design, sampling methods, metamodels, and model fitting techniques.

4.3 Surrogate-Assisted Evolutionary Algorithms

Surrogate-assisted evolutionary algorithms (category [2.2.2.3]) are evolutionary algorithms that decouple the evolutionary search and the direct evaluation of the objective function. A cheap surrogate model, M, replaces evaluations of an expensive objective function, f.

A combination of a genetic algorithm and neural networks for aerodynamic design optimization is suggested in [22]. Ratle [52] creates an approximate model of the fitness landscape using Kriging interpolation to accelerate the convergence of EAs. Jin and et al. [31] investigate the convergence property of an *evolution strategy* (ES) with neural network based fitness evaluations. Emmerich et al. [13] present several MBO approaches for ES. Jin [30] presents a survey of surrogate-assisted evolutionary algorithms approaches. Jin and Sendhoff [32] use clustering techniques and neural networks ensembles to reduce the number of function evaluations. Branke and Schmidt [10] propose not evaluate every candidate solution (individual), but to just estimate the objective function value of some of the individuals. The reduction in the number of function evaluations is obtained by estimating an individual's function value on the basis of previously observed objective function values of neighboring individuals. Zhou et al. [70] present a surrogate-assisted EA framework, which uses computationally cheap hierarchical surrogate models constructed through online learning to replace the exact computationally expensive objective functions during evolutionary search.

5. Quality Criteria: How to Select Surrogates

The model building and selection process is crucial for the effectivity and efficiency of SBO. Fundamental for the improvement of a selected surrogate model as well as for the selection of an alternative surrogate model type is the evaluation of the expensive (true) objective function, which requires the determination of sample points. In the selection of adequate sample points, two conflicting goals have to be satisfied. The sample points can be selected with respect to

- exploration, i.e., improving the model quality (related to the model M) or
- exploitation, i.e., improving the optimization and determining the optimum (related to the objective function f).

Furthermore, regarding the model choice, the user can decide whether to use a

- single model, i.e., one unique global model is used during the optimization or
- multiple models, i.e., an ensemble of different, possibly local, models.

The static SBO uses a single, global surrogate model, which is usually refined by *adaptive sampling*. The same model type, e.g., Kriging interpolation, is used during the optimization. This is category [2.2.2.1] in Fig. 1.

5.1 Model Refinement

Adaptive sampling, a well-known selection strategy, proceeds as follows: An initial model, which uses a limited amount of sample points from the expensive objective function, is refined during the optimization. Adaptive sampling identifies new points, so-called *infill points*. Adaptive sampling tries to find a balance between exploration, i.e., improving the overall, global quality of the surrogate model, and exploitation, i.e., improving the local quality (in the region of the actual optimum), of the surrogate model. A popular adaptive sampling method is *expected improvement* (EI) [34, 45], which is discussed in [33]. The EI approach handles the initialization and refinement of a surrogate model, but not the selection of the model itself. The popular *efficient global optimization* (EGO) algorithm uses a Kriging model, because Kriging inherently determines the prediction variance, which is necessary for the EI criterion.

But there is no proof that Kriging is the best choice. Alternative surrogate models, e.g., regression trees, support vector machine, or lasso and ridge regression may be better suited. An *a priory* selection of the best suited surrogate model is conceptually impossible in the framework treated in this article, because of the black-box setting described in Section 1.

5.2 Multiple Models

Instead of using one surrogate model only, several models M_i, $i = 1, 2, \ldots, p$, can be generated and evaluated in parallel. Each model uses the same candidate solutions (from the population P) and results from expensive function evaluations.

Multiple models can also be used to partition the search space. The *tree-based Gaussian process* (TGP) approach uses regression trees to

partition the search space into separate regions and to fit local GP surrogates in each region [21]. Nelson et al. [47] propose an algorithm, that creates a tree-based partitioning of an aerodynamic design space and employs independent Kriging surfaces in each partition. Couckuyt et al. [12] propose to combine an *evolutionary model selection* (EMS) algorithm with the EI criterion in order to dynamically select the best performing surrogate model type at each iteration of the EI algorithm. A new expensive sample point, $\mathbf{x}'$, is chosen based on the EI criterion at each iteration step t. The point $\mathbf{x}'$ itself is based on the best surrogate model found by the EMS algorithm.

In the last decade, ensembles of surrogate models gained popularity (category [2.2.2.4]) in Fig. 1. Zerpa et al. [69] use multiple surrogate models and build an adaptive weighted average model of the individual surrogates. Goel at al. [20] explore the possibility of using the best surrogate model or a weighted average surrogate model instead of one single model. Model quality, i.e., the errors in surrogates, is used to determine the weights assigned to each model. Sanchez et al. [55] present a weighted-sum approach for the selection of model ensembles. The models for the ensemble are chosen based on their performance and the weights are adaptive and inversely proportional to the local modeling errors.

Recent approaches such as the evolvability learning of surrogates approach implement local models for each offspring individually [42]. This results in an adaptive semi-partition [40] of the search space.

5.3 Criteria for Selecting a Surrogate

Note, this paragraph does not consider the selection of a new sample point as done in EI. Here, we consider criteria for the selection of one (or several) surrogate models, e.g., Kriging models or SVMs [65].

Conventionally, surrogate models are assessed and chosen according to their estimated true error [29, 58]. The *mean absolute error* (MAE) and the *root mean square error* (RMSE) are commonly used as performance metrics. Error measures are discussed in [28]. Willmott and Matsuura [67] presents a comparison of MAE and RMSE. Generally, attaining a surrogate model that has minimal error is the desired feature. Methods from statistics, statistical learning [23], and machine learning [46], such as the simple holdout approach, cross-validation, and the bootstrap are used to choose adequate surrogate models. Tenne and Armfield [61] propose a surrogate-assisted memetic algorithm which generates accurate surrogate-models using multiple cross-validation tests. However, the definition of the corresponding training sets (sampling)

represents a critical issue for the accuracy and efficiency of the meta-models.

The model error is not the only criterion for selecting surrogate models. In contrast to the surrogate model selection approaches so far, the evolvability learning of surrogates approach [42], which will be presented in Section 6.1, uses fitness improvement for determining the quality of surrogate models in enhancing search improvement.

6. Examples

6.1 Evolvability Learning of Surrogates

The *evolvability learning of surrogates* (EvoLS) algorithm, which is introduced by Le et al. [42], belongs to the category of *surrogate-assisted evolutionary algorithms* ([2.2.2.3]).

The authors of EvoLS recommend selecting surrogate models that enhance search improvement in the context of optimization. EvoLS processes information about the (i) different fitness landscapes, (ii) state of the search, and (iii) characteristics of the search algorithm to statistically determine the so-called *evolvability* of each surrogate model. The evolvability of a surrogate model estimates the expected improvement of the objective function value that the new candidate solution has gained after a local search has been performed on the related surrogate model. Three basic steps are necessary for calculating the evolvability (a detailed calculation is presented in [42]):

- Variation. Let $\mathbf{x}$ denote the parent and $\mathbf{y}$ be the offspring generated from $\mathbf{x}$ by evolutionary variation operators, e.g., mutation and/or recombination. Le at al. [42] make a simplified assumption of uniformity in the offspring distribution. Let $V(R)$ denote the volume of an n-dimensional cuboid

$$R = \left[\min_{j=1..N} \{x_j^{(i)}\}, \max_{j=1..N} \{x_j^{(i)}\} \right]_{i=1,..,n}.$$

  The density distribution is modeled as

$$P(\mathbf{y} \,|P(t), \mathbf{x}) = \mathcal{U}(R) = \begin{cases} 1/(R) & \text{if } y \in R \\ 0 & \text{otherwise.} \end{cases}$$

  The evolutionary variation operators recombination and uniform mutation force the offspring to be located in the n-dimensional region R. To determine the probability at time step t of moving from parent $\mathbf{x}$ via stochastic variation, the followoing weights can

be used:

$$w_i(\mathbf{x}) = \frac{P(\mathbf{y}_i \,|P(t), \mathbf{x})}{\sum_{j=1}^{K} P(\mathbf{y}_j \,|P(t), \mathbf{x})}.$$

The weight measures the influence of the samples $(\mathbf{y}_i, \varphi_M(y_i))$ on the evolvability.

- Local search. After recombination and mutation, a local search is performed. It uses a local metamodel, M, for each offspring. The local optimizer, φ_M, uses an offspring $\mathbf{y}$ as an input and returns $\mathbf{y}^*$ as the refined offspring. The local optimizer on the surrogate model guarantees (theoretically) convergence to the stationary point of the exact objective function [1, 48].

- Evolvability. Finally, the evolvability measure can be estimated as follows:

$$Ev_M(\mathbf{x}) = f(\mathbf{x}) - \sum_{i=1}^{K} f(\mathbf{y}_i^*) \times w_i(\mathbf{x}).$$

6.2 Sequential Parameter Optimization

Early versions of the *sequential parameter optimization* (SPO) combined methods from *design of experiments* (DOE), *response surface methodology* (RSM), *design and analysis of computer experiments* (DACE), and regression trees for the analysis of algorithms [3, 5, 6]. The SPO was developed as a tool for the analysis and for an understanding of the working principles of EAs. The SPO tools might as well be integrated into the evolutionary loop and therefore improve performance of an EA. This consideration lays the cornerstone for the development of the SPO as an optimizer.

Subsequent versions of the SPO use a sequential, model based approach to optimization. Nowadays, the SPO is an established parameter tuner and an optimization algorithm, which has been extended in several ways. For example, Hutter et al. [27] benchmark an SPO derivative, the so-called *sequential model-based algorithm configuration* (SMAC) procedure, on the BBOB set of blackbox functions. They demonstrate that with a small budget of $10 \times d$ evaluations of d-dimensional functions, SMAC in most cases outperforms the state-of-the-art blackbox optimizer CMA-ES.

The most recent version, SPO2, is currently under development. It will integrate state-of-the-art ensemble learners. The SPO2 *ensemble engine* can be briefly outlined as follows: The portfolio of surrogate models includes a pleiotropy of metamodels such as regression trees and random forest, *least angle regression* (LARS), and Kriging. The SPO2

ensemble engine uses cross validation to select an improved model from the portfolio of candidate models [64]. It implements methods for creating a weighted combination of several surrogate models to build the improved model and methods, which use stacked generalization to combine several level-0 models of different types with one level-1 model into an ensemble [68]. The level-1 training algorithm is typically a relatively simple linear model.

Preliminary results indicate that the SPO2 ensemble engine can lead to significant performance improvements of the SPO algorithms, which is illustrated by the following example: Rebolledo et al. [54] present a comparison of different data driven modeling methods. The first instance of a data driven linear Bayesian model is compared with several linear regression models, a Kriging model and a genetic programming model. The models are built on industrial data for the development of a robust gas sensor. The data contain limited amount of samples and a high variance. The mean square error of the models implemented in a test dataset is used as the comparison strategy. Two sensors were tested in this comparison. The mean squared errors are as follows. Linear model (0.76), OLS (0.79), lasso (0.56), Kriging (0.57), Bayes (0.79), and genetic programming (0.58). SPO2 obtained an MSE of 0.38, which outperforms the best model. Results from the second sensor read as follow: Linear model (0.67), OLS (0.80), lasso (0.49), Kriging (0.49), Bayes (0.79), and genetic programming (0.27). Here, SPO2 obtained an MSE of 0.29.

This first real-world application example demonstrates the potential of SBO with ensembles (category [2.2.2.4]).

7. Summary

Especially in the engineering domain, model-based approaches are probably the most efficient methods for expensive and time-demanding real-world optimization problems. This article proposed a taxonomy of model based algorithms for global optimization problems. The taxonomy was developed from an algorithm-centered perspective. The categorization scheme, which started with a bird's eye view on GO, was refined as summarized in Fig. 1. Finally, working principles of two state-of-the-art MBO algorithms were shown. EvoLS, which constructs a metamodel for every new candidate solution, and SPO2, which uses an ensemble engine to combine a broad variety of surrogate models. The survey presented in the first sections of this article as well as the examples in Section 6 emphasize the trend to ensemble based metamodels.

Acknowledgement: This work has been supported by the *Bundesministeriums fur Wirtschaft und Energie* under the grants KF3145101WM3

und KF3145103WM4. This work is part of a project that has received funding from the *European Unions Horizon 2020 research and innovation program* under grant agreement No 692286.

References

[1] N. M. Alexandrov, J. E. Dennis Jr, R. M. Lewis, and V. Torczon. A trust-region framework for managing the use of approximation models in optimization. *Structural Optimization*, 15(1):16–23, 1998.

[2] A. B. Antognini and M. Zagoraiou. Exact optimal designs for computer experiments via Kriging metamodelling. *Journal of Statistical Planning and Inference*, 140(9):2607–2617, 2010.

[3] T. Bartz-Beielstein. Experimental Analysis of Evolution Strategies—Overview and Comprehensive Introduction. Technical report, Nov. 2003.

[4] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann. Evolutionary Algorithms. *WIREs Data Mining and Knowledge Discovery*, 4:178–195, 2014.

[5] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential Parameter Optimization. *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 773–780, 2005.

[6] T. Bartz-Beielstein, K. E. Parsopoulos, and M. N. Vrahatis. Design and analysis of optimization algorithms using computational statistics. *Applied Numerical Analysis and Computational Mathematics*, 1(2):413–433, 2004.

[7] A. J. Booker, J. E. Dennis Jr, P. D. Frank, D. B. Serafini, and V. Torczon. Optimization Using Surrogate Objectives on a Helicopter Test Example. In *Computational Methods for Optimal Design and Control*, pages 49–58. Birkhäuser Boston, Boston, MA, 1998.

[8] A. J. Booker, J. E. Dennis Jr, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, 1999.

[9] G. E. P. Box and K. B. Wilson. On the Experimental Attainment of Optimum Conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):1–45, 1951.

[10] J. Branke and C. Schmidt. Faster convergence by means of fitness estimation. *Soft Computing*, 9(1):13–20, 2005.

[11] D. Büche, N. N. Schraudolph, and P. Koumoutsakos. Accelerating Evolutionary Algorithms With Gaussian Process Fitness Function Models. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 35(2):183–194, 2005.

[12] I. Couckuyt, F. De Turck, T. Dhaene, and D. Gorissen. Automatic surrogate model type selection during the optimization of expensive black-box problems. *Proceedings of the Winter Simulation Conference (WSC)*, pages 4269–4279, 2011.

[13] M. Emmerich, A. Giotis, M. özdemir, T. Bäck, and K. Giannakoglou. Metamodel-assisted evolution strategies. *Lecture Notes in Computer Science*, 2439:361–370, 2002.

[14] M. Emmerich, J. Hundemer, M.-C. Varcol, B. Naujoks, and M. Abdel-Maksoud. Design Optimization of Ship Propellers by Means of Advanced Metamodel-Assisted Evolution Strategies. *Proceedings of the International Conference on Design Optimization (ERCOFTAC)*, 2006 .

[15] A. Forrester, A. Sóbester, and A. Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 463(2088):3251–3269, 2007.

[16] A. Forrester, A. Sóbester, and A. Keane. *Engineering Design via Surrogate Modelling*. Wiley, 2008.

[17] A. I. J. Forrester and A. J. Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1-3):50–79, 2009.

[18] K. C. Giannakoglou. Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence. *Progress in Aerospace Sciences*, 38(1):43–76, 2002.

[19] F. Glover and M. Laguna. Tabu Search. In C. Reeves (Ed.) *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, U.K., Blackwell Scientific Publishing, 1993.

[20] T. Goel, R. T. Haftka, W. Shyy, and N. V. Queipo. Ensemble of surrogates. *Structural and Multidisciplinary Optimization*, 33(3):199–216, 2006.

[21] R. B. Gramacy. tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models. *Journal of Statistical Software*, 19(9):1–46, 2007.

[22] P. Hajela and E. Lee. Topological optimization of rotorcraft subfloor structures for crashworthiness considerations. *Computers & Structures*, 64(1-4):65–76, 1997.

[23] T. Hastie. *The elements of statistical learning : data mining, inference, and prediction*. Springer, New York, 2nd ed., 2009.

[24] M. Hauschild and M. Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111–128, 2011.

[25] J. Hu, Y. Wang, E. Zhou, M. C. Fu, and S. I. Marcus. A Survey of Some Model-Based Methods for Global Optimization. In D. Hernández-Hernández and J. A. Minjárez-Sosa (Eds.) *Optimization, Control, and Applications of Stochastic Systems*, pages 157–179. Birkhäuser Boston, Boston, 2012.

[26] E. Huang, J. Xu, S. Zhang, and C. H. Chen. Multi-fidelity Model Integration for Engineering Design. *Procedia Computer Science*, 44:336–344, 2015.

[27] F. Hutter, H. Hoos, and K. Leyton-Brown. An Evaluation of Sequential Model-based Optimization for Expensive Blackbox Functions. *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 1209–1216, 2013.

[28] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[29] R. Jin, W. Chen, and T. W. Simpson. Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and Multidisciplinary Optimization*, 23(1):1–13, 2001.

[30] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2003.

[31] Y. Jin, M. Olhofer, and B. Sendhoff. On Evolutionary Optimization with Approximate Fitness Functions. *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO)*, 2000.

[32] Y. Jin and B. Sendhoff. Reducing Fitness Evaluations Using Clustering Techniques and Neural Network Ensembles. *Lecture Notes in Computer Science*, 3102:688–699, 2014.

[33] D. R. Jones. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

[34] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13:455–492, 1998.

[35] M. Kennedy. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[37] J. P. C. Kleijnen. *Design and Analysis of Simulation Experiments*. International Series in Operations Research and Management Science. Springer International Publishing, 2015.

[38] J. P. C. Kleijnen. Kriging metamodeling in simulation: A review. *European Journal of Operational Research*, 192(3):707–716, 2009.

[39] S. Koziel, A. Bekasiewicz, I. Couckuyt, and T. Dhaene. Efficient Multi-Objective Simulation-Driven Antenna Design Using Co-Kriging. *IEEE Transactions on Antennas and Propagation*, 62(11):5900–5905, 2014.

[40] M. Kryszkiewicz, J. F. Peters, H. Rybinski, and A. Skowron (Eds.) *Rough Sets and Intelligent Systems Paradigms*, volume 4585 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2007.

[41] P. Larraaga and J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer, Boston MA, 2002.

[42] M. N. Le, M. N. Le, Y. S. Ong, Y. S. Ong, S. Menzel, S. Menzel, Y. Jin, Y. Jin, B. Sendhoff, and B. Sendhoff. Evolution by adapting surrogates. *Evolutionary Computation*, 21(2):313–340, 2013.

[43] L. Le Gratiet and C. Cannamela. Kriging-based sequential design strategies using fast cross-validation techniques with extensions to multi-fidelity computer codes. *arXiv.org*, Oct. 2012.

[44] R. Li, M. T. M. Emmerich, J. Eggermont, E. G. P. Bovenkamp, T. Bäck, J. Dijkstra, and J. H. C. Reiber. Metamodel-assisted mixed integer evolution strategies and their application to intravascular ultrasound image analysis. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2764–2771, 2008.

[45] J. Mockus, V. Tiesis, and A. Zilinskas. Bayesian Methods for Seeking the Extremum. In L. C. W. Dixon and G. P. Szegö (Eds.) *Towards Global Optimization*, pages 117–129. Amsterdam, 1978.

[46] K. P. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[47] A. Nelson, J. Alonso, and T. Pulliam. Multi-Fidelity Aerodynamic Optimization Using Treed Meta-Models. *Proceedings of the Fluid Dynamics and Co-located Conferences*, 2007.

[48] Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modeling. *AIAA Journal*, 41(4):687–696, 2003.

[49] M. Powell. Radial Basis Functions. *Algorithms for Approximation*, 1987.

[50] M. Preuss. *Multimodal Optimization by Means of Evolutionary Algorithms*. Natural Computing Series. Springer International Publishing, Cham, 2015.

[51] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. Kevin Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1):1–28, 2005.

[52] A. Ratle. Accelerating the Convergence of Evolutionary Algorithms by Fitness Landscape Approximation. *Lecture Notes in Computer Science*, 1498:87–96, 1998.

[53] S. Razavi, B. A. Tolson, and D. H. Burn. Review of surrogate modeling in water resources. *Water Resources Research*, 48(7):n/a–n/a, 2012.

[54] M. A. Rebolledo Coy, S. Krey, T. Bartz-Beielstein, O. Flasch, A. Fischbach, and J. Stork. Modeling and Optimization of a Robust Gas Sensor. Technical Report 03/2016, Cologne Open Science, Cologne, 2016.

[55] E. Sanchez, S. Pintos, and N. V. Queipo. Toward an Optimal Ensemble of Kernel-based Approximations with Engineering Applications. *Proceedings of the IEEE International Joint Conference on Neural Network Proceedings*, pages 2152–2158, 2006.

[56] T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer, Berlin, Heidelberg, New York, 2003.

[57] M. Schonlau. Computer Experiments and Global Optimization. PhD thesis, University of Waterloo, Ontario, Canada, 1997.

[58] L. Shi and K. Rasheed. A Survey of Fitness Approximation Methods Applied in Evolutionary Algorithms. In *Computational Intelligence in Expensive Optimization Problems*, pages 3–28. Springer, Berlin, Heidelberg, 2010.

[59] T. Simpson, V. Toropov, V. Balabanov, and F. Viana. Design and Analysis of Computer Experiments in Multidisciplinary Design Optimization: A Review of How Far We Have Come - Or Not. *Proceedings of the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, pages 1–22, 2012.

[60] G. Sun, G. Li, S. Zhou, W. Xu, X. Yang, and Q. Li. Multi-fidelity optimization for sheet metal forming process. *Structural and Multidisciplinary Optimization*, 44(1):111–124, 2011.

[61] Y. Tenne and S. W. Armfield. A Versatile Surrogate-Assisted Memetic Algorithm for Optimization of Computationally Expensive Functions and its Engineering Applications. In *Success in Evolutionary Computation*, pages 43–72. Springer, Berlin, Heidelberg, 2008.

[62] R. Tuo, C. F. J. Wu, and D. Yu. Surrogate Modeling of Computer Experiments With Different Mesh Densities. *Technometrics*, 56(3):372–380, 2014.

[63] S. Ulaganathan, I. Couckuyt, F. Ferranti, E. Laermans, and T. Dhaene. Performance study of multi-fidelity gradient enhanced kriging. *Structural and Multidisciplinary Optimization*, 51(5):1017–1033, 2014.

[64] M. J. van der Laan and S. Dudoit. Unified Cross-Validation Methodology For Selection Among Estimators and a General Cross-Validated Adaptive Epsilon-Net Estimator: Finite Sample Oracle Inequalities and Examples. 2003.

[65] V. N. Vapnik. *Statistical learning theory*. Wiley, 1998.

[66] G. G. Wang and S. Shan. Review of Metamodeling Techniques in Support of Engineering Design Optimization. *Journal of Mechanical Design*, 129(4):370–380, 2007.

[67] C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(7982):1–4, 2005.

[68] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[69] L. E. Zerpa, N. V. Queipo, S. Pintos, and J.-L. Salager. An optimization methodology of alkaline–surfactant–polymer flooding processes using field scale numerical simulation and multiple surrogates. *Journal of Petroleum Science and Engineering*, 47(3-4):197–208, 2005.

[70] Z. Zhou, Y. S. Ong, P. B. Nair, A. J. Keane, and K. Y. Lum. Combining Global and Local Surrogate Models to Accelerate Evolutionary Optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 37(1):66–76, 2007.

[71] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-Based Search for Combinatorial Optimization: A Critical Survey. *Annals of Operations Research*, 131(1-4):373–395, 2004.

[72] J. M. Zurada. Analog implementation of neural networks. *IEEE Circuits and Devices Magazine*, 8(5):36–41, 1992.

PARALLEL MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS

El-Ghazali Talbi
University of Lille 1, Villeneuve d'Ascq, France
el-ghazali.talbi@univ-lille1.fr

Abstract This paper describes a general overview of parallel multi-objective evolutionary algorithms (MOEA) from the design and the implementation point of views. A unified taxonomy using three hierarchical parallel models is proposed. Different parallel architectures are considered. The performance evaluation issue of parallel MOEA is also discussed.

Keywords: Multi-objective optimization, Parallel evolutionary algorithms.

1. Motivation

On one hand, multi-objective optimization problems (MOPs), such as in engineering design and life science, are more and more complex and their resource requirements to solve them are ever increasing. Real-life MOPs are often NP-hard, and CPU time and/or memory consuming. Although the use of multi-objective evolutionary algorithms (MOEAs) allows to significantly reduce the computational complexity of the solving algorithms, the latter remains time-consuming for many MOPs in diverse domains of application, where the objective function and the constraints associated to the problem are resource (e.g., CPU, memory) intensive and the size of the search space is huge. Moreover, more and more complex and resource intensive MOEAs are developed to obtain a good approximation of the Pareto front in a reasonable time.

On the other hand, the rapid development of technology in designing processors (e.g., multi-core processors, dedicated architectures), networks (local networks (LAN) such as Myrinet and Infiniband or wide area networks (WAN) such as optical networks), and data storage make the use of parallel computing more and more popular. Such architectures represent an effective opportunity for the design and implementation of parallel multi-objective optimization algorithms. Indeed, sequential architectures are reaching physical limitations (speed of light, thermody-

namics). Nowadays, even laptops and workstations are equipped with multi-core processors, which represent one class of parallel architecture. Moreover, the ratio cost/performance is constantly decreasing. The proliferation of powerful processors and fast communication networks have shown the emergence of dedicated architectures (e.g GPUs), clusters of processors (COWs), networks of workstations (NOWs), large-scale networks of machines (Grids) and Clouds as platforms for high performance computing.

Parallel computing can be used in the design and implementation of MOEAs for the following reasons:

- **Speedup the search to approximate the Pareto front**: The goal here is to reduce the search time. This helps designing interactive optimization methods which is an important issue for multi-criteria decision making. This is a also an important aspect for some class of problems where there are hard requirements on search time such as in dynamic MOPs and time-critical operational MOPs such as "real-time" planning and control.

- **Improve the quality of the obtained Pareto solutions**: some parallel models for MOEAs allow to improve the quality of Pareto solutions. Indeed, exchanging information between algorithms will alter their behavior in terms of searching in the landscape associated to the MOP. The main goal in a cooperation between algorithms is to improve the quality of Pareto solutions. Both convergence to better Pareto solutions and reduced search time may happen. Let us notice that a parallel model for MOEAs may be more effective than a sequential algorithm even on a single processor.

- **Improve the robustness**: a parallel MOEA may be more robust in terms of solving in an effective manner different MOPs and different instances of a given problem. Robustness may be measured in terms of the sensitivity of the algorithm to its parameters and the target MOPs.

- **Solve large scale MOPs**: parallel MOEAs allow to solve large scale instances of complex MOPs. A challenge here is to solve very large instances that cannot be solved on a sequential machine. Another similar challenge is to solve more accurate mathematical models associated to different MOPs. Improving the accuracy of mathematical models increases in general the size of the associated problems to be solved. Moreover, some optimization prob-

lems need the manipulation of huge databases such as data mining problems.

In this paper, a clear difference is made between the parallel design aspect and the parallel implementation aspect of MOEAs. A unifying view of parallel models for MOEAs is presented. The implementation point of view deals with the efficiency of parallel MOEAs on a target parallel architecture using a given parallel language, programming environment or middleware. Different architectural criteria, which affect the efficiency of the implementation, will be considered: shared memory versus distributed memory, homogeneous versus heterogeneous, shared versus non shared by multiple users, local network versus large network. Indeed, those criteria have a strong impact on the deployment technique employed such as load balancing and fault-tolerance. Depending on the type of parallel architecture used, different parallel and distributed languages, programming environments and middlewares may be used such as message passing (e.g., MPI), shared memory (e.g., multithreading, OpenMP, CUDA), remote procedural call (e.g., Java RMI, RPC), high-throughput computing (e.g., Condor), and grid computing (e.g., Globus).

This paper is organized as follows. In Section 2, the main parallel models for designing MOEAs are presented. Section 3 deals with the implementation issues of parallel MOEAs. In this section, the main concepts of parallel architectures and parallel programming paradigms, which interfere with the design and implementation of parallel MOEAs are outlined. The main performance indicators that can be used to evaluate a parallel multi-objective search algorithms in terms of efficiency are detailed.

2. Parallel Design of Multi-Objective Metaheuristics

In terms of designing parallel MOEAs, three major parallel models are identified. They follow the three hierarchical levels (Table 1):

- **Algorithmic-level:** in this model, independent or cooperating self-contained MOEAs are used. It is a problem-independent inter-algorithm parallelization. If the different MOEAs are independent, the search will be equivalent to the sequential execution of the algorithms in terms of the quality of Pareto solutions. However, the cooperative model will alter the behavior of the MOEAs and enable the improvement in terms of the quality of Pareto solutions.

- **Iteration-level:** in this model, each iteration of a MOEA is parallelized. It is a problem-independent intra-algorithm parallelization. The behavior of the MOEA is not altered. The main objective is to speedup the algorithm by reducing the search time. Indeed, the iteration cycle of MOEAs on large populations, especially for real-world MOPs, requires a large amount of computational resources.

- **Solution-level:** in this model, the parallelization process handles a single solution of the search space. It is a problem-dependent intra-algorithm parallelization. In general, evaluating the objective functions or constraints for a generated solution is frequently the most costly operation in MOEAs. In this model, the behavior of the search algorithm is not altered. The objective is mainly the speedup of the search.

Table 1: Parallel models of MOEAs.

Parallel model	Problem dependency	Behavior	Granularity	Goal
Algorithmic-level	Independent	Altered	MOP algorithm	Effectiveness
Iteration-level	Independent	Non altered	Iteration	Efficiency
Solution-level	Dependent	Non altered	Solution	Efficiency

2.1 Algorithmic-Level Parallel Model

In this model, many MOEAs are launched in parallel. They may cooperate or not to solve the target MOPs.

2.1.1 Independent algorithmic-level parallel model. In the independent-level parallel model, the different MOEAs are executed without any cooperation. The different MOEAs may be initialized with different populations. Different parameter settings may be used for the MOEAs such as the mutation and crossover probabilities. Moreover, each search component of an MOEA may be designed differently: encoding, search operators (e.g., variation operators), objective functions, constraints, fitness assignment, diversity preserving, elitism. This parallel model is straightforward to design and implement. The master/worker paradigm is well suited to this model. A worker implements an MOEA. The master defines the different parameters to use by the workers and determines the best found Pareto solutions from those obtained by the

different workers. In addition to speeding up the MOEA, this parallel model enables to improve its robustness [29].

This model raises particularly the following question: is it equivalent to execute k MOEAs during a time t and to execute a single MOEA during $k * t$? The answer depends on the landscape properties of the problem (e.g., distribution of the Pareto local optima).

2.1.2 Cooperative algorithmic-level parallel model. In the cooperative model for parallel MOEAs, the different MOEAs are exchanging information related to the search with the intent to compute a better and more robust Pareto front [30]. In general, an archive is maintained in parallel to the current population. This archive contains all Pareto optimal solutions generated during the search.

In designing this parallel cooperative model for any MOEA, the same design questions need to be answered:

- **The exchange decision criterion (When?)**: the exchange of information between the MOEAs can be decided either in a *blind* (periodic or probabilistic) way or according to an *"intelligent"* adaptive criterion. Periodic exchange occurs in each algorithm after a fixed number of iterations; this type of communication is synchronous. Probabilistic exchange consists in performing a communication operation after each iteration with a given probability. Conversely, adaptive exchanges are guided by some characteristics of the multi-objective search. For instance, it may depend on the evolution of the quality of the Pareto front. A classical criterion is related to the update of the archive, in which a new Pareto solution is generated.

- **The exchange topology (Where?)**: the communication exchange topology indicates for each MOEA its neighbor(s) regarding the exchange of information, i.e., the source/destination algorithm(s) of the information. The ring, mesh and hypercube regular topologies are the most popular ones.

- **The information exchanged (What?)**: this parameter specifies the information to be exchanged between the MOEAs. In general, the information exchanged is composed of:

  - Pareto solutions: this information deals with any selection strategy of the generated Pareto solutions during the search. In general, it contains solutions from the current population and/or the archive. The number of selected Pareto optimal solutions may be an absolute value or a percentage of the sets.

  - Search memory: this information deals with a search memory of a MOEA excluding the Pareto optimal solutions. This information deals with any element of the search memory that is associated to the involved MOEA.

- **The integration policy (How?)**: analogously to the information exchange policy, the integration policy deals with the usage of the received information. In general, there is a local copy of the received information. The local copies of the information received are generally updated using the received ones. The Pareto solutions received will serve to update the local Pareto archive. For the current population, any replacement strategy can be used (e.g., random, elitist). For instance, the best Pareto set is simply updated by the best between the local best Pareto set and the neighboring best Pareto set. Any replacement strategy may be applied on the local population by the set of received solutions.

Few of such parallel search models have been especially designed for multi-objective optimization [29].

The other well known parallel model for MOEAs, the cellular model. may be seen as a special case of the island model where an island is composed of a single individual. Traditionally, an individual is assigned to a cell of a grid. The selection occurs in the neighborhood of the individual. Hence, the selection pressure is less important than in sequential MOEAs. The overlapped small neighborhood in cellular MOEAs helps exploring the search space because a slow diffusion of Pareto solutions through the population provides a kind of exploration, while exploitation takes place inside each neighborhood. Cellular models applied to complex problems can have a higher convergence probability to better solutions than panmictic MOEAs [17].

The different MOEAs involved in the cooperation may evaluate different subsets of objective functions (Fig. 1). For instance, each MOEA may handle a single objective. Another approach consists in using a different aggregation weights in each MOEA, or different constraints [22].

Each MOEA may also represent a different partition of the decision space or the objective space [15, 27]. By this way, each MOEA is destined to find a particular portion of the Pareto-optimal front.

Another main issue in the development of parallel MOPs is how the Pareto set is built during the optimization process. Two different approaches may be considered (Fig. 1):

- *Centralized Pareto Front*: the front is a centralized data structure of the algorithm that it is built by the MOEAs during the whole

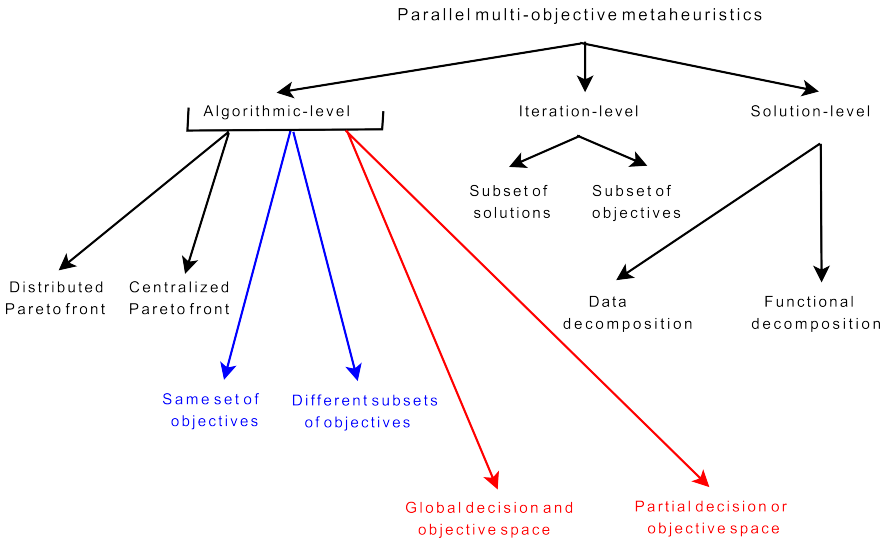


Figure 1: Classification of parallel MOEAs for multi-objective optimization.

computation. This way, the new non-dominated solutions in the Pareto optimal set are global Pareto optima [1, 5, 28].

- *Distributed Pareto Front*: the Pareto front is distributed among the MOEAs so that the algorithm works with local non-dominated solutions that must be somehow combined at the end of their work [8, 18, 19]. No pure centralized approach has been found clearly motivated by efficiency issues [16]. All the found centralized approaches are combined with distributed phases where local non-dominated solutions are considered. After each distributed phase, a single optimal Pareto front is built by using these local Pareto optima. Then, the new Pareto front is again distributed for local computation, and so on.

2.2 Iteration-Level Parallel Model

In this parallel model, a focus is made on the parallelization of each iteration of MOEAs. The iteration-level parallel model is generally based on the distribution of the handled solutions. Indeed, the most resource-consuming part in an MOEA is the evaluation of the generated solutions. Our concerns in this model are only search mechanisms that are problem-independent operations such as the generation of successive populations. Any *search operator* of an MOEA which is not specific to the tackled optimization problem is involved in the iteration-level parallel model.

This model keeps the sequentiality of the original algorithm, and, hence, the behavior of the MOEA is not altered.

It is the easiest and the most widely used parallel model in MOPs. Indeed, many MOPs are complex in terms of the objective functions. For instance, some engineering design applications integrate solvers dealing with different surrogate models: computational fluid dynamics (CFD), computational electromagnetics (CEM), or finite element methods (FEM). Other real-life applications deals with complex simulators. A particularly efficient execution is often obtained when the ratio between communication and computation is high. Otherwise, most of the time can be wasted in communications, leading to a poor parallel algorithm.

The population of individuals can be decomposed and handled in parallel. In *master-worker* a master performs the selection operations and the replacement. The selection and replacement are generally sequential procedures, as they require a global management of the population. The associated workers perform the recombination, mutation and the evaluation of the objective function. The master sends the partitions (subpopulations) to the workers. The workers return back newly evaluated solutions to the master [19] (Fig. 2).

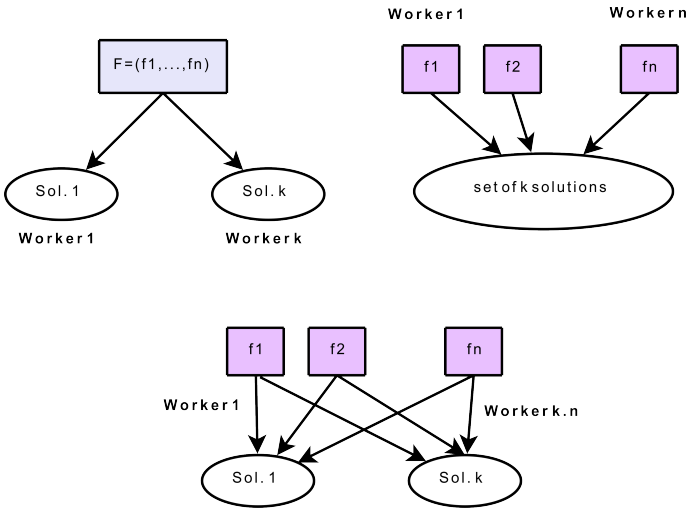


Figure 2: The iteration-level parallel model in parallel MOEAs.

According to the order in which the evaluation phase is performed in comparison with the other parts of the MOEA, two modes can be distinguished:

- **Synchronous:** in the synchronous mode, the worker manages the evolution process and performs in a serial way the different steps of selection and replacement. At each iteration, the master distributes the set of new generated solutions among the workers and waits for the results to be returned back. After the results are collected, the evolution process is re-started. The model does not change the behavior of the MOEA compared to a sequential model.

- **Asynchronous:** in the asynchronous mode, the worker does not wait for the return of all evaluations to perform the selection, reproduction and replacement steps. The *steady-state* MOEA is a good example illustrating the asynchronous model and its advantages. In the asynchronous model applied to a steady-state MOEA, the recombination and the evaluation steps may be done concurrently. The master manages the evolution engine and two queues of individuals of a given fixed size: individuals to be evaluated, and solutions being evaluated. The individuals of the first queue wait for a free evaluating node. When the queue is full the process blocks. The individuals of the second queue are assimilated into the population as soon as possible. The reproduced individuals are stored in a FIFO data structure, which represents the individuals to be evaluated. The MOEA continues its execution in an asynchronous manner, without waiting for the results of the evaluation phase. The selection and reproduction phase are carried out until the queue of non-evaluated individuals is full. Each evaluator agent picks an individual from the data structure, evaluates it, and stores the results into another data structure storing the evaluated individuals. The order of evaluation defined by the selection phase may not be the same as in the replacement phase. The replacement phase consists in receiving, in a synchronous manner, the results of the evaluated individuals, and applying a given replacement strategy of the current population.

In some MOEAs (e.g., blackboard-based ones) some information must be shared. For instance, in ant colony optimization (ACO), the pheromone matrix must be shared by all ants. The master has to broadcast the pheromone trails to each worker. Each worker handles an ant process. It receives the pheromone trails, constructs a complete solution, and evaluates it. Finally, each worker sends back to the master the constructed and evaluated solution. When the master receives all the constructed solutions, it updates the pheromone trails [14].

Ranking methods are used to assign a fitness to each solution of a population. Those ranking methods are computation-intensive and may be also parallelized. Updating the archives at each iteration is also a time consuming task.

2.3 Solution-Level Parallel Model

The main objective of the solution-level parallel model for MOP is to speedup the search by parallelizing the treatments dealing with single solutions (e.g., objectives evaluation, constraint satisfaction). Indeed, the evaluation of multiple objective functions in MOPs is the most time-consuming part into a MOEA. Therefore, several algorithms try to reduce this time by means of parallelizing the calculation of the fitness evaluation [21, 22, 23]. The classical approaches must be adapted to multi-objective optimization (Fig. 1):

- **Functional decomposition:** this approach consists in distributing the different objective functions among the workers, and each of them computes the value of its assigned function on each solution. The master will then aggregate the partial results for all the solutions. Such approach allows a degree of concurrency and the scalability is limited to the number of objective functions, meaning often 2 or 3. Moreover, each objective function may be decomposed into several sub-functions. Then, the degree of concurrency will be equal to the number of sub-functions.
- **Data decomposition:** for each data partition of the problem (database, geographical area, structure, ...), all the objectives of the problem are evaluated and returned to the master. The master will aggregate the different results.

In the multi-objective context, the scalability of this model is limited by the number of objectives and the number of sub-functions per objective. The scalability could be improved again if the different objective functions are simultaneously parallelized.

2.4 Hierarchical Combination of the Parallel Models

The three presented models for parallel MOEAs may be used in conjunction within a hierarchical structure [26]. The parallelism degree associated with this hybrid model is very important. Indeed, this hybrid model is very scalable; the degree of concurrency is $k * m * n$, where k is the number of MOEAs used, m is the size of the population, and n is the number of partitions or tasks associated to the evaluation of a single solution.

3. Parallel Implementation of MOEAs

Parallel implementation of MOEAs deals with the efficient mapping of a parallel model of MOEAs on a given parallel architecture.

3.1 Parallel Architectures

Parallel architectures are evolving quickly. The main criteria of parallel architectures, which will have an impact on the implementation of parallel MOEAs, are: memory sharing, homogeneity of resources, resource sharing by multiple users, scalability, and volatility. Those criteria will be used to analyze the different parallel models and their efficient implementation. A guideline is given for the efficient implementation of each parallel model of MOEAs according to each class of parallel architectures.

Shared memory/Distributed memory architectures: in shared memory parallel architectures, the processors are connected by a shared memory. There are different interconnection schemes for the network (e.g., bus, crossbar, multistage crossbar). This architecture is easy to program. Conventional operating systems and programming paradigms of sequential programming can be used. There is only one address space for data exchange but the programmer must take care of synchronization in memory access, such as the mutual exclusion in critical sections. This type of architecture has a poor scalability (from 2 to 128 processors in current technologies) and a higher cost. An example of such shared memory architectures are SMPs (Symmetric Multiprocessors) machines and multi-core processors.

In distributed memory architectures, each processor has its own memory. The processors are connected by a given interconnection network using different topologies (e.g., hypercube, 2D or 3D torus, fat-tree, multistage crossbars). This architecture is harder to program; data and/or tasks have to be explicitly distributed to processors. Exchanging information is also explicitly handled using message passing between nodes (synchronous or asynchronous communications). The cost of communication is not negligible and must be minimized to design an efficient parallel MOEA. However, this architecture has a good scalability in terms of the number of processors. In recent years, clusters of processors (COWs) became one of the most popular parallel distributed memory architectures. A good ratio between cost and performance is obtained with this class of architectures.

Homogeneous/Heterogenous parallel architectures: parallel architectures may be characterized by the homogeneity of the used processors, communication networks, operating systems, etc. For instance,

COWs are in general homogeneous parallel architectures. The proliferation of powerful workstations and fast communication networks have shown the emergence of heterogeneous networks of workstations (NOWs) as platforms for high performance computing. This type of architecture is present in any laboratory, company, campus, institution, etc. These parallel platforms are generally composed of an important number of owned heterogeneous workstations shared by many users.

Shared/Non shared parallel architectures: most massively parallel machines (MPP) and clusters of workstations (COW) are generally non shared by the applications. Indeed, at a given time, the processors composing those architectures are dedicated to the execution of a single application. NOWs constitute a low-cost hardware alternative to run parallel algorithms but are in general shared by multiple users and applications.

Local network (LAN)/Wide-area network (WAN): massively parallel machines, clusters and local networks of workstations may be considered as tightly coupled architectures. Large networks of workstations and grid computing platforms are loosely coupled and are affected by a higher cost of communication. During the last decade, *grid computing* systems have been largely deployed to provide high performance computing platforms. A computational grid is a scalable pool of heterogeneous and dynamic resources geographically distributed across multiple administrative domains and owned by different organizations [9]. Two types of Grids may be distinguished:

- **High-Performance Computing Grid** (HPC Grid): this grid interconnect supercomputers or clusters via a dedicated high-speed network. In general, this type of grid is non-shared by multiple users (at the level of processors).
- **Desktop Grid**: this class of grids is composed of numerous owned workstations connected via non dedicated network such as the internet. This grid is volatile and shared by multiple users and applications.

Volatile/Non volatile parallel architectures: desktop grids constitute an example of volatile parallel architectures. In a volatile parallel architecture, there is a dynamic temporal and spatial availability of resources. In a desktop grid or a large network of shared workstations, volatility is not an exception but a rule. Due to the large scale nature of the grid, the probability of resource failure is high. For instance, desktop grids have a faulty nature (e.g., reboot, shutdown, failure).

The following table 2 recapitulates the characteristics of the main parallel architectures according to the presented criteria. Those criteria will

be used to analyze the efficient implementation of the different parallel models of MOEAs.

Table 2: Characteristics of the main parallel architectures.

Criteria	Memory	Homogeneity	Sharing	Network	Volatility
SMP Multi-core	Shared	Hom	Yes or No	Local	No
COW	Distributed	Hom or Het	No	Local	No
NOW	Distributed	Het	Yes	Local	Yes
HPC Grid	Distributed	Het	No	Large	No
Desktop Grid	Distributed	Het	Yes	Large	Yes

Hom: Homogeneous, Het: Heterogeneous.

3.2 Dedicated Architectures

Dedicated hardware represents programmable hardware or specific architectures that can be designed or re-used to execute a parallel MOEA. The best known dedicated hardware is represented by Field Programmable Gate Arrays (FPGAs) and Graphical Processing Unit (GPU).

FPGAs are hardware devices that can be used to implement digital circuits by means of a programming process [31]. The use of the Xilinx's FPGAs to implement different MOEAs is more and more popular. The design and the prototyping of a FPGA-based hardware board to execute parallel MOEAs may restrict the design of some search components. However, for some specific challenging optimization problems with a high use rate such as in bioinformatics, dedicated hardware may be a good alternative.

GPU is a dedicated graphics rendering device for a workstation, personal computer, or game console. Recent GPUs are very efficient at manipulating computer graphics, and their parallel SIMD structure makes them more efficient than general-purpose CPUs for a range of complex algorithms [2]. The main companies producing GPUs are AMD and NVIDIA. The use of GPUs for an efficient implementation of MOEAs is a challenging issue [12, 13].

3.3 Parallel Programming Environments and Middlewares

The architecture of the target parallel machine strongly influences the choice of the parallel programming model to use. There are two main parallel programming paradigms: shared memory and message passing.

Two main alternatives exist to program shared memory architectures:

- **Multi-threading:** a thread may be viewed as a lightweight process. Different threads of the same process share some resources and the same address space. The main advantages of multi-threading are the fast context switch, the low resource usage, and the possible recovery between communication and computation. Each thread can be executed on a different processor or core. Multi-threaded programming may be used within libraries such as the standard Pthreads library [3] or programming languages such as Java threads [10].

- **Compiler directives:** one of the standard shared memory paradigms is OpenMP (Open Multi-Processing, `www.openmp.org`) and CUDA. It represents a set of compiler directives interfaced with the languages Fortran, C and C++ [4]. Those directives are integrated in a program to specify which sections of the program to be parallelized by the compiler.

Distributed memory parallel programming environments are based mainly on the three following paradigms:

- **Message passing:** message passing is probably the most widely used paradigm to program parallel architectures. Processes of a given parallel program communicate by exchanging messages in a synchronous or asynchronous way. The well known programming environments based on message passing are sockets and MPI (Message Passing Interface).

- **Remote Procedure Call:** Remote procedure call (RPC) represents a traditional way of programming parallel and distributed architectures. It allows a program to cause a procedure to execute on another processor.

- **Object oriented models:** as in sequential programming, parallel object oriented programming is a natural evolution of RPC. A classical example of such a model is Java RMI (Remote Method Invocation).

In the last decade, great work has been carried out on the development of grid middlewares. The Globus toolkit (`www.globus.org`) represents the *de facto* standard grid middleware. It supports the development of distributed service-oriented computing applications [20].

It is not easy to propose a guideline on which environment to use in programming a parallel MOEA. It will depend on the target architecture, the parallel model of MOEAs, and the user preferences. Some languages

are more system oriented such as C and C++. More portability is obtained with Java but the price is less efficiency. This tradeoff represents the classical efficiency/portability compromise. A Fortran programmer will be more comfortable with OpenMP. RPC models are more adapted to implement services. Condor represents an efficient and easy way to implement parallel programs on shared and volatile distributed architectures such as large networks of heterogeneous workstations and desktop grids, where fault tolerance is ensured by a checkpoint/recovery mechanism. The use of MPI within Globus is more or less adapted to high performance computing (HPC) grids. However, the user has to deal with complex mechanisms such as dynamic load balancing and fault-tolerance. Table 3 presents a guideline depending on the target parallel architecture.

Table 3: Parallel programming environments for different parallel architectures.

Architecture	**Examples of suitable programming environment**
SMP Multi-core	Multi-threading library within an operating system (e.g., Pthreads) Multi-threading within languages: Java OpenMP interfaced with C, C++ or Fortran
COW	Message passing library: MPI interfaced with C, C++, Fortran
Hybrid ccNUMA	MPI or Hybrid models: MPI/OpenMP, MPI/Multi-threading
NOW	Message passing library: MPI interfaced with C, C++, Fortran Condor or object models (JavaRMI)
HPC Grid	MPICH-G (Globus) or GridRPC models (Netsolve, Diet)
Desktop Grid	Condor-G or object models (Proactive)

3.4 Performance Evaluation

For sequential algorithms, the main performance measure is the execution time as a function of the input size. In parallel algorithms, this measure depends also on the number of processors and the characteristics of the parallel architecture. Hence, some classical performance indicators such as speedup and efficiency have been introduced to evaluate the scalability of parallel algorithms [11]. The scalability of a parallel algorithm measures its ability to achieve performance proportional to the number of processors.

The speed-up S_N is defined as the time T_1 it takes to complete a program with one processor divided by the time T_N it takes to complete

the same program with N processors

$$S_N = \frac{T_1}{T_N}.$$

One can use *wall-clock time* instead of *CPU time*. The CPU time is the time a processor spends in the execution of the program, and the wall-clock time is the time of the whole program including the input and output. Conceptually the speed-up is defined as the gain achieved by parallelizing a program. If $S_N > N$ (resp. $S_N = N$), a super-linear (resp. linear) speedup is obtained [25]. Mostly, a sub-linear speedup $S_N < N$ is obtained. This is due to the overhead of communication and synchronization costs. The case $S_N < 1$ means that the sequential time is smaller than the parallel time which is the worst case. This will be possible if the communication cost is much higher than the execution cost.

The efficiency E_N using N processors is defined as the speed-up S_N divided by the number of processors N.

$$E_N = \frac{S_N}{N}.$$

Conceptually the efficiency can be defined as how well N processors are used when the program is computed in parallel. An efficiency of 100% means that all of the processors are fully used all the time. For some large real-life applications, it is impossible to have the sequential time as the sequential execution of the algorithm cannot be performed. Then, the incremental efficiency E_{NM} may be used to evaluate the efficiency extending the number of processors from N to M processors.

$$E_{NM} = \frac{N \times E_N}{M \times E_M}.$$

Different definitions of speedup may be used depending on the definition of the sequential time reference T_1. Asking what is the best measure is useless; there is no global dominance between the different measures. The choice of a given definition depends on the objective of the performance evaluation analysis. Then, it is important to specify clearly the choice and the objective of the analysis.

The *absolute speedup* is used when the sequential time T_1 corresponds to the best known sequential time to solve the problem. Unlike other scientific domains such as numerical algebra where for some operations the best sequential algorithm is known, in MOEA search, it is difficult to identify the best sequential algorithm. So, the absolute speedup is rarely used. The *relative speedup* is used when the sequential time T_1 corresponds to the parallel program executed on a single processor.

Moreover, different stopping conditions may be used:

- **Fixed number of iterations:** this condition is the most used to evaluate the efficiency of a parallel MOEA. Using this definition, a superlinear speedup is possible $S_N > N$ [7]. This is due to the characteristics of the parallel architecture where there is more resources (e.g., size of main memory and cache) than in a single processor. For instance, the search memory of an MOEA executed on a single processor may be larger than the main memory of a single processor and then some swapping will be carried out, which represents an overhead in the sequential time. When using a parallel architecture, the whole memory of the MOEA may fit in the main memory of its processors, and then the memory swapping overhead will not occur.
- **Convergence to a set of solutions with a given quality:** this measure is interesting to evaluate the effectiveness of a parallel MOEA. It is only valid for parallel models of MOEAs based on the algorithmic-level, which alter the behavior of the sequential MOEA. A super-linear speedup is possible and is due to the characteristics of the parallel search. Indeed, the order of searching different regions of the search space may be different from sequential search. The sequences of visited solutions in parallel and sequential search are different. This is similar to the super-linear speedups obtained in exact search algorithms such as branch and bound [24].

Most of evolutionary algorithms are stochastic algorithms. When the stopping condition is based on the quality of the solution, one cannot use the speedup metric as defined previously. The original definition may be extended to the average speedup:

$$S_N = \frac{E(T_1)}{E(T_N)}.$$

The same *seed* for the generation of random numbers must be used for a more fair experimental performance evaluation. The speedup metrics have to be reformulated for heterogeneous architectures. The efficiency metric may be used for this class of architectures. Moreover, it can be used for shared parallel machines with multiple users.

3.5 Main Properties of Parallel MOEAs

The performance of a parallel MOEA on a given parallel architecture depends mainly on its *granularity*. The granularity of a parallel program

is the amount of computation performed between two communications. It computes the ratio between the computation time and the communication time. The three parallel models (algorithmic-level, iteration-level, solution-level) have a decreasing granularity from coarse-grained to fine-grained. The granularity indicator has an important impact on the speedup. The larger is the granularity the better is the obtained speedup.

The *degree of concurrency* of a parallel MOEA is represented by the maximum number of parallel processes at any time. This measure is independent from the target parallel architecture. It is an indication of the number of processors that can employed usefully by the parallel MOEA. Asynchronous communications and the recovery between computation and communication is also an important issue for a parallel efficient implementation. Indeed, most of the actual processors integrate different parallel elements such as ALU, FPU, GPU, DMA, etc. Most of the computing part takes part in cache. Hence, the RAM bus is often free and can be used by other elements such as the DMA. Hence, input/output operations can be recovered by computation tasks.

Scheduling the different tasks composing a parallel MOEA is another classical issue to deal with for their efficient implementation. Different scheduling strategies may be used depending on whether the number and the location of works (tasks, data) depend or not on the load state of the target machine:

- **Static scheduling**: this class represents parallel MOEAs in which both the number of tasks of the application and the location of work (tasks, data) are generated at compile time. Static scheduling is useful for homogeneous, and non shared and non volatile heterogeneous parallel architectures. Indeed, when there are noticeable load or power differences between processors, the search time of an iteration is derived by the maximum execution time over all processors, presumably on the most highly loaded processor or the least powerful processor. A significant number of tasks are often idle waiting for other tasks to complete their work.

- **Dynamic scheduling**: this class represents parallel MOEAs for which the number of tasks is fixed at compile time, but the location of work is determined and/or changed at run-time. The tasks are dynamically scheduled on the different processors of the parallel architecture. Dynamic load balancing is important for shared (multi-user) architectures, where the load of a given processor cannot be determined at compile time. Dynamic scheduling is also important for *irregular* parallel MOEAs in which the exe-

cution time cannot be predicted at compile time and varies during the search. For instance, this happens when the evaluation cost of the objective functions depends on the solution.

- **Adaptive scheduling**: parallel adaptive algorithms are parallel computations with a dynamically changing set of tasks. Tasks may be created or killed as a function of the load state of the parallel machine. A task is created automatically when a node becomes idle. When a node becomes busy, the task is killed. Adaptive load balancing is important for volatile architectures such as desktop grids.

For some parallel and distributed architectures such as shared networks of workstations and grids, fault tolerance is an important issue. Indeed, in volatile shared architectures and large-scale parallel architectures, the fault probability is relatively important. Checkpointing and recovery techniques constitute one answer to this problem. Application-level checkpointing is much more efficient than system-level checkpointing. Indeed, in system-level checkpointing, a checkpoint of the global state of a distributed application composed of a set of processes is carried out. In application-level checkpointing, only minimal information will be checkpointed (e.g., population of individuals, generation number). Compared to system-level checkpointing, a reduced cost is then obtained in terms of memory and time. Finally, security issues may be important for large-scale distributed architectures such as grids and Clouds (multi-domain administration, firewall, etc) and some specific applications such as medical and bioinformatics research applications of industrial concern [30].

3.6 Algorithmic-Level Parallel Model

Granularity: the algorithmic-level parallel model has the largest granularity. Indeed, the time for exchanging the information is in general much less than the computation time of a MOEA. There are relatively low communication requirements for this model. The more important is the frequency of exchange and the size of exchanged information, the smaller is the granularity. This parallel model is the most suited to large-scale distributed architectures over internet such as grids. Moreover, the trivial model with independent algorithms is convenient for low-speed networks of workstations over intranet. As there is no essential dependency and communication between the algorithms, the speedup is generally linear for this parallel model. The size of the data exchanged (for instance the number of Pareto solutions) will influence the granularity of the model. If the number of Pareto solutions is high

the communication cost will be exorbitant particularly on a large-scale parallel architectures such as grids.

For an efficient implementation, the frequency of exchange (resp. the size of the exchanged data) must be correlated to the latency (resp. bandwidth) of the communication network of the parallel architecture. To optimize the communication between processors, the exchange topology can be specified according to the interconnection network of the parallel architecture. The specification of the different parameters associated with the blind or intelligent migration decision criterion (migration frequency/probability and improvement threshold) is particularly crucial on a computational grid. Indeed, due to the heterogeneous nature of computational grids these parameters must be specified for each MOEA in accordance with the machine it is hosted on.

Scalability: the degree of concurrency of the algorithmic-level parallel model is limited by the number of MOEAs involved in solving the problem. In theory, there is no limit. However, in practice, it is limited by the owned resources of the target parallel architectures, and also by the effectiveness aspect of using a large number of MOEAs.

Synchronous versus asynchronous communications: the implementation of the algorithmic-level model is either *asynchronous* or *synchronous*. The asynchronous mode associates with each MOEA an exchange decision criterion, which is evaluated at each iteration of the MOEA from the state of its memory. If the criterion is satisfied, the MOEA communicates with its neighbours. The exchange requests are managed by the destination MOEAs within an undetermined delay. The reception and integration of the received information is thus performed during the next iterations. However, in a computational grid context, due to the material and/or software heterogeneity issue, the MOEAs could be at different evolution stages leading to the *non-effect* and/or *super-solution* problem. For instance, the arrival of poor solutions at a very advanced stage will not bring any contribution as these solutions will likely not be integrated. In the opposite situation, the cooperation will lead to premature convergence.

From another point of view, as it is non-blocking, the model is more efficient and fault tolerant to such a degree a threshold of wasted exchanges is not exceeded. In the synchronous mode, the MOEAs perform a synchronization operation at a predefined iteration by exchanging some data. Such operation guarantees that the MOEAs are at the same evolution stage, and so prevents the non-effect and super-solution problem quoted before. However, in heterogeneous parallel architectures, the synchronous mode is less efficient in term of consumed CPU time. Indeed, the evolution process is often hanging on powerful machines waiting

the less powerful ones to complete their computation. The synchronous model is also not fault tolerant as a fault of a single MOEA implies the blocking of the whole model in a volatile environment. Then, the synchronous mode is globally less efficient on a computational grid.

Asynchronous communication is more efficient than synchronous communication for shared architectures such as NOWs and desktop grids (e.g., multiple users, multiple applications). Indeed, as the load of networks and processors is not homogeneous, the use of synchronous communication will degrade the performances of the whole system. The least powerful machine will determine the performance.

On a volatile computational grid, it is difficult to efficiently maintain topologies such as rings and torus. Indeed, the disappearance of a given node (i.e., MOEA) requires a dynamic reconfiguration of the topology. Such reconfiguration is costly and makes the migration process inefficient. Designing a cooperation between a set of MOEAs without any topology may be considered. For instance, a communication scheme in which the target MOEA is selected randomly is more efficient for volatile architecture such as desktop grids. Many experimental results show that such topology allows a significant improvement of the robustness and quality of solutions. The random topology is therefore thinkable and even commendable in a computational grid context.

Scheduling: concerning the scheduling aspect, in the algorithmic-level parallel model the tasks correspond to MOEAs. Hence, the different scheduling strategies will differ as follows:

- Static scheduling: the number of MOEAs is constant and correlated to the number of processors of the parallel machine. A static mapping between the MOEAs and the processors is realized. The localization of MOEAs will not change during the search.

- Dynamic scheduling: MOEAs are dynamically scheduled on the different processors of the parallel architecture. Hence, the migration of MOEAs during the search between different machines may happen.

- Adaptive scheduling: the number of MOEAs involved into the search will vary dynamically. For example, when a machine becomes idle, a new MOEA is launched to perform a new search. When a machine becomes busy or faulty, the associated MOEA is stopped.

Fault-tolerance: the memory state of the algorithmic-level parallel model required for the checkpointing mechanism is composed of the

memory of each MOEA and the information being migrated (i.e., population, archive, generation number).

3.7 Iteration-Level Parallel Model

Granularity: a medium granularity is associated to the iteration-level parallel model. The ratio between the evaluation of a partition and the communication cost of a partition determines the granularity. This parallel model is then efficient if the evaluation of a solution is time-consuming and/or there are a large number of candidate solutions to evaluate. The granularity will depend on the number of solutions in each sub-population.

Scalability: the degree of concurrency of this model is limited by the size of the population. The use of large populations will increase the scalability of this parallel model.

Synchronous versus asynchronous communications: introducing asynchronism in the iteration-level parallel model will increase the efficiency of parallel MOEAs. In the iteration-level parallel model, asynchronous communications are related to the asynchronous evaluation of partitions and construction of solutions. Unfortunately, this model is more or less synchronous. Asynchronous evaluation is more efficient for heterogeneous or shared or volatile parallel architectures. Moreover, asynchronism is necessary for optimization problems where the computation cost of the objective function (and constraints) depends on the solution and different solutions may have different evaluation cost.

Asynchronism may be introduced by relaxing the synchronization constraints. For instance, steady-state algorithms may be used in the reproduction phase [6].

The two main advantages of the asynchronous model over the synchronous model are fault tolerance and robustness if the fitness computation takes very different computations time. Whereas some time-out detection can be used to address the former issue, the latter one can be partially overcome if the grain is set to very small values, as individuals will be sent out for evaluations upon request of the workers. Therefore, the model is blocking and, thus, less efficient on a heterogeneous computational grid. Moreover, as the model is not fault tolerant, the disappearance of an evaluating agent requires the redistribution of its individuals to other agents. As a consequence, it is essential to store all the solutions not yet evaluated. The scalability of the model is limited to the size of the population.

Scheduling: in the iteration-level parallel model, tasks correspond to the construction/evaluation of a set of solutions. Hence, the different scheduling strategies will differ as follows:

- Static scheduling: here, a static partitioning of the population is applied. For instance, the population is decomposed into equal size partitions depending on the number of processors of the parallel homogeneous non-shared machine. A static mapping between the partitions and the processors is realized. For a heterogeneous non-shared machine, the size of each partition must be initialized according to the performance of the processors. The static scheduling strategy is not efficient for variable computational costs of equal partitions. This happens for optimization problems where different costs are associated to the evaluation of solutions. For instance, in genetic programming individuals may widely vary in size and complexity. This makes a static scheduling of the parallel evaluation of the individuals not efficient.

- Dynamic scheduling: a static partitioning is applied but a dynamic migration of tasks can be carried out depending on the varying load of processors. The number of tasks generated may be equal to the size of the population. Many tasks may be mapped on the same processor. Hence, more flexibility is obtained for the scheduling algorithm. For instance, the approach based on the master-workers cycle stealing may be applied. To each worker is first allocated a small number of solutions. Once it has performed its iterations the worker requests from the master additional solutions. All the workers are stopped once the final result is returned. Faster and less loaded processors handle more solutions than the others. This approach allows to reduce the execution time compared to the static one.

- Adaptive scheduling: the objective in this model is to adapt the number of partitions generated to the load of the target architecture. More efficient scheduling strategies are obtained for shared, volatile and heterogeneous parallel architectures such as desktop grids.

Fault-tolerance: the memory of the iteration-level parallel model required for the checkpointing mechanism is composed of different partitions. The partitions are composed of a set of (partial) solutions and their associated objective values.

3.8 Solution-Level Parallel Model

Granularity: this parallel model has a fine granularity. There is a relatively high communication requirements for this model. In the functional decomposition parallel model, the granularity will depend on the ratio between the evaluation cost of the sub-functions and the communication cost of a solution. In the data decomposition parallel model, it depends on the ratio between the evaluation of a data partition and its communication cost.

The fine granularity of this model makes it less suitable for large-scale distributed architectures where the communication cost (in terms of latency and/or bandwidth) is relatively important, such as in grid computing systems. Indeed, its implementation is often restricted to clusters or network of workstations or shared memory machines.

Scalability: the degree of concurrency of this parallel model is limited by the number of sub-functions or data partitions. Although its scalability is limited, the use of the solution-level parallel model in conjunction with the two other parallel models enables to extend the scalability of a parallel MOEA.

Synchronous versus asynchronous communications: the implementation of the solution-level parallel model is always synchronous following a master-workers paradigm. Indeed, the master must wait for all partial results to compute the global value of the objective functions. The execution time T will be bounded by the maximum time T_i of the different tasks. An exception occurs for hard-constrained optimization problems, where feasibility of the solution is first tested. The master terminates the computations as soon as a given task detects that the solution does not satisfy a given hard constraint. Due to its heavy synchronization steps, this parallel model is worth applying to problems in which the calculations required at each iteration are time consuming. The relative speedup may be approximated as follows:

$$S_n = \frac{T}{\alpha + T/n},$$

where α is the communication cost.

Scheduling: in the solution-level parallel model, tasks correspond to sub-functions in the functional decomposition and to data partitions in the data decomposition model. Hence, the different scheduling strategies will differ as follows:

- Static scheduling: usually, the sub-functions or data are decomposed into equal size partitions depending on the number of processors of the parallel machine. A static mapping between the

sub-functions (or data partitions) and the processors is applied. As for the other parallel models, this static scheme is efficient for parallel homogeneous non-shared machines. For a heterogeneous non-shared machine, the size of each partition in terms of sub-functions or data must be initialized according to the performance of the processors.

- Dynamic scheduling: dynamic load balancing will be necessary for shared parallel architectures or variable costs for the associated sub-functions or data partitions. Dynamic load balancing may be easily achieved by evenly distributing at run-time the sub-functions or the data among the processors. In optimization problems, where the computing cost of the sub-functions is unpredictable, dynamic load balancing is necessary. Indeed, a static scheduling cannot be efficient because there is no appropriate estimation of the task costs (i.e., unpredictable cost).
- Adaptive scheduling: in adaptive scheduling, the number of sub-functions or data partitions generated is adapted to the load of the target architecture. More efficient scheduling strategies are obtained for shared, volatile and heterogeneous parallel architectures such as desktop grids.

Fault-tolerance: the memory of the solution-level parallel model required for the checkpointing mechanism is straightforward. It is composed of the solution(s) and their partial objective value calculations.

Depending on the target parallel architecture, table 4 presents a general guideline for the efficient implementation of the different parallel models of MOEAs. For each parallel model (algorithmic-level, iteration-level, solution-level), the table shows its characteristics according to the outlined criteria (granularity, scalability, asynchronism, scheduling and fault-tolerance).

4. Conclusions and Perspectives

Parallel and distributed computing can be used in the design and implementation of MOEAs to speedup the search, to improve the quality of the obtained solutions, to improve the robustness, and to solve large scale problems. The clear separation between parallel design and parallel implementation aspects of MOEAs is important to analyze parallel MOEAs. The most important lessons of this paper can be summarized as follows:

- In terms of parallel design, the different parallel models for MOEAs have been unified. Three hierarchical parallel models have been

Table 4: Efficient implementation of parallel MOEAs according to some performance metrics and used strategies.

Property	**Algorithmic-level**	**Iteration-level**	**Solution-level**
Granularity	Coarse (Frequency of exchange, size of information)	Medium (Nb. of solutions per partition)	Fine (Eval. sub-functions, eval. data partitions)
Scalability	Number of MOEAs	Neighborhood size, populations size	Nb. of sub-functions, nb. data partitions
Asynchronism	High (Information exchange)	Moderate (Eval. of solutions)	Exceptional (Feasibility test)
Scheduling and Fault-tolerance	MOEA	Solution(s)	Partial solution(s)

extracted: algorithmic-level, iteration-level and solution-level parallel models.

- In terms of parallel implementation, the question of an efficient mapping of a parallel model of MOEAs on a given parallel architecture and programming environment (i.e., language, library, middleware) is handled. The focus was made on the key criteria of parallel architectures that influence the efficiency of an implementation of parallel MOEAs.

One of the perspectives in the coming years is to achieve Exascale performance. The emergence of heterogeneous platforms composed of multicore chips and many-core chips technologies will speedup the achievement of this goal. In terms of programming models, cloud computing will become an important alternative to traditional high performance computing for the development of large-scale MOEAs that harness massive computational resources. This is a great challenge as nowadays cloud frameworks for parallel MOEAs are just emerging.

In the future design of high-performance computers, the ratio between power and performance will be increasingly important. The power represents the electrical power consumption of the computer. An excess in power consumption uses unnecessary energy, generates waste heat and decreases reliability. Very few vendors of high-performance architecture publicize the power consumption data compared to the performance data.

In terms of target optimization problems, parallel MOEAs constitute unavoidable approaches to solve large scale real-life challenging problems (e.g., engineering design, data mining). They are also an important alternative to solve dynamic and uncertain optimization MOPs, in which

the complexities in terms of time and quality are more difficult to handle by traditional sequential approaches. Moreover, parallel models for MOPs with uncertainty have to be deeply investigated.

Acknowledgement: This work is part of a project that has received funding from the *European Unions Horizon 2020 research and innovation program* under grant agreement No 692286.

References

[1] M. Basseur, F. Seynhaeve, and E.-G. Talbi. Adaptive mechanisms for multi-objective evolutionary algorithms. *Proceedings of the Congress on Engineering in System Application (CESA)*, pages 72–86, 2003.

[2] A. R. Brodtkorb, T. R. Hagen, and M. L. Sætra. Graphics Processing Unit (GPU) Programming Strategies and Trends in GPU Computing. *Journal of Parallel and Distributed Computing*, 73(1):4–13, 2013.

[3] D. R. Butenhof. *Programming with POSIX threads*. Addison-Wesley, 1997.

[4] B. Chapman, G. Jost, R. van der Pas, and D. J. Kuck. *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2007.

[5] C. A. Coello and M. Reyes. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. *Lecture Notes in Artificial Intelligence*, 2972:688–697, 2004.

[6] M. Depolli, R. Trobec, and B. Filipič. Asynchronous master-slave parallelization of differential evolution for multi-objective optimization. *Evolutionary Computation*, 21(2):261–291, 2013.

[7] B. Dorronsoro, G. Danoy, A. J. Nebro, and P. Bouvry. Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution. *Computers & Operations Research*, 40(6):1552–1563, 2013.

[8] S. Duarte and B. Barán. Multiobjective network design optimisation using parallel evolutionary algorithms. *Proceedings of the XXVII Conferencia Latinoamericana de Informática (CLEI)*, 2001.

[9] I. Foster and C. Kesselman (Eds.) *The grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann, San Fransisco, 1999.

[10] P. Hyde. *Java thread programming*. Sams, 1999.

[11] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: Design and analysis of algorithms*. Addison Wesley, 1994.

[12] Z. Li, Y. Bian, R. Zhao, and J. Cheng. A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU. *Lecture Notes in Computer Science*, 8084:111–125, 2013.

[13] T. V. Luong, E. Taillard, N. Melab, and E-G. Talbi. Parallelization Strategies for Hybrid Metaheuristics Using a Single GPU and Multi-core Resources. *Lecture Notes in Computer Science*, 7492:368–377, 2012.

[14] A. M. Mora, P. García-Sánchez, J. J. Merelo Guervós, and P. A. Castillo. Pareto-based multi-colony multi-objective ant colony optimization algorithms: an island model proposal. *Soft Computing*, 17(7):1175–1207, 2013.

[15] S. Mostaghim, J. Branke, and H. Schmeck. Multi-objective particle swarm optimization on computer grids. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 869–875, 2007.

[16] A. J. Nebro, F. Luna, E.-G. Talbi, and E. Alba. Parallel multiobjective optimization. In E. Alba (Ed.) *Parallel metaheuristics*, Wiley, 2005, pages 371–394.

[17] S. Nesmachnow. Parallel multiobjective evolutionary algorithms for batch scheduling in heterogeneous computing and grid systems. *Computational Optimization and Applications*, 55(2):515–544, 2013.

[18] K. E. Parsopoulos, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis. Vector evaluated differential evolution for multiobjective optimization. *Proceedings of the IEEE 2004 Congress on Evolutionary Computation (CEC'04)*, 2004.

[19] J. Rowe, K. Vinsen, and N. Marvin. Parallel GAs for multiobjective functions. *Proceedings of the 2nd Nordic Workshop on Genetic Algorithms and Their Applications (2NWGA)*, pages 61–70, 1996.

[20] B. Sotomayor and L. Childers. *Globus toolkit 4: Programming Java services.* Morgan Kaufmann, 2005.

[21] N. Srinivas and K. Deb. Multiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.

[22] T. J. Stanley and T. Mudge. A parallel genetic algorithm for multi-objetive microprocessor design. *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 597–604, 1995.

[23] R. Szmit and A. Barak. Evolution Strategies for a Parallel Multi-Objective Genetic Algorithm. P*roceedings of the Genetic and Evolutionary Computation Conference*, pages 227–234, 2000.

[24] E.-G. Talbi. *Parallel combinatorial optimization.* Wiley, 2006.

[25] E.-G. Talbi and P. Bessière. Superlinear speedup of a parallel genetic algorithm on the SuperNode. *SIAM News*, 24(4):12–27, 1991.

[26] E.-G. Talbi, S. Cahon, and N. Melab. Designing cellular networks using a parallel hybrid metaheuristic on the computational grid. *Computer Communications*, 30(4):698–713, 2007.

[27] E.-G. Talbi, S. Mostaghim, H. Ishibushi, T. Okabe, G. Rudolph, and C.C. Coello. Parallel approaches for multi-objective optimization. *Lecture Notes in Computer Science*, 5252:349–372, 2008.

[28] F. de Toro, J. Ortega, E. Ros, S. Mota, B. Paechter, and J.M. Martín. PSFGA: Parallel processing and evolutionary computation for multiobjective optimisation. *Parallel Computing*, 30(5-6):721–739, 2004.

[29] D. A. van Veldhuizen, J. B. Zydallis, and G. B. Lamont. Considerations in engineering parallel multi-objective evolutionary algorithms. *IEEE Transasctions on Evolutionary Computation*, 7(2):144–173, 2003.

[30] G. Yao, Y. Ding, Y. Jin, and K. Hao. Endocrine-based coevolutionary multiswarm for multi-objective workflow scheduling in a cloud system. *Soft Computing*, 1–14, 2016.

[31] R. Zeidman. *Designing with FPGAs and CPLDs.* Elsevier, 2002.

II

THEORY AND ALGORITHMS

ARTIFICIAL BEE COLONY OPTIMIZATION APPROACH TO DEVELOP STRATEGIES FOR THE ITERATED PRISONER'S DILEMMA

Manousos Rigakis, Dimitra Trachanatzi, Magdalene Marinaki, Yannis Marinakis

School of Production Engineering and Management, Technical University of Crete, Greece

mrigakis@isc.tuc.gr, dtrachanatzi@isc.tuc.gr, magda@dssl.tuc.gr, marinakis@ergasya.tuc.gr

Abstract This study proposes a binary Artificial Bee Colony (ABC) approach to develop game strategies for Iterated Prisoner's Dilemma (IPD). To determine the quality of the evolved strategies, a comparison is made between this binary ABC approach, several known man-made strategies and strategies developed by Particle Swarm Optimization (PSO) algorithm. In this paper, we examine the suitability of the nature inspired ABC algorithm to generate strategies for IPD which has not been investigated before. In general, the ABC algorithm provides better strategies against PSO and benchmark strategies.

Keywords: Artificial bee colony, Iterated prisoner's dilemma, Particle swarm optimization.

1. Introduction

The Prisoner's Dilemma (PD) is a well-known game of strategy in several sciences such as economics [9], biology [6], game theory, computer science and political science. Originally, it was proposed by Merril Flood and Melvin Dresher in 1950 [7] as a non-cooperative pair game and later Albert W. Tucker [15] characterized it as PD. In 1944 the theory of the two player non-cooperative game was introduced by Von Neumann and Morgenstern formalizing the mathematical base in the field of game theory [12]. This game is useful to demonstrate the evolution of cooperative behavior. The main method to study PD is proposed by Axelrod in 1987 [2] using Genetic Algorithms (GAs). Darwen and Yao followed Axelrod's work to evolve strategies for the IPD using co-evolution [5]. They also extended their research in a variation of IPD where players

have a range of intermediate choices [3, 4]. Since then, some work has been done at the study of IPD using nature inspired algorithms. Tekol and Acan in 2003 applied Ant Colony Optimization (ACO) to develop robust game strategies [14] and in 2005 Franken and Engelbrecht used Particle Swarm Optimization (PSO) to approach IPD [8].

In this paper, we examine the suitability of the nature inspired artificial bee colony algorithm [10] to evolve strategies for the iterated prisoner's dilemma. Specifically, our IPD approach uses the binary ABC algorithm to generate a bit string in order to produce a complete playing strategy. Overall the strategy of the ABC approach plays against man-made strategies and against PSO-players. This paper includes the following sections. In Section 2 we give the necessary background about the prisoner's dilemma. Section 3 includes the description of our approach, the definition of the benchmark strategies used, the main steps of our algorithm and a brief analysis of the PSO algorithm. Section 4 contains experimental results and in Section 5 conclusions and future work are given.

2. Prisoner's Dilemma

The prisoner's dilemma is referred to a non-zero-sum, a non-cooperative game, named by Albert W. Tucker in 1950 and is one of the most famous problems in game theory. It is applied to many cases of two conflicted players. The non-zero-sum term implies that when one player wins, the loss of the other is not a necessity. In addition, the non-cooperative implies that players have no way to communicate with each other prior to the game, they have no knowledge of the others' decision of historic behavior and they are not able of making any kind of agreement with the other prisoner. A general description [15] is given in the following. Two persons are jointly charged with a crime and held in separate cells. They have two possible choices: the prisoner can cooperate (C) meaning that he/she will keep quite and keep to the pre-planned story made with the other prisoner before they have both been arrested or the prisoner can defect (D) meaning that she/he will reach an agreement with the police and accuse the other prisoner. Thus, the prisoner that defects is free to go if the other prisoner decides to cooperate and the prisoner that cooperates will be jailed for m years. In case that both of them decide to cooperate, then, they will both be jailed for n years (where $n < m$). Finally, if both of them decide to defect, they will both be jailed for r years (where $n < r < m$). In order to model the game in a matrix form, the payoff matrix is used. The payoff that every player gains according to his/her choice is given in Table 1 [2]. The first value

Table 1: Payoff matrix for prisoner's dilemma

		Player II	
		Cooperate	Defect
Player I	Cooperate	R,R	S,T
	Defect	T,S	P,P

mentioned inside the cells refers to the payoff of the player I and the second to the player II. The payoffs [2] included in Table 1 have a code letter according to the players' action. The letter R denotes the payoff if both players cooperate as a reward, S expresses the sucker's payoff, in case of cooperation against defection. T is the temptation payoff to defect against a cooperating opponent and P denotes the punishment payoff if both decide to defect. The prisoners dilemma is defined by the following inequalities on the value of S, P, R, T.

$$T > R > P > S \tag{1}$$

$$2R > S + T \tag{2}$$

This first constraint is a classification of the payoffs. The temptation payoff is the highest and the suckers payoff is the lowest. It also ensures that parallel cooperation is more profitable than parallel defection. The second constraint ensures that mutual cooperation of the players is more profitable than one player's defection against cooperation of the other and backwards. For this study, the numerical payoff of each decision is given in Table 2 which is same to the payoff matrix used by Axelrod [2]. This general description is actually aligned with the one-shot prisoner's

Table 2: Payoff matrix used for this paper

		Player II	
		Cooperate	Defect
Player I	Cooperate	3,3	0,5
	Defect	5,0	1,1

dilemma, meaning that each prisoner has only one opportunity to decide his action: cooperation or defection. In the way that the game is constructed, the most profitable solution for a player is to defect, irrelevant to his opponent strategy. Two scenarios can emerge: to gain the biggest payoff T if the other player cooperates or to receive a smaller

payoff P but equal to the payoff of the other player. This way the risk of receiving the lowest payoff S, as a victim of his opponents confession, is avoided. This leaves behind the option of mutual cooperation R, a strategy more profitable than mutual defection. In order to give the player the chance of choosing this strategy, Axelrod proposed the Iterated Prisoner's Dilemma IPD [1]. The iterated prisoner's dilemma is actually a sequence of repeated prisoner's dilemma games. The number of iterations must be unknown to both players. This hypothesis ensures that both players will not be trapped into a repeated mutual defection. To understand this situation, we have to assume that a known number of iterations transforms the game into a one-shot prisoner's dilemma repeated for every iteration. Thus, if the player knows which is the last iteration and reasonably defects on that iteration and the same logic is followed by his opponent, then, the same phenomenon will be carried out as we look back to the first iteration. Several versions of IPD exist, depended by the different ways of studying them. Since 1980, when Axelrod used Genetic Algorithms (GAs) to generate strategies for the IPD, little work has been done in the implementation of evolution algorithms to the problem [8].

3. Bees Play Prisoner's Dilemma

3.1 The Proposed ABC Approach

Artificial bee colony algorithm is based on the intelligent behavior of bee swarms and is mainly applied to continuous time optimization problems. ABC algorithm has been proposed by Karaboga and Basturk in 2008 [10] and simulates the waggle dance of the bees in their effort to find food. In the ABC algorithm, the colony consists of three groups of bees: the employed, the onlookers and the scout bees. Every food source is referred to only one employed bee, thus, the fleet of the employed bees to be used is equal to the amount of the food sources. In this study, we create a set of solutions as we generate a set of random players equal to the employed bees. For the iterated prisoner's dilemma the value of the fitness function is actually the payoff of each player. Our goal is to estimate the maximum payoff that a player achieves while playing with all the other random players. Every player i, who is generated, has a decision vector x_{ij} that contains binary values, $x_{ij} \in [0, 1]$. Every element of the decision vector describes the player's strategy on every game. The number of the played games j is the length of the vector. When $x_{ij} = 1$, player i cooperates and when $x_{ij} = 0$ player

i defects. For example, a decision vector for $j = 8$ is given below.

Decision Vector	*0*	*1*	*0*	*1*	*0*	*0*	*1*	*1*

As we mentioned above, we have a definite number of N players and N decision vectors. Without exception every player faces all others and compares the values of the decision vectors. In this way the payoff matrix is created containing all payoffs the players have achieved from all the games. The payoff matrix is used to calculate the probability that the onlooker bees visit the food sources by using roulette wheel selection method. The necessary probability $P(i)$ is calculated by the following equation:

$$P_i = \frac{F(i)}{F(N)} = \frac{F(i)}{\sum_{k=1}^{N} F(k)}, \tag{3}$$

where $F(i)$ is the payoff of player i, and $F(N)$ is the sum payoff of all players N. We are able to conclude that more onlookers are placed to players with high payoff in order to improve them. Considering that ABC algorithm was developed for problems with continuous-valued variables $y_{ij} \in R$, the decision vector of each player (discrete values) must be converted to a continuous-valued vector before proceed with the method. The function used for this transformation is the following:

$$sig(x_{ij}) = \frac{1}{1 + exp(-x_{ij})}. \tag{4}$$

After the transformation of the decision vectors we apply the equation (5) of the ABC algorithm that produces new food sources. To be exact, the equation differs the decision vectors of the players that have been selected by onlooker bees in pursuance of creating competitive players. The equation that gives a new food source given by Karaboga and Basturk [10] is the following:

$$y_{ij}(t+1) = y_{ij}(t) + \phi(y_{ij}(t) - y_{kj}(t)). \tag{5}$$

The y_{ij} is the decision vector of player i, ϕ is random number between $(0, 1)$, y_{kj} is the decision vector of a random player k such as $k \neq i$ and $k \in N$ and t is the current iteration. An essential point of the procedure is the number of onlooker bees that are allocated to a single food source to one player. In case of more than one bees, the new decision vectors y_{ij} are produced using equation (5) as many times as the number of bees allocated to this food source using different ϕ and different k each time. Thus, q new vectors are actually generated (q is equal to the number of onlooker bees). If a player has not been visited by any onlooker bee, then, his decision vector remains as previous. As we have to deal with a

combinatorial optimization problem, it is necessary to convert the newly created decision vectors y_{ij}, to binary values according to the following rule:

$$x_{ij}(t+1) = \begin{cases} 1, & \text{if } \phi < y_{ij}(t+1) \\ 0, & \text{if } \phi \geq y_{ij}(t+1) \end{cases} \tag{6}$$

When the conversion has been completed, $N + q$ decision vectors have been created. Summarizing, $N + q$ players are ready to compete with each other at prisoner's dilemma, as we described earlier. When all the games are completed, the payoffs are calculated. For each individual player the decision vector with maximum payoff is selected and saved into the memory. Finally, only one of the N players emerges. This player has the best decision vector meaning that he has achieved the maximum payoff against all the others.

We have presented a simulation study of artificial bee colony algorithm for solving PD that is practically without memory. Summarizing the context of the above paragraphs, the initial solution strategies are randomly created in every simulation irrelevant to the benchmark strategies (Section 3.2) and to the PSO evolved strategy (Section 3.3) that our player has to face. In order to improve our player's performance, the algorithm is alternated in a way to memorize the decision vector that maximizes the player's payoff after competing with each one of the strategies. Thus, as the iterations go on, the algorithm includes our player's best strategies of the previous iteration as a starting point of the next iteration. That gives advantage to his generated strategies. The pseudo-code used is given in Algorithm 1.

3.2 Benchmark Strategies

In order to picture our experimental work, some of the man-made strategies that participated in Axelrod's experiments, have been chosen. Additionally a randomly chosen player has been used, as well as players who have been generated by a PSO algorithm in order to compare those two algorithms. The man-made strategies [2] are: **Random**: This player has a random binary decision vector and he is the most unpredicted opponent. **Always Cooperate (AC)**: This is the most innocent strategy, because that player cooperates on every move. **Pavlov**: This player repeats the previous played move if that move was beneficial or plays the opposite if that move was unproductive. **Tit-for-tat (TFT)**: In this strategy proposed by Anatol Rapoport [13], the player begins with cooperation but he continues by imitating the last move of his opponent. **Evil tit-for-tat (ETFT)**: In this strategy, the player be-

Algorithm 1 Artificial Bee Colony Algorithm

1: Define number of employed bees (collection of players) (N)
2: Define number of onlooker bees (T) and number of scout bees (S)
3: Define number of executions (W), iterations (L) and games (M)
4: ***Initialization***
5: Randomly create decision vectors (x)
6: Pairing each player with one employed bee
7: Calculate the payoff for each player according Table 2 by playing all against all
8: ***Main Phase***
9: **while** the maximum number of iterations has not been reached **do**
10: Return employed bees to the hive
11: Transform the decision vector to a continued-valued with eq. (4)
12: Calculate P_i and place the onlooker bees using eq. (3)
13: Create new decision vectors using eq. (5)
14: All players compete with each other
15: Compare the payoffs and save the personal best decision vector
16: **end while**
17: Save the decision vector that gives maximum payoff of all the players
18: Best player plays M games against each of the 5 benchmark strategies (section 3.2)
19: Return to step 4 until W executions have been completed
20: When all executions have been completed, W players have been generated with ABC algorithm.

gins with defection but he continues by imitating the last move of his opponent.

3.3 Particle Swarm Optimization Algorithm

In order to compare the quality of our solution through ABC algorithm, we evolve strategies with PSO. Alike to ABC, PSO is an evolutionary technique and it is inspired by the behavior of bird flocks and swarming theory. Originally, it was introduced by Kennedy and Eberhart [11] for optimizing continuous nonlinear functions. The algorithm simulates the movement of a swarm (population) consisting of particles (number of players). The swarm moves through a solution space and each particle represents a feasible solution to the optimization problem. Each particle associates with a value of the objective function being optimized (payoff). Furthermore, velocity is, also, assigned to each particle in order to direct the movement towards better solutions (positions).

In the PSO algorithm, each particle knows its previous best solution and the best solution of the whole swarm [8]. Thus, particles modify their positions towards their personal best positions and the global best position of the whole swarm according to the following equations:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \tag{7}$$

$$v_{ij}(t+1) = v_{ij}(t) + c_1 * \phi_1(pbest_{ij} - x_{ij}(t)) + c_2 * \phi_2(gbest_j - x_{ij}(t)). \tag{8}$$

The description of the equations (7), (8) is relevant to our approach. Thus, the equation (7) represents the new decision vector of player i, t is the current iteration, c_1 and c_2 are velocity variables ($c_1 = c_2 = 2$) and ϕ_1, ϕ_2 are random numbers in the interval (0,1). Particle's personal best obtained value (payoff) is denoted by *pbest* and *gbest* denotes the best obtained value from all the particles in the swarm (the best payoff from all the players). PSO initially was proposed for continuous-valued variables, thus, it is necessary to convert the binary-valued decision vectors $x_{ij}(t)$ to continuous values and vice-versa. For this purpose, we use equations (4) and (6) as they are used in the ABC approach (section 3.1).

4. Experimental Results

In this section, a brief analysis of our experiments is presented. The figures below show the achieved payoffs (in the vertical axis) of ABC versus benchmark strategies for 20 different executions (players). The achieved payoff is the total payoff in en execution.

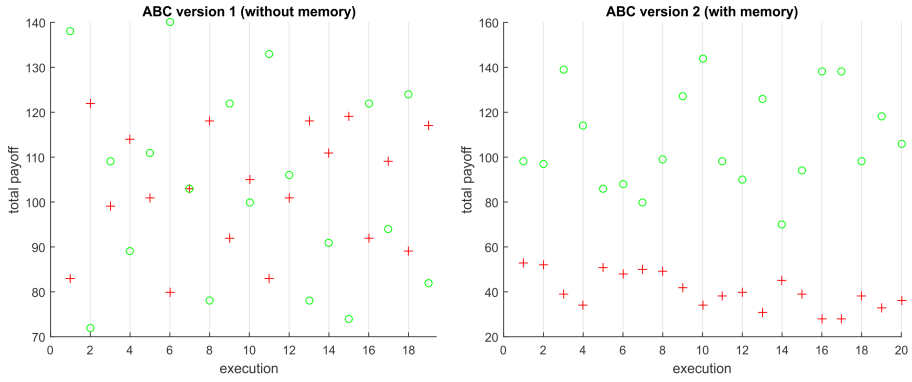


Figure 1: ABC vs RANDOM (ABC: cross, RANDOM: circle)

Figure 1 shows that version 1 of our algorithm (ABC version 1 (without memory)) against random strategies has unpredictable behavior de-

spite that this version generates competitive players. The memory that version 2 provides is meaningless cause of the irregularity of the opponent strategies.

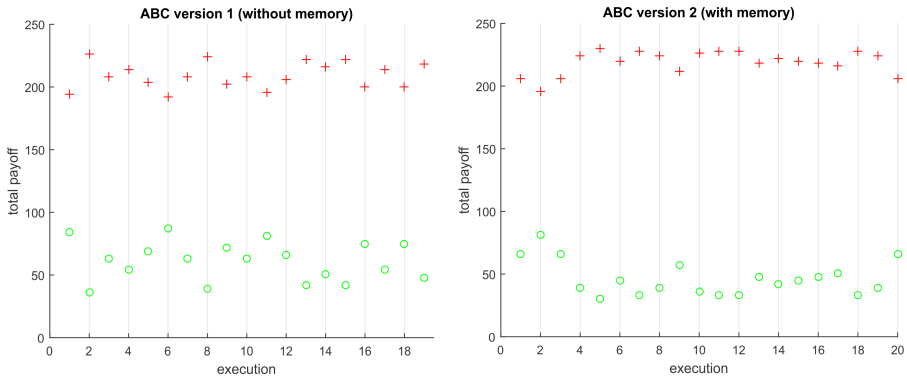


Figure 2: ABC vs AC (ABC: cross, AC: circle)

It is obvious from Fig. 2 that the AC strategy is ineffective due to the PD's formulation as our player achieves higher or equivalent to his AC-opponent's payoff at every game. Thus, both our versions always win the AC strategy.

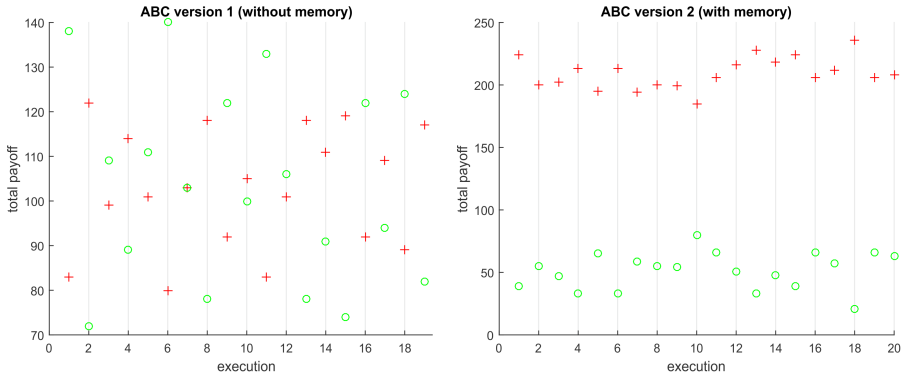


Figure 3: ABC vs PAVLOV (ABC: cross, PAVLOV: circle)

The generated strategies by version 1 of our algorithm are competitive against Pavlov's strategies. Version 2 of ABC performs better and constantly generates strategies able to win Pavlov's strategies as Fig. 3 shows.

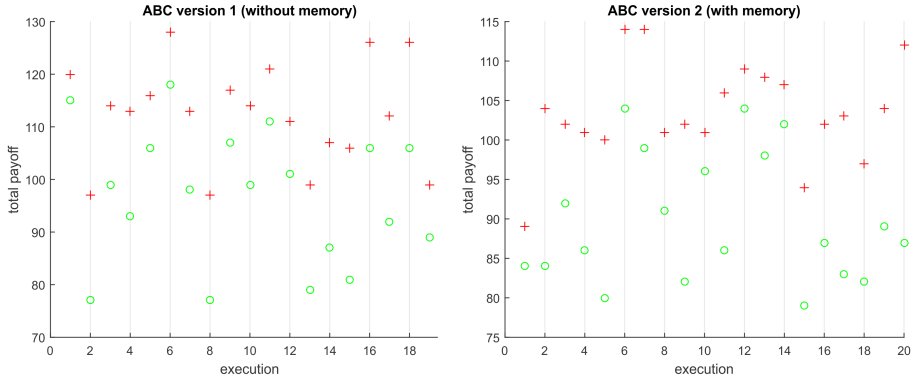


Figure 4: ABC vs TFT (ABC: cross, TFT: circle)

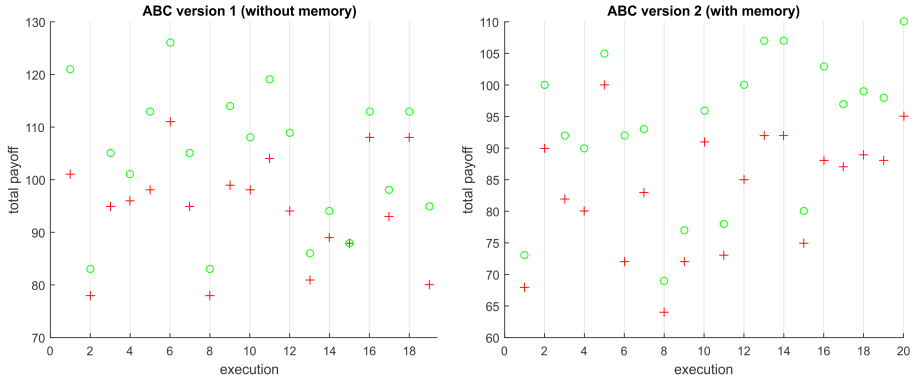


Figure 5: ABC vs ETFT (ABC: cross, ETFT: circle)

Figure 4 shows that both versions of our ABC algorithm develop strategies that help our player to gain higher payoff than a player who follows the TFT strategy.

The ETFT is the first strategy that our algorithm gives inferior results at every execution (Fig. 6). A player that follows ETFT strategy has the advantage to start with defection, thus, he gains payoff regardless of our player's move. If our player cooperates initially, he gains zero payoff and the ETFT opponent gains 5 points of payoff. In case that our player chooses to defect too, they both gain 1 point of payoff according to Table 2. Thus, we are reasonably able to assume that ETFT player gains more payoff than a player who follows any of the other above strategies.

The memory that version 2 provides has no effect in the improvement of our player's payoff.

Table 3: Percentage differences of payoffs (%) between the two versions of ABC algorithm and Benchmark strategies.

	W=20, N=5, M=5, L=10		W=20, N=20, M=5, L=50	
Strategies	ABC(Version 1)	ABC(Version 2)	ABC(Version 1)	ABC(Version 2)
AC	1.9863± 0.18	2.5634± 0.15	2.6674± 0.15	3.9222± 0.17
Random	0.1756± 0.17	-0.6832± 0.23	0.3660± 0.17	-0.6077± 0.16
Pavlov	0.1310± 0.17	2.2174± 0.12	0.1683± 0.17	2.8255± 0.18
TFT	0.2818± 0.11	0.3940± 0.17	0.4293± 0.16	0.4300± 0.13
ETFT	-0.1897± 0.16	-0.2841± 0,16	-0.1663± 0.19	-0.2201± 0.17
PSO	0.0620± 0.19	0.3331± 0.22	0.1443± 0.13	0.4886± 0.16

The PSO algorithm evolved strategies have been used against both of the ABC's versions in the same way that the benchmark strategies did. W players generated by PSO played the PD versus W other generated by ABC. The experiments show that PSO develops very competitive strategies but version 1 and version 2 of ABC algorithm are often more effective and the ABC's players gain higher payoffs as Fig. 6 shows. The most interesting observation is that as the number of iterations increases the ABC's players payoffs are increasing too. As it occurs with most of the benchmark strategies, version 2 with memory performs better. In Table 3 the percentage differences of payoffs between the two versions of ABC algorithm and his opponent (benchmark strategies and PSO) are calculated using the following equation: $Perc_{opponent} = (payoff_{ABC} - payoff_{opponent})/payoff_{opponent}$. In Table 3 we observe the stability of our algorithm.

5. Conclusion

In this paper, we present our algorithmic approach for solving the iterated prisoner's dilemma using a nature inspired method, the artificial bee colony algorithm. We have managed to evolve strategies for a player who faces a PD game for unknown number of iterations and examine his behavior against benchmark strategies and against generated strategies from the PSO algorithm. We have come to conclusion that the ABC algorithm with memory evolves more efficient strategies providing to our player higher payoff compared to the original memory-less ABC algorithms. Furthermore, there is room for future work. For instance, the applicability of other nature inspired algorithms to develop IPD strategies need to be tested and to be compared with our ABC

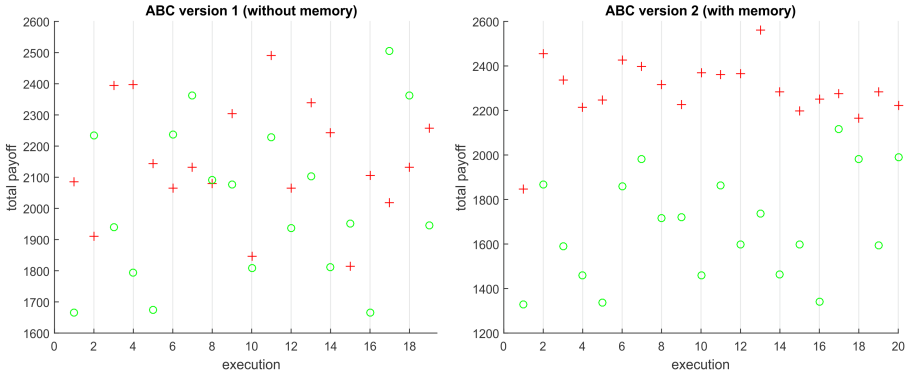


Figure 6: ABC vs PSO (ABC: cross, PSO: circle)

approach. Also, an extension of our algorithm to N-person IPD [16, 17] is interesting.

References

[1] R. Axelrod. The Evolution of Strategies in the Iterated Prisoner's Dilemma. In L. Davis (Ed.) *Genetic Algorithms in Simulated Annealing*, pages 32–41, Pitman, London, 1987.

[2] R. Axelrod and W. D. Hamilton. The evolution of Cooperation. *Science*, 211:1390–1396, 1981.

[3] P. J. Darwen and X. Yao. Co-evolution in iterated prisoners dilemma with intermediate levels of cooperation: Application to missile defense.*International Journal of Computational Intelligence and Applications*, 2:83–107, 2002.

[4] P. J. Darwen and X. Yao. Does extra genetic diversity maintain escalation in a co-evolutionary arms race? *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4(3):191–200, 2000.

[5] P. J. Darwen and X. Yao. On Evolving Robust Strategies for Iterated Prisoner's Dilemma. *Proceedings of the AI Workshops on Evolutionary Computation*, pages 276–292, 1995.

[6] L. A. Dugatkin. Animal Cooperation Among Unrelated Individuals. *Naturwissenschaften*, 89:533-541, 2002.

[7] M. M. Flood. On game-learning theory and some decision-making experiments. Technical Report *DTIC Document*, 1952.

[8] N. Franken and A. P. Engelbrecht. Particle swarm optimization approaches to coevolve strategies for the iterated prisoner's dilemma. *IEEE Transactions on Evolutionary Computation*, 9(6):562–579, 2005.

[9] S. P. H. Heap and Y. Varoufakis. *Game Theory: A Critical Introduction*. Routledge, New York, 1995.

[10] D. Karaboga and B. Basturk. On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, 8(1):687–697, 2008.

[11] J. Kennedy and R. Eberhart. Particle Swarm Optimization. *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, pages 1942–1948, 1995.

[12] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[13] A. Rapoport and A. M. Chammah. *Prisoner's dilemma: A study in conflict and cooperation*. University of Michigan Press, 1965.

[14] Y. Tekol and A. Acan. Ants can play prisoner's dilemma. *IEEE Congress on Evolutionary Computation*, 2:1348–1354, 2003.

[15] A. W. Tucker. The mathematics of Tucker: A Sampler. *The Two-Year College Mathematics Journal*, 14:228-232, 1983.

[16] X. Yao and P. J. Darwen. An experimental study of n-person iterated prisoners dilemma games. *Informatica*, 18(4):435–450, 1994.

[17] X. Yao and P. J. Darwen. Genetic Algorithms and Evolutionary Games. *Commerce, Complexity and Evolution*, pages 313–333, Cambridge University Press, 2000.

SENSITIVITY ANALYSIS OF THE BEE COLONY OPTIMIZATION ALGORITHM

Tatjana Jakšić Krüger
Mathematical Institute of Serbian Academy of Sciences and Arts, Belgrade, Serbia
Faculty of Technical Sciences, Novi Sad, Serbia
tatjana@mi.sanu.ac.rs

Tatjana Davidović
Mathematical Institute of Serbian Academy of Sciences and Arts, Belgrade, Serbia
tanjad@mi.sanu.ac.rs

Abstract Bee Colony Optimization (BCO) is a nature-inspired population-based meta-heuristic method that belongs to the class of Swarm intelligence algorithms. BCO was proposed by Lučić and Teodorović, who were among the first to use the basic principles of collective bee intelligence in dealing with optimization problems. Designing a BCO method in principle includes choosing a procedure for constructive/improvement moves, an evaluation function and setting BCO parameters to a suitable values. Topic of this work is addressing the influence of right choice of BCO underlying procedures, such as the choice of loyalty functions, and influence of parameter variations on algorithm performance, by means of visual inspection. Analyses were conducted for simple variant of scheduling problem. Also, to achieve good alternatives for reported solutions, new evaluation methods for scheduling problem are presented.

Keywords: Empirical analysis, Meta-heuristics, Parameter tuning, Swarm intelligence.

1. Introduction

Apart from the significant advances in computer technology and progress in disciplines relevant for solving optimization problems, practical complex problems are still challenging in the sense that it is hard to solve realistically large instances in reasonable computation times. On the top of that issue, there is a question of configuring solver's parameters. The

performance of most meta-heuristic methods is tightly connected with the right choice of their parameters, resulting with analyses that involve yet another optimization problem. One possible approach for dealing with such issues is empirical analysis, in most of the cases connected with the concrete application and implementation.

Bee colony optimization is a population-based meta-heuristic method that was first proposed by Lučić and Teodorović in 2001 [13]. The inspiration for creating new multi-agent system, such as BCO, originates from foraging behavior of the honey bees. This behavior is suitable for modeling since the practice of collecting and processing nectar is highly organized. The first version of the BCO algorithm was developed as a constructive procedure, where each artificial bee is building a solution from scratch. Later variant of BCO, known as improvement BCO, used modification of complete solutions. To provide better understanding of the BCO structure, the introduction into the behavior of the bees in nature is being presented.

In nature, honey bees succeed to find nectar among limited resources in quite efficient manner, without control of some central management and within unpredictable and dynamic environment. The reason for such a success is the capacity for communication using skills that most resemble to symbolic language [10]. It was Karl von Frisch that in the mid-1940s first recognized the *waggle dance* [18], for which he earned Nobel Prize in 1973. The bees are using waggle dance to learn about various properties of food source, such as the position defined by the direction and the distance.

Basically, a mathematical model of the foraging behavior of honey bees can be described as follows. Bees that are searching and collecting the nectar are known as *worker bees*. They collect and accumulate food for later use by other bees. The worker bees that are exploring the area, typically in the neighborhood of their hive, are called *scout bees*. After completing the exploration, scout bees return to the hive and inform their hive-mates about the locations, quantity and quality of the available food sources in the areas they have examined. In the case they have discovered nectar, scout bees dance in the so-called "dance floor area" of the hive using a ritual called "waggle dance", in an attempt to attract the remaining members of the colony to follow their lead. If a bee decides to leave the hive and collect the nectar, it will follow one of the dancing scout bees to the previously discovered location. After returning to the hive with a load of nectar, the foraging bee then decides for one of the several scenarios: (1) it can try to recruit its hive-mates with the dance ritual before returning to the food location; (2) it can continue with the foraging behavior at the discovered nectar source, without recruiting the

rest of the colony; (3) it can abandon the food source and return to its role of an *uncommitted* follower [3, 4]. Although it is yet unknown how an uncommitted bee decides among several recruiters, the fact is that "the loyalty and recruitment among bees are always a function of the quantity and quality of the food source" [8, 17].

2. The BCO Algorithm

The BCO method is based on engagement of a group of *artificial bees* (B individuals) in search for the optimal solution [8]. The homogeneity of artificial bees is being presumed, where each bee generates one solution to the problem. The homogeneity implies that, unlike worker bees in nature, all the artificial bees in BCO are involved in foraging process. The search process of artificial bees is conducted through iterations, during which bees also communicate in order to compare the quality of obtained solutions, until some predefined stopping criteria is being satisfied. In regard to this clear distribution of tasks for artificial bees, each iteration of the BCO algorithm can be represented as a composition of alternating phases (steps): *forward pass* and *backward pass*.

During the forward pass, all the artificial bees are performing the exploration independent from each other, and therefore, no information is being exchanged in this phase. The method of exploration depends on the implementation of the corresponding BCO algorithm, that is, choice of heuristic. The exploration is performed through certain (predefined) number of moves to either construct the part of a solution [7] or modify the existing complete solution [6]. The number of moves within one forward pass can be represented as a function of the parameter NC. The parameter NC represents the second parameter of the BCO method and its values are influencing the exploitation of the search. Typically, it is used to determine the frequency of information exchange between bees, that is, the number of forward/backward passes during one iteration. At the end of the forward pass the new (partial or complete) solution is generated for each bee [8].

During the backward pass of the BCO algorithm all the artificial bees share the information about the discovered solutions. The information being exchanged in the BCO algorithm contains the quality of each (partial) solution, with respect to the best and the worst solution. Each artificial bee decides, with a probability depending on the solution quality, whether it will stay *loyal* to its solution or not. The artificial bees that stay loyal to their solutions are becoming *recruiters*. Artificial bees that are not loyal to their current solutions, become *uncommitted*, and have to select among the solutions advertised by the recruiters. The se-

lection process for one of the advertised solutions is stochastic, in such a way that better solutions are given higher probabilities to be chosen for further exploration. Consequently, within each backward pass all bees are being divided into two groups: recruiters, and uncommitted bees.

Initialization: Read input data.
Do
(1) Assign a(n) (empty) solution to each bee.
(2) *For* $(i = 0; i < NC; i++)$
// **forward pass**
(i) Perform move for each bee.
// **backward pass**
(ii) Evaluate the (partial/complete) solutions;
(iii) Loyalty decisions;
(iv) *If* (bee not loyal), choose a recruiter by roulette wheel.
(3) Evaluate all solutions. Update x_{best} and $f(x_{best})$
While stopping criterion is not satisfied.
return $(x_{best}, f(x_{best}))$

Figure 1: Pseudo-code for BCO

The pseudocode of the BCO algorithm is given in Fig. 1. Steps (i) and (ii) are problem dependent and should be resolved in each implementation of the BCO algorithm. On the contrary, other steps of the BCO are problem independent. These steps specify loyalty decision (step (iii)) and recruiting process (step (iv)), and are therefore described in more detail in the following text.

2.1 Loyalty Decision

In order for bees to share information about the quality of discovered (partial) solutions, the BCO algorithm is running through three stages: 1. Evaluation; 2. Loyalty decision; and 3. Recruitment. If value $C_b\,(b = 1, 2, ..., B)$ denotes the evaluation value of the b-th bee (partial) solution, then it is being normalized to the $[0, 1]$ interval in such a way that larger normalized value O_b corresponds to the better (partial) solution. Usually the evaluation is implemented so that it corresponds to the formulation of the objective function [15].

In the next stage of backward pass the loyalty functions allow, for bees who start exploration from different points in the search space, to decide whether to become uncommitted followers, or to continue exploring already known solutions. The probability that b-th bee (at the beginning

of the new forward pass) is loyal to its previously generated (partial) solution can be expressed as follows:

$$p_b^{0,u+1} = e^{-\frac{1-O_b}{u}}, \quad b = 1, 2, \ldots, B, \tag{1}$$

where parameter u corresponds to the forward pass counter. In this form equation (1) assures that the bee b will stay loyal with a higher probability to discovered (partial) solutions of a good quality (the ones with higher O_b value). Moreover, as the search process advances the influence of the already discovered (partial) solution increases, i.e., the probability that bee will keep and advertise its current solution has larger value.

Until recently, loyalty function $p_b^{0,u+1}$ was most often used when dealing with optimization problems. From an analytical perspective, it can be reasoned that its utilization agrees well when the search process is implemented so that often interruptions during backward pass should be avoided. In other words, when it is obvious that the search path of a bee will most probably lead to a good solutions, then increasing its loyalty during one iteration assists well such endeavor. However, when the emphasis should be on the exploration of the solution space, different perspective into the measure of bees loyalty needs to be considered. In recent work [16] it was reported that for some variants of BCO, better performance could be achieved if the current forward pass index (u) was omitted in the loyalty decision process. Some other probability functions were examined in [14]. A new study was therefore conducted for 10 different loyalty functions:

(1) $p_b^{0,u+1} = e^{-\frac{1-O_b}{u}}$,
(2) $p_b^1 = e^{-O_{\max}-O_b}$,
(3) $p_b^2 = O_b$
(4) $p_b^{3,n_{it}} = e^{-(1-O_b)/n_{it}}$,
(5) $p_b^{4,u+1} = e^{-(1-O_b)/\sqrt{u}}$,
(6) $p_b^{5,u+1} = e^{-(1-O_b)\sqrt{u}/\sqrt{u+1}}$,
(7) $p_b^{6,u+1} = e^{-(1-O_b)/\log u}$,
(8) $p_b^{7,u+1} = e^{-(1-O_b)/u\log(u+1)}$,
(9) $p_b^8 = e^{-2*(1-O_b)}$,
(10) $p_b^{9,u+1} = e^{-(1-O_b)\log(u+1)/\log(u+2)}$.

Two classes of loyalty functions can be distinguished: *Class I*, as a function of parameter O_b ($p_b^{1,2,8}$), and *Class II* as a two variable function of O_b and counter u or iteration counter n_{it} ($p_b^{0,3,4,5,6,7,9}$).

2.2 Recruiting Process

The probability that b's (partial) solution would be chosen by any uncommitted bee depends on the solution quality of a recruiter b and

equals to:

$$p_b = \frac{O_b}{\sum_{k=1}^{R} O_k}, \quad b = 1, 2, \ldots, R, \tag{2}$$

where O_k represents normalized value for the objective function of the k-th advertised partial solution and R denotes the current number of recruiters.

3. Sensitivity Analysis of BCO

Designing a BCO method in principle includes choosing a procedure for constructive/improvement moves, an evaluation function and setting BCO parameters to a certain values that are usually determined by some set of pilot studies, some previously published work or even intuition. However, the analysis of different settings for loyalty function is lacking in current literature, even though it is a part of the generic section of the BCO method and is not problem specific. One of our goals was to address this issue.

Empirical analysis of the meta-heuristic method belongs to interdisciplinary research and in many cases can require great effort due to the stochastic nature of the method or, in some cases, tuning large number of parameters whose interaction should be expected [11]. Unlike specific orientated optimization algorithms, meta-heuristics methods are categorized by its parameters and/or different modules [2]. Such structure, when implemented, can expose different behavior as parameter values are changing. During the last decade, different tools for experimental analysis were proposed and/or inspected, most of them based on modeling response values with linear or nonlinear models and/or implementing three basic steps: screening, experimentation and exploitation [1, 9, 19]. The literature on this topic today is overwhelming, so the right choice of the tunning method for BCO remains one of the future challenges.

The aim of this work is to provide first insights on behaviour of BCO by following guidelines of many researches who were concerned with methodical empirical analysis of heuristic and meta-heuristic methods. A thorough scientific testing of BCO method can be computationally too extensive, which is why first steps into empirical analysis of BCO is addressing questions of *sensitivity analysis*. Sensitivity analysis corresponds to analysis of variation of algorithm's response values, such as quality of solution (usefulness, utility) or time of execution, while changing its parameter configuration [12]. We examined here a constructive variant of BCO algorithm and used a simple scheduling problem as an example.

Problem formulation. Let m be the total number of identical processors engaged, and n number of non-preemptive independent tasks. The considered scheduling problem consists of assigning tasks to processors, and determining their starting times. Let $T = \{1, 2, \ldots, n\}$ be a given set of independent tasks, where each tasks $i \in T$ has to be processed by exactly one among the identical processors from the set $P = \{1, 2, \ldots, m\}$. Each processor can process only one tasks at the time, and once the tasks has started it will continue to run on the same processor until completion. Let l_i be the processing time of task i $(i = 1, 2, \ldots, n)$, which is known and fixed. The goal is to find a schedule of tasks on processors such that the corresponding completion time of all tasks is minimized. The mathematical programming formulation of the problem can be found in [7], together with the implementation of BCO algorithm that was used in this work. Problem here is referred to as finding minimal *makespan*.

Problem instances. Instances used for testing BCO algorithm represent randomly generated instances with known optimal solutions [7]. They were introduced in [5] for Multiprocessor Scheduling Problems with Communications Delays. The test instances are named as Iogra$< n >_< m >$, where n designates number of tasks, and m denotes number of processors (graph density was set to zero). Specifically, in the work of Davidović et al [7] different problem-size instances were used, i.e., $m = \{2, 4, 6, 8, 9, 10, 12\}$ and number of tasks ranging from 100 to 500. It was concluded that n does not influence the complexity of the problem, as confirmed in new studies. Additionally, new results have shown that the influence of the varying number of processors on the complexity of the problem is not so straightforward. Structure of these problem instances is introduced using box-plots and presented in Fig. 2.

4. Results

Evaluation function $f_b^1 = y_{max}$, introduced in the paper of Davidović et al. [7] depends only on the value of the makespan (y_{max}), and therefore is more receptive due to its lower computational costs. Newly proposed function $f_b^2 = y_{max}/L'$ depends on two parameters, makespan and the current sum of computational time of non-scheduled tasks L'. With introduction of evaluation function f_b^2 better partial solution was associated with larger values (maximality principle), which suggested new course of how the problem can be solved while maintaining objective of minimizing makespan. To best describe new approaches, a concept of *methods of evaluations* was introduced in order to illustrate different evaluation of partial solution in the backward pass of BCO. In case of f_b^1

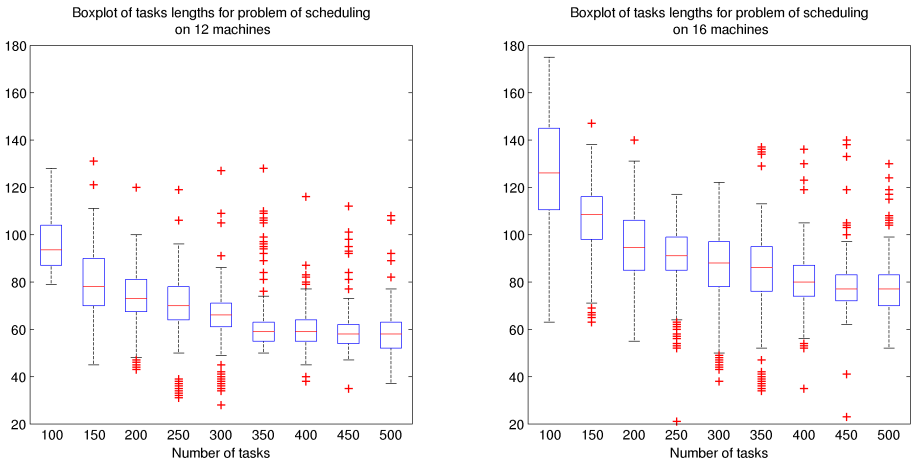


Figure 2: Box-plot for $m = 12, 16$ and 9 instances in relation to the n, where possible outliers are marked with red crosses.

both maximization and minimization principles can be used, thus yielding two different methods of evaluations. When partial solution with biggest makespan is evaluated as the best, it's normalized value is equal to 1, while the solution with lowest makespan will be appointed with normalized value 0. This method of evaluation is denoted here as max, f_b^1. In case of minimization, partial solution with lowest makespan is marked as the best among the population of solutions, and its normalized value corresponds to 1, while maximal makespan will be normalized to value 0. Such method of evaluation is denoted here as min, f_b^1. Justification for incorporating these methods comes from initial set of studies when it was recognized that the solution with smaller makespan doesn't necessarily lead to best result and that both minimal and maximal makespan can be used to quantify good partial solution.

An experiment will be considered as a set of 100 independent runs of the investigated algorithm. All experiments were conducted for predefined values of parameters. In case of the test instances Iogra100_12/16 the experiments were accomplished on complete parameter's search space (Fig. 3). However, for instances where $n \geq 150$ the experiments were conducted on sub-regions of parameters search space. Such restriction comes from limitations imposed on values of NC as they are dependent on number of constructive moves in BCO instance. For example, as the number of tasks increases, maximal number of forward/backward passes also increases which greatly expands the parameters search subspace $S \subseteq \mathcal{B} \times \mathcal{NC}$. Since experimental analyses should be conducted

for each pair (B, NC) it would be too time consuming to include values for $NC > 100$. The choice of values for maximal number of bees was determined arbitrary. Domains of all BCO parameters are provided in Table 1.

Table 1: Parameter space for experimental analysis of BCO.

Parameter	**Domain**
method of evaluation	min, f_b^1; max, f_b^1; max, f_b^2
loyalty function	$p_b^i,\ i \in [0, .., 9]$
B	$[1, 20]$
NC	$[1, 100]$

In each run of an experiment the solution quality was measured within the stopping criteria defined as maximum number of iterations, while maximal number of iterations was set to 100. The set of results used to conduct sensitivity analysis of the BCO parameters is being presented in Fig. 3.

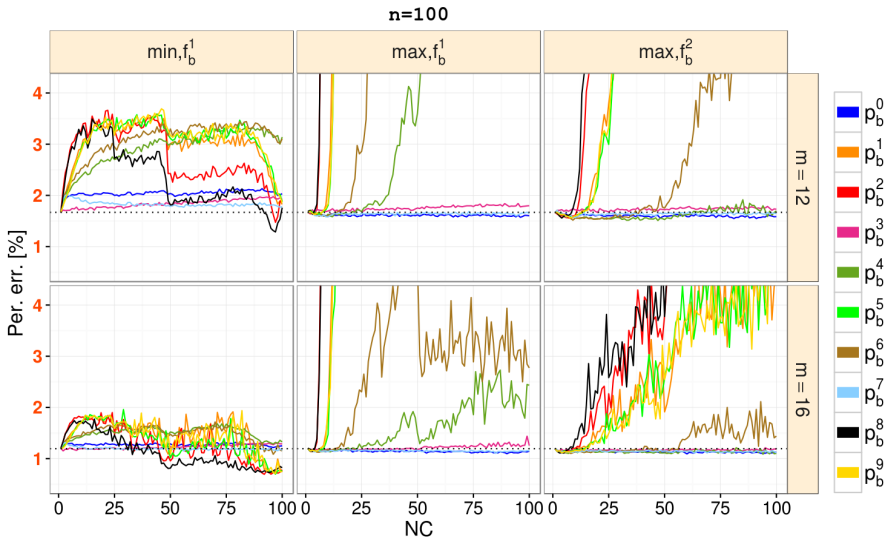


Figure 3: The influence of parameter NC on the averate solution quality in regard to the method of evaluation and loyalty function.

Graphics in Fig. 3 illustrate influence of different BCO parameters on reported average solutions quality, measured by percent error, for two problem instances of different class and same number of tasks. The

main goal of this presentation was to visually inspect improvements in the solution quality when parameter NC changes its value in respect to structural parameters of BCO. As three methods of evaluations were used, graphics are arranged to distinguish their influence on each loyalty functions when $NC \in [1, 100]$. Specifically, each color on a plot corresponds to different loyalty function, whereas dashed black line signifies reference (start) case when $NC = 1, B = 20$. Reference case is used to simulate the behaviour of an underlying heuristic. All values on the graphics correspond to a number of bees that generated best results. It should be noted that parameter B can take different values when reporting on the best value. For example, on problem instance Iogra100_12 and for method of evaluation min, f_b^1, presented profiles of average results are mostly generated when $B = 20$. On the same instance, the best average result in the case of configuration (max, f_b^1,p_b^9) was achieved when $B = 18$. Actually the best solutions were generated when large population of bees was utilized ($B \geq 18$).

There are few interesting observations drawn from Fig. 3. First, for some cases of loyalty functions the influence of methods of evaluation is not distinguishable, as it is the case for $p_b^{0,3,7}$. The reason for such behaviour comes from the fact that these three loyalty functions can converge fast toward cases where the majority of the partial solutions will be transfered to the next iteration. It is most likely that the search is stranded in the local minimum. Therefore, it can be concluded that loyalty functions $p_b^{0,3,7}$ show robustness to changes of other qualitative and quantitative BCO parameters, however, without significant improvement in the solution quality when compared with reference case.

Unlike previous group of loyalty functions, some do not converge fast ($p_b^{4,5,6,9}$) or not converge at all ($p_b^{1,2,8}$). Such a property can yield bigger perturbations in the reported solution quality since the number of partial solutions that will be used for exploitation, varies throughout an iteration. This variation depends on the utilization of evaluation function (or method of evaluation) and the structure of problem instance. Although showing significant fluctuations, those loyalty functions are able to bring improvements into the solution quality, at least for minimization of f_b^1. Among them, loyalty functions p_b^2 and p_b^8 perform the best in respect to the starting case $NC = 1$.

In addition, graphics also reveal that inside of these set of loyalty functions certain groups exhibit similar behaviour. Such groups are: $p_b^{1,5,9}$, $p_b^{2,8}$ and $p_b^{4,6}$. To distinguish the influence of loyalty functions within a group, further analysis needs to be conducted on the properties of recruitment process. However, this is beyond the scope of this paper.

Since the results were sensitive to the choice of problem instance, it was obvious that additional analysis on the whole considered set of problem instances should be undertaken. Such presentation was then used to determine a robust set of parameters configurations that would generate the best results. For this reason a group of graphics presenting the influence of BCO parameters on different problem instances is given in Fig. 4. As before, the value for B varies in interval $[18, 20]$ when generating good quality soutions, with one exception where $B = 15$ was reported by function p_b^8 on problem instance Iogra400_12. The series of graphics on Fig. 4 consists of Fig. 3 and eight more, in regard to the dimension of a problem instance. Once more it should be noted that NC values do not cover complete parameter space for problems of dimension $n > 100$ due to high computational cost. However, we can still notice similarities between the graphics from different groups, and draw similar conclusions as in case of $n = 100$. First paramount conclusion is related to Class II type of loyalty functions, such as p_b^k, $k \in \{0, 3, 4, 5, 6, 7, 9\}$. From this group, loyalty functions p_b^0, p_b^3 and p_b^7 are the most conservative due to small changes in the reported average solutions over the complete interval $NC \in [1, 100]$, regardless of method of evaluation. Additionally, they have generated practically insignificant improvements, and as such do not represent good choice for the BCO method on considered set of problem instances. Remaining Class II loyalty functions showed high sensitivity to utilization of method of evaluation and problem instance. No pattern was able to be identified in respect to NC that would generate good solutions. Actually, only p_b^5 and p_b^9 succeeded to be better then the starting $NC = 1$ case but solely on instances Iogra100_12/16, Iogra150_16, Iogra200_12, Iogra250_16 and Iogra300_12/16. Also, these two loyalty functions exhibit similar behavior throughout the search. Loyalty functions of Class I, p_b^1, p_b^2, p_b^8, showed very high sensitivity to changes in quantitative values of B and NC and choice of problem instance. Between these three, the most unsuccessful was p_b^1 and generated solutions that resemble those of $p_b^{5,9}$. Loyalty functions p_b^2 and p_b^8 are the only one that presented certain pattern for values of NC which brings improvements with respect to the starting case $NC = 1$.

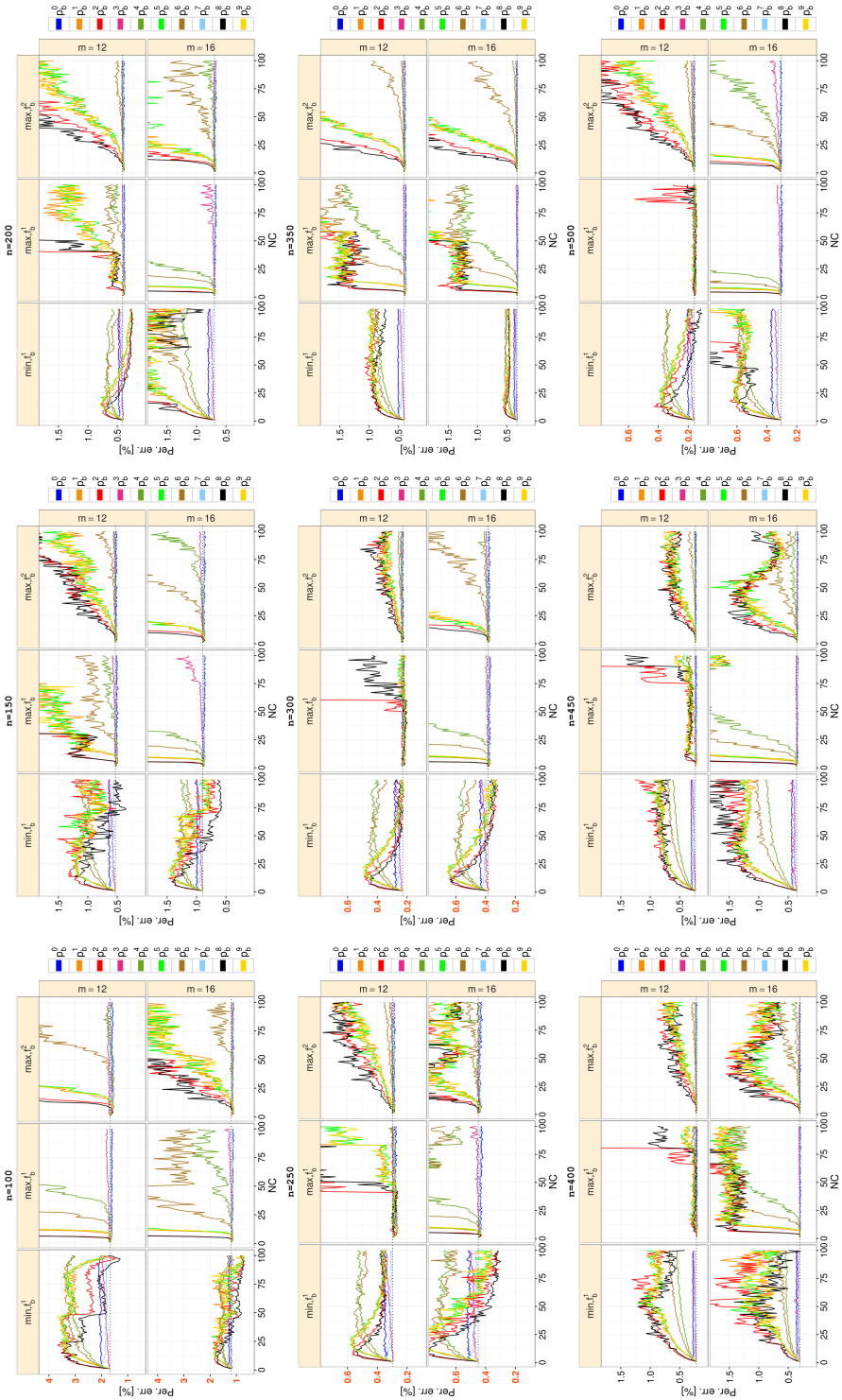


Figure 4: Illustrating the influence of method of evaluation, loyalty function and parameter NC on the performance of BCO on problem of scheduling independent tasks on identical processors.

5. Conclusion

We have tested the influence of different BCO method's parameters on the quality of solutions. In total four BCO parameters were analyzed: B, NC, method of evaluation and loyalty function. Sensitivity analysis was conducted by means of visual inspection of series of graphics categorized by the type of problem instance. Furthermore, each graphic consists of set of plots that reveal the influence of loyalty function with regard to method of evaluation, for fixed values of bees and varying number of parameter NC.

Conducted empirical analysis showed that on provided set of problem instances in 50% of cases best results were obtained for minimization of f_b^1. Furthermore, good quality was obtained for larger population of bees, that is $B \in [18, 20]$, and when $NC \geq 90$. The most successful loyalty functions were those of Class I, p_b^2 and p_b^8 in particular, which were not (often) used previously in the literature. We can conclude that on considered benchmark set of problem instances, configurations $\{min, f_b^1, p_b^{2,8}, B \in [18, 20], NC \geq 90\}$ were most successful, being the only one to offer significant improvements in the solution quality in comparison with reference case. Some additional tests indicate that successful values of NC can be restricted to $[0.9n, n]$, which has yet to be confirmed.

The possible directions for future work could be implementing some of the tuning methods mentioned in [9], on constructive and improvement versions of BCO.

Acknowledgment: This research has been supported by Serbian Ministry of Education, Science and Technological development, Grant. Nos. 174010, 174033, 174008, 044006.

References

[1] T. Bartz-Beielstein and M. Preuß. Experimental analysis of optimization algorithms: Tuning and beyond. In *Theory and Principled Methods for the Design of Metaheuristics*, pages 205–245, Springer, 2014.

[2] M. Birattari. *Tuning metaheuristics: a machine learning perspective*. Studies in Computational Intelligence, vol. 197. Springer, 2009.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford, 1999.

[4] S. Camazine and J. Sneyd. A model of collective nectar source selection by honey bees: self-organization through simple rules. *Journal of Theoretical Biology*, 149(4):547–571, 1991.

[5] T. Davidović and T. G. Crainic. Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multi-

processor systems. *Computers & Operations Research*, 33(8):2155–2177, 2006.

[6] T. Davidović, D. Ramljak, M. Šelmić, and D. Teodorović. Bee colony optimization for the p-center problem. *Computers & Operations Research*, 38(10):1367–1376, 2011.

[7] T. Davidović, M. Šelmić, D. Teodorović, and D. Ramljak. Bee colony optimization for scheduling independent tasks to identical processors. *Journal of Heuristics*, 18(4):549–569, 2012.

[8] T. Davidović, D. Teodorović, and M. Šelmić. Bee Colony Optimization Part I: The Algorithm Overview. *Yugoslav Journal of Operational Research*, 25(1):1367–1376, 2014.

[9] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[10] J. L. Gould. Honey bee recruitment: the dance-language controversy. *Science*, 189(4204):685–693, 1975.

[11] H. H. Hoos and T. Stützle. *Stochastic local search: Foundations & applications*. Elsevier, 2005.

[12] H. H. Hoos and T. Stützle. Empirical analysis of randomized algorithms. In T. F. Gonzalez (Ed.) *Handbook of Approximation Algorithms and Metaheuristics*, Chapter 14, Chapman & Hall/CRC, 2007.

[13] P. Lučić and D. Teodorović. Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence. *Preprints of the TRISTAN IV*, pages 441–445, 2001.

[14] P. Maksimović and T. Davidović. Parameter Calibration in the Bee Colony Optimization Algorithm. *Proceedings of the 11th Balkan Conference on Operational Research*, pages 263–272, 2013.

[15] Z. Michalewicz and David B. Fogel. *How to solve it: modern heuristics*. Springer, 2004.

[16] M. Nikolić and D. Teodorović. Empirical study of the Bee Colony Optimization (BCO) algorithm. *Expert Systems with Applications*, 40(11):4609–4620, 2013.

[17] D. Teodorović. Bee colony optimization (BCO). *Innovations in swarm intelligence*, Studies in Computational Intelligence, vol. 248, pages 39–60. Springer, 2009.

[18] K. von Frisch. Decoding the language of the bee. *Science*, 185(4152):663–668, 1974.

[19] X.-S. Yang, S. Deb, M. Loomes, and M. Karamanoglu. A framework for self-tuning optimization algorithm. *Neural Computing and Applications*, 23(7-8):2051–2057, 2013.

A PARAMETER CONTROL SCHEME FOR DE INSPIRED BY ACO

Dražen Bajer, Goran Martinović
Faculty of Electrical Engineering, Josip Juraj Strossmayer University of Osijek, Croatia
drazen.bajer@etfos.hr, goran.martinovic@etfos.hr

Abstract Differential evolution requires a prior setting of its parameters. Appropriate values are not always easy to determine, even more since they may change during the optimisation process. This is where parameter control comes in. Accordingly, a scheme inspired by ant colony optimisation for controlling the crossover-rate and mutation factor is proposed. Conceptually, artificial ants guided by a pheromone model select parameter value pairs for each individual. The pheromone model is steadily updated in order to reflect the current optimisation state. Promising results were achieved in the comprehensive experimental analysis. Nonetheless, much room for potential improvements is available.

Keywords: Ant colony optimisation, Construction graph, Differential evolution, Parameter control, Pheromone model.

1. Introduction

Differential evolution (DE) [17, 19] is a simple and effective population-based search and optimisation method. It has been successfully applied to various global optimisation problems, although originally proposed for numerical optimisation (see, e.g., [2, 6, 10, 12]). Like other evolutionary algorithms (EAs), DE also requires the setting of its parameters prior to running and its performance is largely dependent on those settings. Another issue is the fact that different parameter settings may be suitable for different problems [5, 22, 23]. Accordingly, although various recommendations for setting the parameters exist (see, e.g., [17, 19]), the search for adequate parameter values (*parameter tuning*) that will result in satisfactory performance on a given problem requires a considerable effort and usually boils down to a trial and error approach. Furthermore, "optimal" values of the parameters may change during the optimisation [8, 23]. This problem cannot be solved by just finding good parameter values a priori, but requires a constant adjustment of those values

(*parameter control*). The need for parameter control in EAs was realised early [5]. Finally, well designed schemes can improve or enhance performance in terms of robustness and convergence-rate [23].

This paper proposes a parameter control scheme for DE, inspired by ant colony optimisation (ACO). More particularly, a scheme for controlling the parameters representing the crossover-rate and mutation factor. Two sets of parameter values are provided, and for each individual a value pair is selected. Conceptually, the selection is made by artificial ants (each solution is assigned one), while their choices are influenced by artificial pheromone.

The remainder of the paper is organised as follows. Section 2 briefly describes DE and ACO, it also provides a short overview of parameter control in DE. The proposed parameter control scheme is described in Section 3. In Section 4 the obtained experimental results are reported and discussed. Finally, the drawn conclusions are stated in Section 5.

2. Preliminaries

2.1 Differential Evolution

Differential evolution is an example of a fairly successful EA for numerical optimisation. Besides that, it is conceptually simple which makes it attractive to practitioners attempting to solve their problem or problems using a bio-inspired optimisation algorithm. A brief description of the canonical or standard DE, as outlined in Algorithm 1, follows.

The population of DE is composed of NP individuals typically called vectors $\mathbf{v}^j = (\mathbf{v}_1^j, \ldots, \mathbf{v}_d^j) \in \mathbb{R}^d$, $j = 1, \ldots, NP$. In each generation/iteration a new population is created by mutation and crossover of individuals, i.e., vectors of the current population. Mutation and crossover produce a trial vector (offspring)

$$\mathbf{t}_i^j = \begin{cases} \mathbf{v}_i^{r1} + F \cdot (\mathbf{v}_i^{r2} - \mathbf{v}_i^{r3}), \\ \text{if} \quad \mathcal{U}_i(0,1) \leq CR \text{ or } i = r_j \\ \mathbf{v}_i^j, \\ \text{otherwise} \end{cases} \quad , \quad i = 1, \ldots, d, \tag{1}$$

where $\mathbf{v}^{r1}$, $\mathbf{v}^{r2}$ and $\mathbf{v}^{r3}$ are randomly selected vectors from the current population, and which are selected anew for each target vector $\mathbf{v}^j$, such that $j \neq r1 \neq r2 \neq r3$. The parameter $F \in [0, \infty\rangle$ is the scale (mutation) factor, while $CR \in [0, 1]$ is the crossover-rate, $\mathcal{U}_i(0, 1)$ is a uniform deviate in $[0, 1]$, and r_j is randomly chosen from the set $\{1, \ldots, d\}$. After the new trial vector population of size NP is created, a one-to-one selection takes place. More specifically, a trial vector $\mathbf{t}^j$ passes into the next generation

only if it is equal or better (in terms of the objective function f) than the corresponding target vector $\mathbf{v}^j$.

The described algorithm represents the canonical DE, usually denoted as DE/rand/1/bin [17, 19]. Many other variants that improve on the canonical algorithm have been proposed in the literature. A comprehensive review of DE variants can be found in, e.g., [14].

2.2 Parameter control in differential evolution

Essentially, the DE algorithm requires the setting of three parameters. A multitude of different parameter control mechanisms or schemes for DE can be found in the literature. Most of those schemes are designed for controlling a subset of the parameters, most notably CR and F. For example, a few simple deterministic mechanisms for varying the value of parameter F during the optimisation process can be found in [3, 9]. Those propose to randomly vary F inside a preset interval, or to linearly reduce it with the number of performed iterations. Tvrdík [21] proposed, among other, a scheme where for each individual one value pair of F and CR, from nine available, is chosen probabilistically. The probabilities associated with the pairs are based on their success. The adaptive scheme by Yu and Zhang [22] utilizes the quality of population individuals and their distance from the best-so-far. Based on that data and defined rules the values of F and CR are appropriately adjusted. In the self-adaptive scheme proposed by Brest *et al.* [1], each individual is assigned its own value of F and CR. The assigned values are regenerated randomly before mutation and crossover take place with set probabilities. Noman *et al.* [15] took a similar approach, where the parameter values assigned to a individual are regenerated (randomly) only if its offspring is worse than the current population average. Furthermore, Zhang and Sanderson [23] proposed a scheme where the values of CR and F assigned to each individual are generated by Gaussian and Cauchy deviates, respectively.

Algorithm 1 Canonical DE (`DE/rand/1/bin`)

```
1: Set NP, CR and F, and initialize population
2: while termination condition not met do
3:     for j := 1 → NP do
4:         create trail vector t^j (Eq. (1))
5:     end for
6:     for j := 1 → NP do
7:         if f(t^j) ≤ f(v^j) then
8:             v^j := t^j
9:         end if
10:    end for
11: end while
```

In both cases the values are around the means of previously successful parameter values taken from the whole population. New values are generated each time before mutation and crossover take place. This scheme was adopted and refined in other studies, like the ones in [7, 20]. Also, Zhao *et al.* [24] recently proposed a scheme that is somewhat similar but differs in the calculation of the mean/location around which new values of F and CR are generated. Moreover, new values of CR and F are generated by, conversely to the approach taken in [23], by Cauchy and Gaussian deviates, respectively.

2.3 Ant Colony Optimisation

Ant colony optimisation [4, 18] is a meta-heuristic in which a population of artificial ants cooperates in the search for solutions to a given opitmisation problem. The main inspiration comes from the indirect communication among ants (stigmergy) via the pheromone trails laid by them as they move. A number of ACO algorithms have been proposed and successfully applied to various optimisation problems [13].

Prior to the application of ACO, the problem at hand must be transformed into the problem of finding the best path on a weighted graph (construction graph). Solutions are constructed incrementally by the ants as they traverse the construction graph, whereby each node represents a solution component. The choices of paths to take are made probabilistically, and are influenced by artificial pheromone trails and eventually available heuristic information, which represent the pheromone model. The values associated with the pheromone model (graph weights associated with the arcs or nodes) are dynamically modified. This is achieved through pheromone deposition by selected ants, and a preceding pheromone evaporation.

3. Proposed Parameter Control Scheme

Although all DE parameters are interdependent, most of the approaches from the literature seem to be limited to the control of parameters F and CR. This may lead to the conclusion that a well designed scheme can adjust those values in accordance with the set population size NP. Hence, the proposed scheme is intended for controlling parameters F and CR.

In the proposed scheme the sets $\mathcal{S}_F = \{F_i = 0.1 \cdot i \colon i = 1, \ldots, p{=}10\}$ and $\mathcal{S}_{CR} = \{CR_i = 0.1 \cdot (i-1) \colon i = 1, \ldots, q = 11\}$ are given, representing available values of F and CR, respectively. Good value pairs $(F_k, CR_l) \in \mathcal{S}_F \times \mathcal{S}_{CR}$ are selected for and assigned to each vector $\mathbf{v}^j$ separately. In order to facilitate the selection and to establish good val-

ues, the problem is modeled as a complete bipartite graph, as illustrated in Fig. 1. Conceptually, artificial ants assigned to each vector, traverse the construction graph. Each node is associated with a value from the set $\mathcal{S}_F$, i.e., the set $\mathcal{S}_{CR}$. This way, parameters values are selected. The selection is probabilistic (roulette wheel selection), and the probability of selecting some value $F_k \in \mathcal{S}_F$ or $CR_l \in \mathcal{S}_{CR}$ is

$$p_{F,k} = \frac{\tau_{F,k}}{\sum_{i=1}^{p} \tau_{F,i}},\ p_{CR,l} = \frac{\tau_{CR,l}}{\sum_{i=1}^{q} \tau_{CR,i}},\ k = 1, \ldots, p,\ l = 1, \ldots, q, \quad (2)$$

where $\tau_{F,k}$ and $\tau_{CR,l}$ are the artificial pheromone deposited on node (F, k) and (CR, l), respectively. This represents the pheromone model.

The pheromone values are updated after selecting the new generation. First, evaporation takes place, followed by the deposition of new pheromone, that is

$$\begin{aligned} \tau_{F,k} &= (1-\rho)\tau_{F,k} + \sum_{j \in \mathcal{J}} \Delta\tau^{j}_{F,k}, \qquad k = 1, \ldots, p, \\ \tau_{CR,l} &= (1-\rho)\tau_{CR,l} + \sum_{j \in \mathcal{J}} \Delta\tau^{j}_{CR,l}, \qquad l = 1, \ldots, q, \end{aligned} \quad (3)$$

where $\rho \in \langle 0, 1]$ is the evaporation-rate ($=0.1$), $\mathcal{J}$ is the set of indices of trial vectors that made it into the new generation, $\Delta\tau^{j}_{F,k}$ and $\Delta\tau^{j}_{CR,l}$ are the pheromone to be deposited for vector $\mathbf{v}^j$ on node (F, k) and (CR, l), respectively. A value of 0.1 is deposited on nodes associated with the parameter values selected for $\mathbf{v}^j$, whereas a value of 0 is deposited on the remaining nodes. The pheromone values are bounded by $\tau_{min} = 0.1$ and $\tau_{max} = 1$.

4. Experimental Analysis

An experimental analysis was conducted in order to assess the advantages and shortcomings of the proposed parameter control scheme. The analysis was conducted on the benchmark functions prepared for the IEEE CEC2013 [11]. The test suite is composed of 28 functions.

The proposed scheme was incorporated into the canonical DE (denoted as $\text{DE}_{(\text{PBPS})}$) for the analysis. A comparison with the canonical DE and the same algorithm incorporating the scheme utilised in DER9 [21] (denoted as $\text{DE}_{(\text{DER9})}$), the scheme utilised in aDE [15] (denoted as $\text{DE}_{(\text{aDE})}$), and the scheme utilised in SLADE [24] (denoted as $\text{DE}_{(\text{SLADE})}$) was performed. This way, only the impact of the parameter control schemes was assessed and a fair comparison was enabled.

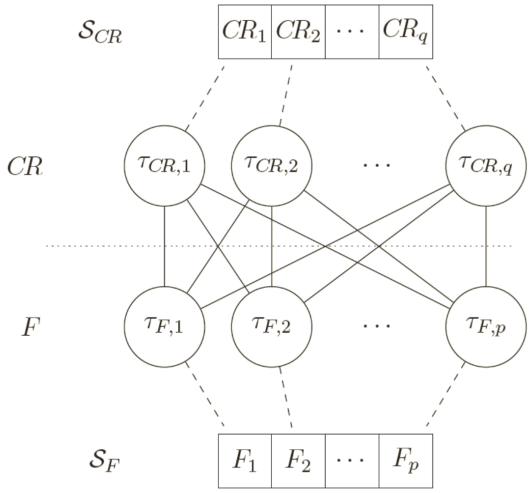


Figure 1: Construction graph and associated parameter values.

4.1 Experiment Setup

For each used algorithm and problem instance, 51 independent algorithm runs were performed. All algorithms were allowed a maximum of $10^4 \cdot d$ function evaluations. Termination occurred as soon as the targeted optimisation error $\Delta f < 10^{-8}$ was reached or the maximum number of function evaluations ($\mathrm{NFE}_{\max}$) was performed. Further on, the common population size of $NP = 100$ (used in, e.g., [1, 7, 20]) was used in all algorithms, while $F = 0.5$ and $CR = 0.9$ (used in, e.g., [1, 15, 16, 22] along with $NP = 100$) were used in the canonical DE.

4.2 Results and Discussion

The results obtained on the test functions for $d = 10$ and $d = 30$ are reported in Table 1 and 4, respectively. The tables show the mean and standard deviation (std. dev.). The Wilcoxon signed rank test with a confidence interval of 95% was performed in order to find if the differences (in means) are statistically significant. Accordingly, the symbol $(-)$ indicates a difference in favour of the DE incorporating the proposed scheme ($\mathrm{DE}_{(\mathrm{PBPS})}$) compared to a given algorithm, the opposite is indicated by the symbol $(+)$, whereas the symbol $(\approx)$ indicates an absence of statistical significance. Furthermore, Tables 2 and 5 show the success-rate in reaching the targeted optimisation error for the considered algorithms.

According to the shown results, the $\mathrm{DE}_{(\mathrm{PBPS})}$ algorithm performed in summary better than the other algorithms used in the comparison. This is also evident from Fig. 2. It may be noted that best results on unimodal functions ($f_1 \sim f_5$) for $d=10$ were achieved by DE and $\mathrm{DE}_{(\mathrm{aDE})}$, but this was not the case for $d=30$. Considering multimodal functions ($f_6 \sim f_{20}$), in most cases, the best results were achieved by $\mathrm{DE}_{(\mathrm{PBPS})}$. This is most prominent on problems for $d = 10$, and slightly less for $d=30$. A similar observation can be made in the case of composition functions ($f_{21} \sim f_{28}$). Nonetheless, it must be noted that in just a few cases, the differences in means are very small or virtually nonexistent, but are statistically significant according to the performed test. Those are certainly of no practical importance. Figures 3 and 4 show, for several chosen functions, the average optimisation error $\overline{\Delta f}$ in relation with the number of performed function evaluations. The figures suggest a convergence-rate of $\mathrm{DE}_{(\mathrm{PBPS})}$ that is greater or close to the best competitor. Interesting to note, on multimodal and composition functions usually better performance was achieved with a DE algorithm incorporating one of the used parameter control schemes. There are several cases in which the canonical DE got trapped early on in a local optimum, unable to escape it. This hints at one of the benefits parameter control seems to provide.

Another relevant matter are the time complexities of the considered algorithms since the parameter control schemes introduce a certain computational overhead. Tables 3 and 6 provide insight into how much this affects the overall execution times and thus the complexity. The shown data have been obtained according to [11], but on function 13 instead of 14 because most of the algorithms were able to reach the targeted optimisation error on 14 however non on function 13. It must be remarked that the timings have been repeated 25 times in order to accommodate variations in execution times and that the median values (as per C—approximated complexity) are reported. As may be noted the largest complexity is ascribed to $\mathrm{DE}_{(\mathrm{DER9})}$ and $\mathrm{DE}_{(\mathrm{PBPS})}$. Nonetheless, those are not substantially larger compared to the other used algorithms incorporating parameter control schemes.

Based on the obtained results, it is clear that parameter control can provide an edge over static parameter settings. This seems to be especially the case on more complex problems (multimodal and composition functions, and higher dimensions). In that regard, the algorithms incorporating parameter control schemes provided greater convergence-rates and were principally less susceptible to being trapped in local optima.

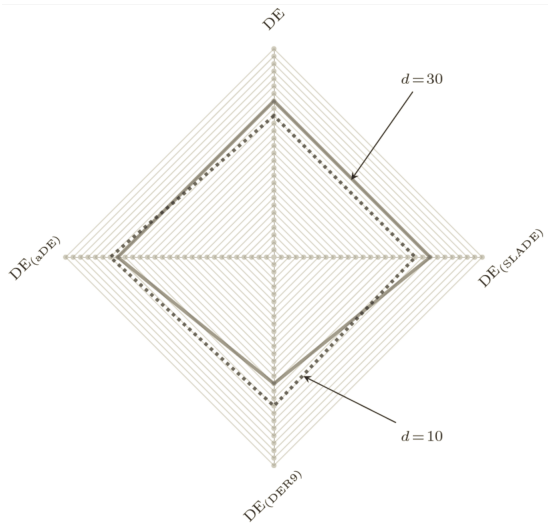


Figure 2: Number of functions on which $\mathrm{DE}_{(\mathrm{PBPS})}$ achieved a lower or equal mean.

5. Conclusion

This paper proposed a scheme for controlling the crossover-rate and scale factor of DE. From a conceptual viewpoint, artificial ants select value pairs from the sets of available parameter values. Their choices are influenced by a pheromone model which is steadily updated. The proposed scheme was incorporated into the canonical DE, and the obtained experimental results suggest good performance.

The main drawback of the proposed scheme is certainly the number of its own parameters. However, they are fixed and need not be changed. It is reasonable to assume that not all are equally important regrading performance, and an analysis in that direction may prove fruitful.

Although promising results were achieved, room for potential improvements is available. Currently, the same amount of pheromone is deposited for each successful parameter value pair, but adjusting that amount according to the achieved improvement may be beneficial. Similarly, the other parameters could be dynamically adjusted as in some successful ACO algorithms. Another direction that should be followed, is the selection of appropriate DE strategies, along with the parameters. In that regard, two possibilities are open, the selection of the crossover and mutation operator separately, or together as one strategy.

Table 1: Results on functions for $d=10$.

f	DE mean	DE std. dev.	$\text{DE}_{(\text{aDE})}$ mean	$\text{DE}_{(\text{aDE})}$ std. dev.	$\text{DE}_{(\text{DER9})}$ mean	$\text{DE}_{(\text{DER9})}$ std. dev.	$\text{DE}_{(\text{SLADE})}$ mean	$\text{DE}_{(\text{SLADE})}$ std. dev.	$\text{DE}_{(\text{PBPS})}$ mean	$\text{DE}_{(\text{PBPS})}$ std. dev.
1	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	7.657E-01 (−)	5.30E+00	0.000E+00	0.00E+00
2	0.000E+00 (+)	0.00E+00	2.012E+02 (−)	2.85E+02	0.000E+00 (+)	0.00E+00	7.169E+05 (−)	6.51E+05	2.614E+01	1.42E+02
3	1.569E-01 (≈)	1.82E-01	1.354E+00 (≈)	2.58E+00	5.034E+03 (−)	1.04E+04	8.690E+07 (−)	4.77E+08	1.165E+00	2.06E+00
4	0.000E+00 (+)	0.00E+00	1.227E+02 (−)	1.60E+02	0.000E+00 (+)	0.00E+00	5.194E+03 (−)	2.50E+03	1.857E-03	1.22E-02
5	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	3.718E-01 (−)	1.73E+00	0.000E+00	0.00E+00
6	1.925E-01 (+)	1.37E+00	8.081E+00 (−)	3.78E+00	3.848E-01 (+)	1.92E+00	1.248E+01 (−)	1.26E+01	5.388E+00	4.93E+00
7	6.351E-04 (+)	4.05E-04	1.824E-03 (+)	4.57E-03	2.328E+00 (−)	1.89E+00	6.393E+00 (−)	4.90E+00	3.035E-02	7.01E-02
8	2.036E+01 (≈)	6.22E-02	2.036E+01 (≈)	7.55E-02	2.036E+01 (≈)	7.68E-02	2.035E+01 (≈)	7.57E-02	2.036E+01	8.07E-02
9	1.664E-01 (+)	3.84E-01	4.087E+00 (−)	1.97E+00	5.055E+00 (−)	7.55E-01	2.360E+00 (−)	1.19E+00	8.203E-01	8.04E-01
10	3.803E-01 (−)	1.40E-01	9.641E-02 (−)	7.53E-02	1.485E-01 (−)	3.33E-02	7.894E-01 (−)	1.75E+00	4.337E-02	2.62E-02
11	1.701E+01 (−)	4.12E+00	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	1.951E-02 (≈)	1.39E-01	0.000E+00	0.00E+00
12	2.544E+01 (−)	4.11E+00	1.507E+01 (−)	3.47E+00	1.430E+01 (−)	2.31E+00	7.863E+00 (−)	2.70E+00	6.276E+00	2.17E+00
13	2.561E+01 (−)	5.05E+00	1.601E+01 (−)	4.87E+00	1.476E+01 (−)	3.99E+00	1.368E+01 (−)	6.83E+00	8.334E+00	4.30E+00
14	1.024E+03 (−)	1.41E+02	7.194E-01 (−)	6.47E-01	2.449E-03 (≈)	1.22E-02	1.103E-02 (−)	2.71E-02	0.000E+00	0.00E+00
15	1.305E+03 (−)	1.47E+02	1.104E+03 (−)	1.69E+02	9.231E+02 (+)	1.43E+02	4.841E+02 (+)	1.74E+02	1.019E+03	1.44E+02
16	1.010E+00 (+)	1.81E-01	1.109E+00 (≈)	2.07E-01	9.865E-01 (+)	1.68E-01	8.612E-01 (+)	1.75E-01	1.114E+00	1.96E-01
17	2.924E+01 (−)	3.10E+00	1.048E+01 (−)	1.50E-01	1.012E+01 (≈)	0.00E+00	1.014E+01 (−)	6.59E-02	1.012E+01	0.00E+00
18	3.611E+01 (−)	4.28E+00	3.426E+01 (−)	3.60E+00	3.423E+01 (−)	3.80E+00	2.091E+01 (+)	3.78E+00	3.087E+01	3.07E+00
19	2.009E+00 (−)	3.29E-01	5.103E-01 (−)	7.94E-02	4.203E-01 (≈)	5.64E-02	1.793E-01 (+)	1.13E-01	4.353E-01	6.14E-02
20	2.519E+00 (+)	2.47E-01	2.836E+00 (−)	2.24E-01	2.826E+00 (−)	2.42E-01	2.592E+00 (≈)	4.04E-01	2.638E+00	3.05E-01
21	3.688E+02 (≈)	7.35E+01	3.590E+02 (≈)	8.53E+01	3.080E+02 (+)	1.09E+02	3.795E+02 (≈)	5.96E+01	3.708E+02	7.57E+01
22	1.041E+03 (−)	2.06E+02	4.797E+01 (−)	2.79E+01	2.120E+01 (−)	6.91E+00	1.201E+01 (+)	8.71E+00	1.586E+01	9.98E+00
23	1.198E+03 (−)	1.47E+02	1.146E+03 (−)	2.00E+02	9.789E+02 (≈)	1.85E+02	5.985E+02 (+)	2.07E+02	9.713E+02	1.90E+02
24	1.966E+02 (≈)	1.83E+01	1.895E+02 (≈)	2.54E+01	1.971E+02 (−)	2.61E+01	1.838E+02 (≈)	3.61E+01	1.911E+02	2.86E+01
25	2.005E+02 (≈)	1.48E+00	1.963E+02 (≈)	1.72E+01	1.982E+02 (−)	2.53E+01	1.981E+02 (−)	1.88E+01	1.955E+02	2.06E+01
26	1.362E+02 (−)	2.83E+01	1.223E+02 (−)	9.88E+00	1.156E+02 (−)	4.38E+00	1.162E+02 (−)	1.90E+01	1.073E+02	5.58E+00
27	3.078E+02 (−)	2.72E+01	3.000E+02 (≈)	2.62E-03	3.274E+02 (−)	4.51E+01	3.170E+02 (−)	3.44E+01	3.020E+02	1.40E+01
28	2.882E+02 (≈)	4.75E+01	2.765E+02 (≈)	6.51E+01	2.569E+02 (≈)	8.31E+01	3.082E+02 (−)	6.76E+01	2.804E+02	6.01E+01
—	13		16		13		17		(−)	
—	8		11		9		5		(≈)	
—	7		1		6		6		(+)	

Table 2: Success-rate on functions for $d=10$.

f	1	2	3	4	5	6	7 ∼ 9	10	11	12, 13	14	15 ∼ 21	22	23 ∼ 28
DE	100%	100%	2%	100%	100%	84%	0%	0%	0%	0%	0%	0%	0%	0%
$\text{DE}_{(\text{aDE})}$	100%	0%	0%	0%	100%	0%	0%	2%	100%	0%	2%	0%	0%	0%
$\text{DE}_{(\text{DER9})}$	100%	100%	0%	100%	100%	92%	0%	0%	100%	0%	96%	0%	0%	0%
$\text{DE}_{(\text{SLADE})}$	76%	0%	0%	0%	80%	0%	0%	0%	98%	0%	73%	0%	2%	0%
$\text{DE}_{(\text{PBPS})}$	100%	8%	0%	8%	100%	8%	0%	6%	100%	0%	100%	0%	0%	0%

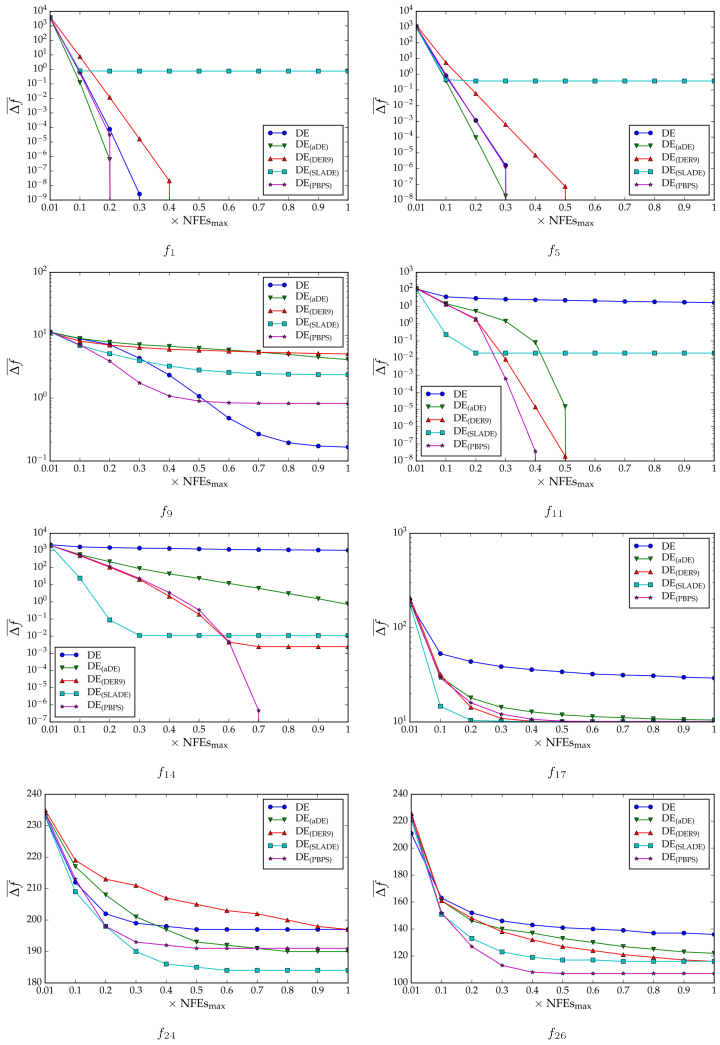


Figure 3: Convergence behaviour on several chosen functions for $d = 10$.

Table 3: Complexity in terms of execution times on function 13 for $d = 10$.

Timing	$\overline{T}_2$	T_1	T_0	$C = (\overline{T}_2 - T_1)/T_0$
DE	0.7086 s	0.5930 s	0.0699 s	1.6526
$\text{DE}_{(\text{aDE})}$	0.7198 s	0.5914 s	0.0699 s	1.8381
$\text{DE}_{(\text{DER9})}$	0.7388 s	0.5893 s	0.0698 s	2.1399
$\text{DE}_{(\text{SLADE})}$	0.7466 s	0.5974 s	0.0701 s	2.1296
$\text{DE}_{(\text{PBPS})}$	0.7441 s	0.5966 s	0.0694 s	2.1262

Table 4: Experimental results on functions for $d=30$.

f	DE		$\text{DE}_{(\text{aDE})}$		$\text{DE}_{(\text{DER9})}$		$\text{DE}_{(\text{SLADE})}$		$\text{DE}_{(\text{PBPS})}$	
	mean	std. dev.	mean	std. dev.	mean	std. dev.	mean	std. dev.	mean	std. dev.
1	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	2.742E+01 (−)	1.19E+02	0.000E+00	0.00E+00
2	4.306E+05 (−)	3.31E+05	1.241E+06 (−)	1.06E+06	7.811E+04 (+)	4.53E+04	2.138E+07 (−)	1.29E+07	2.148E+05	1.45E+05
3	1.193E+00 (+)	5.07E+00	1.365E+06 (+)	2.00E+06	7.382E+07 (−)	7.18E+07	3.721E+09 (−)	7.54E+09	2.773E+06	3.42E+06
4	9.014E+02 (−)	4.45E+02	1.023E+04 (−)	5.49E+03	9.056E+00 (≈)	1.15E+01	3.800E+04 (−)	1.05E+04	1.495E+01	2.35E+01
5	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	8.656E+01 (−)	2.47E+02	0.000E+00	0.00E+00
6	1.030E+01 (+)	6.11E+00	2.030E+01 (≈)	1.67E+01	9.716E+00 (+)	5.62E+00	8.115E+01 (−)	4.24E+01	1.678E+01	8.45E+00
7	8.107E-02 (+)	1.28E-01	1.372E+00 (+)	1.07E+00	5.404E+01 (−)	9.66E+00	5.947E+01 (−)	2.20E+01	1.657E+01	8.26E+00
8	2.095E+01 (−)	4.59E-02	2.095E+01 (≈)	4.13E-02	2.094E+01 (≈)	5.61E-02	2.094E+01 (≈)	5.20E-02	2.095E+01	5.50E-02
9	2.419E+01 (−)	1.48E+01	3.107E+01 (−)	2.23E+00	2.904E+01 (−)	1.61E+00	2.155E+01 (−)	3.45E+00	1.484E+01	4.18E+00
10	5.992E-03 (+)	5.43E-03	9.204E-02 (≈)	5.26E-02	6.148E-03 (+)	7.10E-03	5.604E+01 (−)	1.01E+02	8.818E-02	4.34E-02
11	1.300E+02 (−)	2.49E+01	0.000E+00 (≈)	0.00E+00	0.000E+00 (≈)	0.00E+00	4.679E+00 (−)	2.49E+01	0.000E+00	0.00E+00
12	1.819E+02 (−)	8.75E+00	8.682E+01 (−)	1.88E+01	1.095E+02 (−)	1.76E+01	7.666E+01 (−)	2.97E+01	3.965E+01	1.30E+01
13	1.819E+02 (−)	9.66E+00	1.330E+02 (−)	2.24E+01	1.406E+02 (−)	1.84E+01	1.239E+02 (−)	2.63E+01	7.411E+01	2.01E+01
14	6.323E+03 (−)	4.65E+02	1.236E+02 (−)	7.60E+01	5.307E-03 (≈)	1.09E-02	1.032E+00 (−)	4.85E+00	3.674E-03	9.03E-03
15	7.106E+03 (−)	2.70E+02	5.610E+03 (≈)	5.50E+02	4.730E+03 (+)	2.95E+02	3.277E+03 (+)	5.34E+02	5.509E+03	3.24E+02
16	2.474E+00 (≈)	2.53E-01	2.370E+00 (≈)	3.54E-01	2.032E+00 (+)	2.83E-01	1.555E+00 (+)	2.74E-01	2.356E+00	2.79E-01
17	1.826E+02 (−)	1.81E+01	3.701E+01 (−)	3.72E+00	3.043E+01 (≈)	0.00E+00	3.077E+01 (−)	7.89E-01	3.043E+01	1.40E-06
18	2.112E+02 (−)	9.74E+00	1.863E+02 (−)	1.74E+01	1.971E+02 (−)	1.44E+01	1.041E+02 (+)	1.56E+01	1.692E+02	1.13E+01
19	1.510E+01 (−)	8.94E-01	2.224E+00 (−)	3.67E-01	1.819E+00 (≈)	1.61E-01	4.103E+00 (−)	1.93E+01	1.809E+00	1.37E-01
20	1.209E+01 (−)	2.65E-01	1.207E+01 (−)	4.06E-01	1.200E+01 (−)	2.70E-01	1.099E+01 (+)	8.05E-01	1.167E+01	3.63E-01
21	3.229E+02 (≈)	7.89E+01	2.875E+02 (≈)	7.29E+01	3.525E+02 (−)	1.04E+02	4.965E+02 (−)	3.64E+02	2.987E+02	8.44E+01
22	6.251E+03 (−)	5.79E+02	2.720E+02 (−)	9.50E+01	8.085E+01 (+)	2.82E+01	1.075E+02 (+)	2.42E+01	1.160E+02	2.03E+01
23	7.088E+03 (−)	2.82E+02	6.205E+03 (−)	5.07E+02	5.158E+03 (+)	3.60E+02	3.775E+03 (+)	6.76E+02	5.802E+03	4.46E+02
24	2.000E+02 (+)	1.98E-02	2.048E+02 (+)	4.88E+00	2.687E+02 (−)	6.02E+00	2.469E+02 (−)	1.40E+01	2.179E+02	7.48E+00
25	2.400E+02 (+)	4.28E+00	2.492E+02 (+)	1.11E+01	2.910E+02 (−)	5.02E+00	2.750E+02 (−)	1.20E+01	2.537E+02	5.63E+00
26	2.039E+02 (−)	1.96E+01	2.021E+02 (−)	1.45E+01	2.000E+02 (+)	2.92E-03	2.018E+02 (−)	2.84E+00	2.000E+02	6.76E-03
27	3.049E+02 (+)	2.92E+01	3.753E+02 (+)	9.80E+01	1.043E+03 (−)	3.82E+01	8.634E+02 (−)	9.95E+01	5.543E+02	1.26E+02
28	3.000E+02 (≈)	0.00E+00	3.000E+02 (≈)	0.00E+00	3.000E+02 (≈)	0.00E+00	7.634E+02 (−)	6.00E+02	3.000E+02	0.00E+00
—	16		13		11		21		(−)	
—	5		10		9		1		(≈)	
—	7		5		8		6		(+)	

Table 5: Success-rate on functions for $d=30$.

f	1	$2 \sim 4$	5	$6 \sim 9$	10	11	12, 13	14	$15 \sim 28$
DE	100%	0%	100%	0%	41%	0%	0%	0%	0%
$\text{DE}_{(\text{aDE})}$	100%	0%	100%	0%	0%	100%	0%	0%	0%
$\text{DE}_{(\text{DER9})}$	100%	0%	100%	0%	2%	100%	0%	78%	0%
$\text{DE}_{(\text{SLADE})}$	37%	0%	6%	0	0%	37%	0%	2%	0%
$\text{DE}_{(\text{PBPS})}$	100%	0%	100%	0%	0%	100%	0%	84%	0%

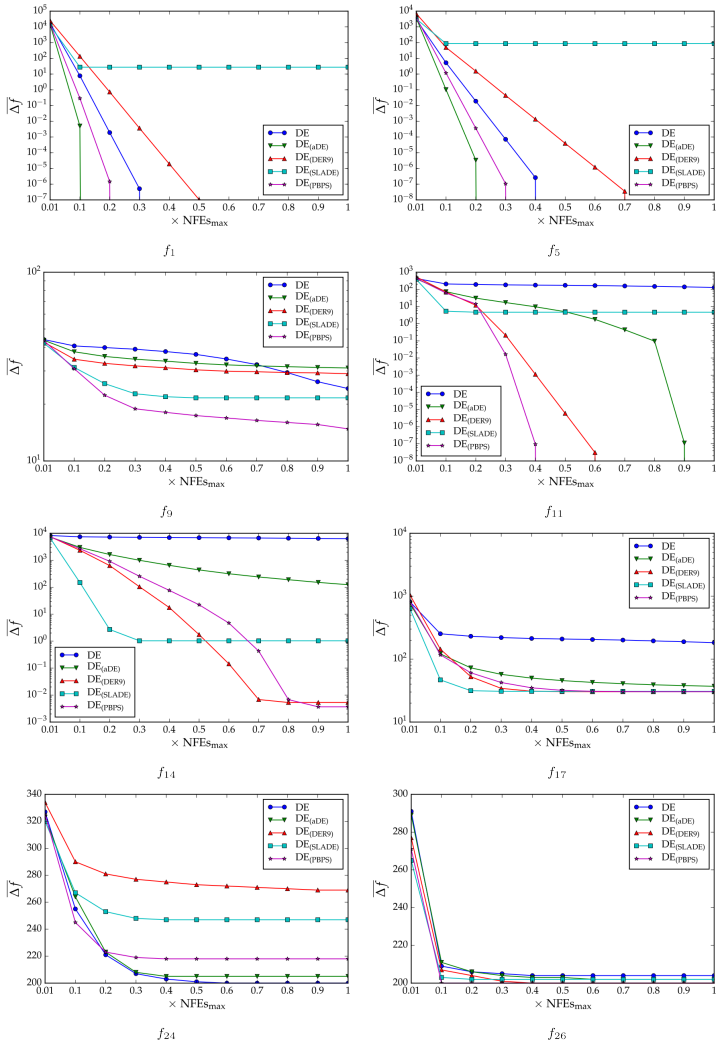


Figure 4: Convergence behaviour on several chosen functions for $d=30$.

Table 6: Complexity in terms of execution times on function 13 for $d=30$.

Timing	$\overline{T}_2$	T_1	T_0	$C = (\overline{T}_2 - T_1)/T_0$
DE	2.8882 s	2.5804 s	0.0700 s	4.3995
DE$_{(aDE)}$	2.9027 s	2.5717 s	0.0699 s	4.7350
DE$_{(DER9)}$	2.9882 s	2.5931 s	0.0715 s	5.5254
DE$_{(SLADE)}$	2.9089 s	2.5887 s	0.0703 s	4.5523
DE$_{(PBPS)}$	2.9598 s	2.5876 s	0.0705 s	5.2804

References

[1] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10:646–657, 2006.

[2] L. Coelho, T. Bora, and L. Lebensztajn. A chaotic approach of differential evolution optimization applied to loudspeaker design problem. *IEEE Transactions on Magnetics*, 48:751–754, 2012.

[3] S. Das, A. Konar, and U. K. Chakraborty. Two improved differential evolution schemes for faster global search. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 991–998, 2005.

[4] M. Dorigo and T. Stützle. *Ant colony optimization*. Bradford Company, 2004.

[5] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141, 1999.

[6] A. Glotić and A. Zamuda. Short-term combined economic and emission hydrothermal optimization by surrogate differential evolution. *Applied Energy*, 141:42–56, 2015.

[7] S. Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42:482–500, 2012.

[8] G. Karafotias, M. Hoogendoorn, and A. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19:167–187, 2015.

[9] H.-K. Kim, J.-K. Chong, K.-Y. Park, and D. Lowther. Differential evolution strategy for constrained global optimization and application to practical engineering problems. *IEEE Transactions on Magnetics*, 43:1565–1568, 2007.

[10] P. Korošec and J. Šilc. Applications of the differential ant-stigmergy algorithm on real-world continuous optimization problems. In W. P. dos Santos (Ed.) *Evolutionary Computation*, InTech, 2009.

[11] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernández-Díaz. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report, 201212, 2013.

[12] G. Martinović, D. Bajer, and B. Zorić. A differential evolution approach to dimensionality reduction for classification needs. *International Journal of Applied Mathmatics and Computer Science*, 24:111–122, 2014.

[13] B. C. Moahn and R. Baskaran. A survey: Ant Colony Optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, 39:4618–4627, 2012.

[14] F. Neri and V. Tirronen. Recent Advances in Differential Evolution: A Survey and Experimental Analysis. *Artificial Intelligence Review*, 33:61–106, 2010.

[15] N. Noman, D. Bollegala, and H. Iba. An adaptive differential evolution algorithm. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2229–2236, 2011.

[16] R. C. Pedrosa Silva, R. A. Lopes, and F. G. Guimarães. Self-adaptive mutation in the differential evolution. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1939–1946, 2011.

[17] K. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, New York, 2005.

[18] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *Eurpean Journal of Operation Research*, 185:1155–1173, 2008.

[19] R. Storn and K. Price. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimzation*, 11:341–359, 1997.

[20] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for Differential Evolution. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 71–78, 2013.

[21] J. Tvrdík. Competitive Differential Evolution. *Proceedings of the International Conference on Soft Computing MENDEL*, pages 7–12, 2006.

[22] W.-J. Yu and J. Zhang. Adaptive differential evolution with optimization state estimation. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1285–1292, 2012.

[23] J. Zhang and A. C. Sanderson. JADE: Adaptive differential evolution with optional external archive. *IEEE Transasctions of Evolutionary Computation*, 13:945–958, 2009.

[24] Z. Zhao, J. Yang, Z. Hu, and H. Che. A differential evolution algorithm with self-adaptive strategy and control parameters based on symmetric Latin hypercube design for unconstrained optimization problems. *European Journal of Operation Research*, 250:30–45, 2016.

EXPERIMENTAL ALGORITHMICS APPLIED TO ON-LINE MACHINE LEARNING

Thomas Bartz-Beielstein
SPOTSeven Lab, TH Köln, Gummersbach, Germany
thomas.bartz-beielstein@th-koeln.de

Abstract The application of methods from experimental algorithmics to on-line or streaming data is referred to as experimental algorithmics for streaming data (EADS). This paper proposes an experimental methodology for on-line machine learning algorithms, i.e., for algorithms that work on data that are available in a sequential order. It is demonstrated how established tools from experimental algorithmics can be applied in the on-line or streaming data setting. The massive on-line analysis framework is used to perform the experiments. Benefits of a well-defined report structure are discussed.

Keywords: Experimental algorithmics, Massive on-line analysis, On-line machine learning, Streaming data.

1. Introduction: Experimental Algorithmics

This article is devoted to the question "Why is an experimental methodology necessary for the analysis of on-line algorithms?" We will mention two reasons to motivate the approach presented in this paper. First, without a sound methodology, there is the danger of generating arbitrary results, i.e., results that happened by chance; results that are not reproducible; results that depend on the seed of a random number generator; results that are statistically questionable; results that are statistically significant, but scientifically meaningless; results that are not generalizable; etc. Second, experiments are the cornerstone of the scientific method. Even the discovery of scientific highly relevant results is of no use, if they remain unpublished or if they are published in an incomprehensible manner. Discussion is the key ingredient of modern science.

Experimental algorithmics (EA) uses empirical methods to analyze and understand the behavior of algorithms. Experimental algorithmics evolved over the last three decades and provides tools for sound experimental analysis of algorithms. Main contributions, which influenced the field of EA are McGeoch's thesis "Experimental Analysis of Algorithms" [19], the experimental evaluation of simulated annealing by Johnson et al. [18], and the article about designing and reporting computational experiments with heuristic methods from Barr et al. [1]. And, Hooker's papers with the striking titles "Needed: An empirical science of algorithms" and "Testing Heuristics: We Have It All Wrong" [15, 16], which really struck a nerve. Theoreticians recognized that their methods can benefit from experimental analysis and the discipline of *algorithm engineering* was established [8]. Parameter tuning methods gained more and more attention in the *machine learning* (ML) and *computational intelligence* (CI) communities. Eiben and Jelasity's "Critical Note on Experimental Research Methodology in EC" [11] enforced the discussion. This increased awareness resulted in several tutorials, workshops, and special sessions devoted to experimental research in evolutionary computation. Results from these efforts are summarized in the collection "Experimental Methods for the Analysis of Optimization Algorithms" [2].

This overview is by far not complete, and several important publications are missing. However, it illustrates the development of an emerging field and its importance. The standard approach described so far focuses on relatively small, static data sets that can be analyzed off-line. We propose an extension of EA to the field of stream data, which will be referred to as *experimental algorithmics for streaming data* (EASD). This extension is motivated by the enormous growth of data in the last decades. Machine learning, i.e., automatically extract information from data, was considered the solution to the immense increase of data. The field of *data mining* evolved to handle data that does not fit into working memory: Data mining became popular, because it provides tools for very large, but static data sets. Models cannot be updated when new data arrives. Nowadays, data are collected in nearly every device—massive, data streams are ubiquitous. Especially, industrial production processes generate huge and dynamic data. This leads to the development of the *data stream paradigm*. Bifet et al. [5] describe core assumptions of data stream processing as follows:

(S-1) The training examples can be briefly inspected a single time only.

(S-2) The data arrive in a high speed stream.

(S-3) Because the memory is limited, data must be discarded to process the next examples.

(S-4) The order of data arrival is unknown.

(S-5) The model is updated incrementally, i.e., directly when a new data arrives.

(S-6) Anytime property: The model can be applied at any point between training examples.

(S-7) Last but not least: theory is nice, but empirical evidence of algorithm performance is necessary.

We will develop an experimental methodology for on-line machine learning, i.e., for situations in which data becomes available in a sequential order. The data is used to update the predictor for future data at each step. On-line learning differs from traditional batch learning techniques, which generate the predictor by learning on the entire training data set at once. The terms "on-line" and "data stream" will be used synonymously in the following.

This paper is structured as follows. Section 2 compares the traditional batch setting with the stream data setting. How to assess model performance is described in Section 3. A simple experiment, which exemplifies the EASD approach, is presented in Sec. 4. This article concludes with a summary in Section 5.

2. Batch Versus Stream Classification

By comparing the traditional batch and the stream classification, the following observations can be made: Both classification procedures partition the data into test and training set. In contrast to batch classification, stream classification is a cyclic process, which uses nonpermanent data. The elementary steps used in both settings are illustrated in Fig. 1. The figure is based on the data stream classification cycle in Bifet et al. [6].

The batch classification cycle processes data as follows:

(CB-1) Input, i.e., the algorithm receives the data.

(CB-2) Learn, i.e., the algorithm processes the data and generates its own data structures (builds a model).

(CB-3) Predict, i.e., the algorithm predicts the class of unseen data using the test set.

Data availability differs in the stream classification cycle. Additional restrictions have to be considered [6]. Freely adapted from Bifet et al. [6], the data stream processing can be described as follows:

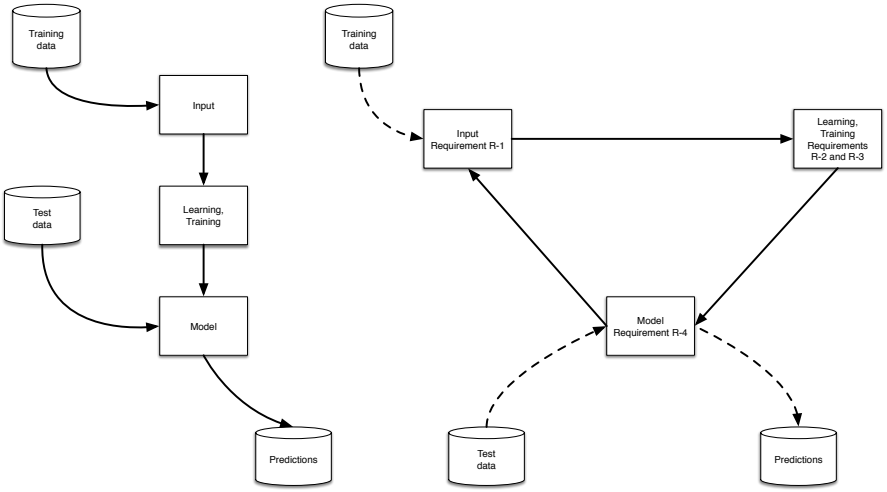


Figure 1: Left: The batch classification cycle. *Right:* The stream classification cycle. Dotted lines represent nonpermanent data. Both classification cycles partition the data into test and training set. To keep the illustration simple, this split is not shown.

(CS-1) Input, i.e., the algorithm receives the next data from the stream. At his stage of the process, the *process only once* (R-1) requirement has to be considered: Data stream data is accepted as they arrive. After inspection, the data is not available any more. However, the algorithm itself is allowed to set up an archive (memory).

(CS-2) Learn, i.e., the algorithm processes the data and updates its own data structures (updates the model). The limited memory and limited time requirements (R-2) and (R-3), respectively, have to be considered. Data stream algorithms allow processing data that are several times bigger than the working memory and real-time processing requires that the algorithm process the data quickly (or even faster) than they arrive.

(CS-3) Predict, i.e., the algorithm is able to receive the next data. It is also able to predict the class of unseen data. The *predict at any point* requirement has to be considered. The best model should be generated as efficiently as possible.

3. Assessing Model Performance

Elementary performance criteria for data stream algorithms are based on

(P-1) Time (speed): We consider the amount of time needed (i) to learn and (ii) to predict. If the time limit is reached, continuing the data processing will take longer or results will loose precision. This consequence is not so hard as the space limit, because overriding the space limit will force the algorithm to stop.

(P-2) Space (memory): A simple strategy for the handling space budget is to stop once the limit is reached. To continue processing if the the space limit is reached is to force the algorithm to discard parts of its data.

(P-3) Error rates (statistical measures): The prediction error is considered. Several error measures are available [17, 26].

A contingency table or *confusion matrix* is a standard methods to summarize results. Based on the values from the confusion matrix, the *accuracy* can be determined as the percentage of correct classifications, i.e., it is defined as the sum of the number of true positives and the number of true negatives divided by the total number of examples (total population). Since accuracy can be misleading (consider the so-called accuracy paradox), further measures are commonly used [27]: For example, the *precision* is defined as the number of true positives divided by the number of true positives and false positives. Precision is also referred to as the *positive predictive value* (PPV). Or, the *negative predictive value* (NPV) is defined as the number of true negatives divided by the number of true negatives and false negatives. The *specifity* (or true negative rate, TNR) is defined as the number of true negatives divided by the number of true negatives and false positives. And, the *sensitivity* (or true positive rate (TPR) or *recall*) is defined as the number of true positives divided by the number of true positives and the number of false negatives. Using training data to measure these statistical measures can lead to overfitting and result in poorly generalizable models. Therefore, testing data, i.e., using unseen data, should be used [13].

Generating test data appears to be trivial at the first sight. However, simply splitting the data into two sets might cause unwanted effects, e.g., introduce bias, or result in an inefficient usage of the available information. The test data generation process needs careful considerations in order to avoid these fallacies. In the dynamic data stream setting, plenty of data is available. The simple holdout strategy can be used without causing the problems mentioned in the batch setting. In contrast to the batch settings, large data sets for exact accuracy estimations can be used for testing without problems. The simplest approach is just holding out one (large) single reference data set during the whole learning (training)

phase. Using this holdout data set, the model can be evaluated periodically. A graphical plot (accuracy versus number of training samples) is the most common way presenting results. Very often, the comparison of two algorithms is based on graphical comparisons by visualizing trends (e.g., accuracy) over time [6]. To obtain reliable results, statistical measures such as the standard error of results, are recommended. The statistical analysis should be accompanied by a comprehensive reporting scheme, which includes the relevant details for understanding and possible replication of the findings [23]. Only a few publications that perform an extensive statistical analysis are available [10]. Fortunately, the open source framework for data stream mining MOA is available and provides tools for an extensive experimental analysis [14].

Simulators for Data Stream Analysis. Random data stream simulators are a valuable tool for the experimental analysis of data stream algorithms. For our experiments in Section 4 a static data set, which contained pre-assigned class labels, was used to simulate a real-world data stream environment. The open source framework for data stream mining MOA [14] is able to generate a few thousand examples up to several hundred thousand examples per second. Additional noise can slow down the speed, because it requires the generation of random numbers.

4. A Simple Experiment in the EASD Framework

The Scientific Question. Before experimental runs are performed, the scientific question, which motivates the experiments, should be clearly stated. To exemplify the EASD approach, the following task is considered: Machine learning methods, which combine multiple models to improve prediction accuracy, are called *ensemble data mining algorithms*. Diversity of the different models is necessary to reach this performance gain compared to individual models. Each individual ML algorithm requires the specification of some parameters. Building ensembles requires the specification of additional algorithm parameters, e.g., the number of ensemble members. The scientific question can be formulated as follows: "How does the number of models to boost affect the performance of online learning algorithms?" To set up experiments, a specific algorithm (or a set of algorithms) has to be selected. Oza et al. presented a simple on-line bagging and boosting algorithm, OzaBoost [20]. The effect of the number of models to boost on the algorithm performance is an important research question, which will be analyzed in the following experiments.

Therefore, the experimental setup as well as the implementation details have to be specified further.

Implementation Details. Our observations are based on the *Massive On-line Analysis* (MOA) framework for data stream mining. The MOA framework provides programs for evaluation of ML algorithms [14, 6]. We will consider a typical classification setting, which can transferred into a regression setting without any fundamental changes. The model is trained on data with known classes. In the MOA classification setting, the following assumptions are made by Bifet et al. [6]:

(i) Small and fixed number of variables,

(ii) large number of examples,

(iii) limited number of possible class labels, typically less than ten,

(iv) the size of the training data will not fit into working memory,

(v) the available time for training is restricted,

(vi) the available time for classification is restricted, and

(vii) drift can occur.

We used *OzaBoost*, the incremental on-line boosting of Oza and Russel [21], which was implemented in Version 12.03 of the MOA software environment [14]. OzaBoost uses the following parameters: The classifier to train, l, the number of models to boost. s, and the option to boost with weights only, p. Experiments were performed in the statistical programming environment R [24]. The *sequential parameter optimization toolbox* (SPOT) was used for the experimental setup [3]. SPOT is implemented as an R package [4]. An additional R package, RMOA, was written to make the classification algorithms of MOA easily available to R users. The RMOA package is available on github (`https://github.com/jwijffels/RMOA`).

Empirical Analysis. The number of models to boost will be referred to as s in the following. In addition to s, the classifier to train will be modified as well. It will be referred to as l. Therefore, two algorithm parameters will be analyzed. The accuracy was used as a performance indicator. Using this setting, the scientific question can be concretized as the following research question: "How does the number of models to boost, s, affect the performance of the OzaBoost algorithm?"

Optimization Problem. After the algorithm was specified, a test function, e.g., an optimization problem, or a classification task, has to be defined. MOA provides tools to generate data streams. To keep the setup simple, we used the iris data set [12], which is a available as an R dataset [24].

Pre-experimental Planning. Before experimental runs are started, it is important to calibrate the problem difficulty to avoid floor- and ceiling effects. We will perform a comparison with a simple algorithm. If the simple algorithm is able to find the optimal solution with a limited computational budget, then the experimental setup is not adequate (too easy). If the simple algorithm is not able to find any solution, this may indicate that the problem is too hard. The naive Bayes classifier, which was used as the simple algorithm, obtained an accuracy of 91 percent. Because no floor- or ceiling effects were observed, we continue our experimentation with the OzaBoost algorithm.

Task and Experimental Setup. Two parameters of the OzaBoost algorithm were analyzed:

(i) the base learner, l, and

(ii) the ensemble size, s.

Hoeffding trees were used as base learners in our experiments [22]. In addition to the standard Hoeffding tree [10], a random Hoeffding tree was used in out experiments as a base learner. To be more specific, the categorical variable l was selected from the set {`HoeffdingTree`, `RandomHoeffdingTree`}, see Bifet et al. [7] for details. Values between one and one hundred were used for the ensemble size s.

Results and Visualizations. A comparison of the mean values from the two learners shows a significant difference: the first learner, i.e., `RandomHoeffdingTree`, obtained a mean accuracy of 0.66 (standard deviation (s.d.) = 0.06), whereas the mean accuracy of the second learner, i.e., `HoeffdingTree`, is 0.81 (s.d. = 0.18). The distributions of the accuracies are plotted in Fig. 2 and provide a detailed presentation of the results. Although the mean values of the two learners are different, the standard deviation of the `HoeffdingTree` learner approximately three times higher than the standard deviation of the `RandomHoeffdingTree`. This is reflected in the plots: the `HoeffdingTree` algorithm is not able to find an acceptable classification in some experimental runs.

Therefore, an additional analysis of the relationship between ensemble size and accuracy for the `HoeffdingTree` learner is of interest. We plot

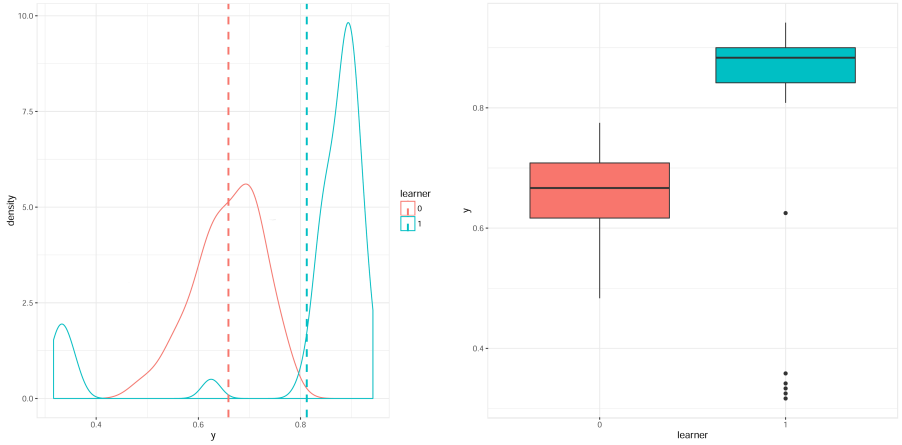


Figure 2: Comparison of the two learners. 0 = `RandomHoeffdingTree`, 1 = `HoeffdingTree`. *Left:* Density plots (accuracy, y). The dotted lines represent the mean values of the corresponding learners. *Right:* Boxplots (accuracy). Same data as in the panel on the left were used in the boxplots. The comparison of these two plots nicely illustrates strength and weakness of the plotting methods.

the results from the `HoeffdingTree` learner and add a smooth curve computed by `loess` (LOcal regrESSion) to a scatter plot [9]. `loess` fitting is done locally, i.e., the fit at a point x is based on points from a neighborhood of x, weighted by their distance from x. The result is shown in Fig. 3. This plot indicates that outliers occur if the sample size is small, i.e., $s < 60$.

Observations. The results reveal that the `HoeffdingTree` learner performs better (on average) than the `RandomHoeffdingTree` learner, but appears to be more sensitive to the settings of the ensemble size.

Discussion. The selection of a suitable base learner is important. Results indicate that too small ensemble sizes worsen the algorithm's performance. This statement has to be investigated further, e.g., by finding improved parameter settings for the OzaBoost learner. The sequential parameter optimization framework can be used for tuning the learner. A typical result from this tuning procedure is shown in the right panel of Fig. 3. In this plot, the accuracy, which was obtained by OzaBoost, is plotted against the number of iterations of the SPO tuner.

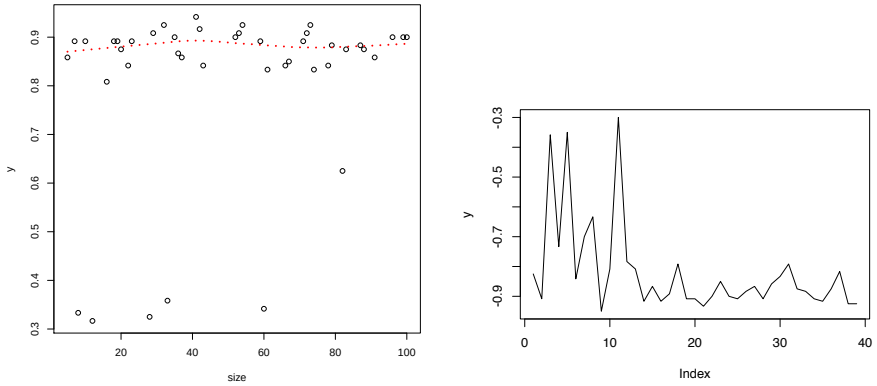


Figure 3: Left: Results, i.e., accuracy (y), obtained with the `HoeffdingTree` learner ($l = 1$) plotted against ensemble size (s). *Right:* A typical result from the parameter tuning procedure. Accuracy (y) is plotted against the number of algorithm runs (Index). The negative accuracy is shown, because the tuner requires minimization problems.

5. Summary and Outlook

An experimental methodology for the analysis of on-line data was presented. Differences between the traditional batch setting and the on-line setting were emphasized. Although useful, the actual practice of comparing run-time plots of on-line algorithms, e.g., accuracy versus time, should be complemented by more advanced tools from *exploratory data analysis* [25] and statistical tools, which were developed for the analysis of traditional algorithms. It was demonstrated, that statistical methods from experimental algorithmics can be successfully applied in the on-line setting. A combination of MOA, RMOA and the SPO toolbox was used to demonstrate the applicability and usefulness of standard tools from experimental algorithmics. The design and analysis of the algorithm were performed in the EASD framework. This report methodology, which is also described and exemplified by Preuss [23], is an integral part of the EASD framework.

References

[1] R. Barr, B. Golden, J. Kelly, M. Rescende, and W. Stewart. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1(1):9–32, 1995.

[2] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss (Eds.) *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin,

Heidelberg, New York, 2010.

[3] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The Sequential Parameter Optimization Toolbox. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss (Eds.) *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Heidelberg, New York, 2010.

[4] T. Bartz-Beielstein and M. Zaefferer. SPOT Package Vignette. Technical report, 2011.

[5] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[6] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Data Stream Mining. pages 1–185, May 2011.

[7] A. Bifet, R. Kirkby, P. Kranen, and P. Reutemann. *Massive online analysis - Manual*, Mar. 2012.

[8] G. Cattaneo and G. Italiano. Algorithm engineering. *ACM Computing Surveys*, 31(3):3, 1999.

[9] J. M. Chambers and T. J. Hastie (Eds.). *Statistical Models in S*. Statistical Models in S. Wadsworth and Brooks/Cole, Pacific Grove, CA, 1992.

[10] P. Domingos and G. Hulten. Mining high-speed data streams. *Proceedings of the the Sixth ACM SIGKDD International Conference*, pages 71–80, 2000.

[11] A. E. Eiben and M. Jelasity. A Critical Note on Experimental Research Methodology in EC. *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 582–587, 2002.

[12] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[13] T. Hastie. *The elements of statistical learning : data mining, inference, and prediction*. Springer, New York, 2nd ed., 2009.

[14] G. Holmes, B. Pfahringer, P. Kranen, T. Jansen, T. Seidl, and A. Bifet. MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering. *Proceedings of the International Workshop on Handling Concept Drift in Adaptive Information Systems in conjunction with European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 3–16, 2010.

[15] J. N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42(2):201–212, 1994.

[16] J. N. Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, 1(1):33–42, 1996.

[17] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[18] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: an Experimental Evaluation. Part I, Graph Partitioning. *Operations research*, 37(6):865–892, 1989.

[19] C. C. McGeoch. Experimental Analysis of Algorithms. PhD thesis, Carnegie Mellon University, Pittsburgh PA, 1986.

[20] N. C. Oza and S. Russell. Experimental comparisons of online and batch versions of bagging and boosting. *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 359–364, 2001.

[21] N. C. Oza and S. Russell. Online bagging and boosting. *Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics*, pages 105–112, 2001.

[22] B. Pfahringer, G. Holmes, and R. Kirkby. New Options for Hoeffding Trees. In *AI 2007: Advances in Artificial Intelligence*, pages 90–99. Springer, Berlin Heidelberg, 2007.

[23] M. Preuss. *Multimodal Optimization by Means of Evolutionary Algorithms*. Natural Computing Series. Springer International Publishing, Cham, 2015.

[24] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[25] J. W. Tukey. *Explorative data analysis*. Addison-Wesley, 1977.

[26] C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(7982):1–4, 2005.

[27] X. Zhu. *Knowledge Discovery and Data Mining: Challenges and Realities: Challenges and Realities*. Gale virtual reference library. Information Science Reference, 2007.

DISADVANTAGES OF STATISTICAL COMPARISON OF STOCHASTIC OPTIMIZATION ALGORITHMS

Tome Eftimov
Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
tome.eftimov@ijs.si

Peter Korošec
Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
peter.korosec@ijs.si

Barbara Koroušić Seljak
Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
barbara.korousic@ijs.si

Abstract In this paper a short overview and a case study in a statistical comparison of stochastic optimization algorithms are presented. The algorithms are part of the Black-Box Optimization Benchmarking 2015 competition that was held at the 5th GECCO Workshop for Real-Parameter Optimization. The question about the difference between parametric and non-parametric tests for single-problem analysis and for multiple-problem analysis is addressed in this paper. The main contributions are the disadvantages that can appear by using multiple-problem analysis, in the case when the data of some algorithms includes outliers.

Keywords: Comparative study, Non-parametric tests, Parametric tests, Statistical methods, Stochastic optimization algorithms.

1. Introduction

Over the last years, many machine learning and stochastic optimization algorithms have been developed. For each new algorithm, according

to its performance, we need to decide whether it is better than the compared algorithms used on the same problem.

One of the most common ways to compare algorithms used on the same problem is to use statistical tests as comparison techniques of their performance [4, 5, 6, 9, 10]. The common thing of the comparative studies, independently of the research area (machine learning, stochastic optimization or some other research areas), is that they are based on the idea of hypothesis testing [18].

The hypothesis testing, also called significance testing, is a method of statistical inference that could be used for testing a hypothesis about parameter in a population, using data measured in a data sample, or about the relationship between two or more populations, using data measured in data samples. The method starts by defining two hypotheses, the *null hypothesis* H_0 and the *alternative hypothesis* H_A. The null hypothesis is a statement that there is no difference or no effect and the alternative hypothesis is a statement that directly contradicts the null hypothesis by indicating the presence of a difference or an effect. This step in the hypothesis testing is very important, because mis-stating the hypotheses will disrupt the rest of the process. The second step is to select an appropriate *test statistic* T, which is a mathematical formula that allows researchers to determine the likelihood of obtaining the outcomes if the null hypothesis is true. Then, the level of significance α, also called *significance level*, which is the probability threshold below which the null hypothesis will be rejected, needs to be selected. The last step of the hypothesis testing is to make a decision either to reject the null hypothesis in favor of the alternative or not to reject it. The last step can be done with two different approaches. In the standard approach, the possible values of the test statistic for which the null hypothesis is rejected, also called the *critical region*, are calculated using the distribution of the test statistic and the probability of the critical region that is the level of significance α. Then the observed value of the test statistic T_obs is calculated according to the observations from the data sample. If the observed value of the test statistic is in the critical region, the null hypothesis is rejected, and if not, it fails to reject the null hypothesis. In the alternative approach, instead of defining the critical region, a *p-value* that is the probability of obtaining the sample outcome, given the null hypothesis is true, is calculated. The null hypothesis is rejected, if the p-value is less than the selected significance level (the most common values for it are 0.05 and 0.01), and if not, it fails to reject the null hypothesis.

In this paper we follow the recommendations given in some papers [4, 5, 6, 9, 10] in order to perform correct statistical comparison of

the behavior of some of the stochastic optimization algorithms over optimization problems presented of the Black-Box Benchmarking 2015 (BBOB 2015) competition helded at the 5th GECCO Workshop on Real-Parameter Optimization organized at the Genetic and Evolutionary Computation Conference (GECCO 2015) [1].

This paper can be seen as a tutorial and a case study on the use of statistical tests for comparison of stochastic optimization algorithms. In Section 2 we give a review and important comments with regard to the standard statistical tests, the parametric tests, and the non-parametric tests. Section 3 presents the empirical study carried out on the results from the workshop in different scenarios, using pairwise comparison for single-problem analysis, pairwise comparison for multiple-problem analysis, multiple comparisons for single-problem analysis, and multiple comparisons for multiple-problem analysis. In Section 4 we conclude the paper by discussing the disadvantages of the standard statistical tests that are used for statistical comparisons of the behavior of stochastic optimization algorithms.

2. Parametric Versus Non-parametric Statistical Tests

In order to distinguish what to use for your data, between the parametric and the non-parametric test, the first step is to check the assumptions of the parametric tests, also called required conditions for the safe use of parametric tests. So the first step is to use the methods for checking the validity of these required conditions. If the data does not satisfy the required conditions for the safe use of parametric tests, then the tests could lead to incorrect conclusions, and it is better to use the analogous non-parametric test. In general, a non-parametric test is less restrictive than a parametric one, but it is less powerful than a parametric one, when the required conditions for the safe use of the parametric test are true [10].

2.1 Required Conditions for the Safe Use of Parametric Tests

The assumptions or the required conditions for the safe use of parametric tests are the independence, the normality, and the homoscedasticity of the variances of the data.

Two events, A and B are independent, if the fact that A occurs does not affect the probability of B occurring. When we compare the behavior of the stochastic optimization algorithms, they are usually independent.

The assumption of normality is just a hypothesis that a random variable of interest, or in our case the data from the data sample, is distributed according to the normal or Gaussian distribution with mean μ and standard deviation σ. In order to check the validity of this condition, the recommended statistical tests are *Kolmogorov-Smirnov* [19], *Shapiro-Wilk* [23], and *D'Agostino-Pearson* [3]. The validity of this condition can be also checked by using graphical representation of the data using histograms and qunatile-quantile plots (Q-Q plots) [7].

The homoscedasticity indicates the hypothesis of equality of variances, and the *Levene's test* is used to check the validity of this condition [13]. Using this test we can see whether or not a given number of samples have equal variances or not.

2.2 An Overview of Some Standard Statistical Tests

In Table 1 we give an overview of the most commonly used statistical tests that can be used for statistical comparison between two or multiple algorithms. We do not go into details for each of them, because they are standard statistical tests [18]. Which of them is chosen depends on the type of analysis we want to perform, either single-problem or multiple-problem analysis.

Table 1: An overview of parametric and non-parametric tests

	Two Algorithms	*Multiple Algorithms*
Parametric tests	Paired T-Test	Repeated-Measures ANOVA
Non-parametric tests	Wilcoxon Signed-Rank Test, The Sign Test	Friedman Test, Iman-Davenport Test

The single-problem analysis is the scenario when the data comes from multiple runs of the stochastic optimization algorithms on one problem, one function. This scenario is common in stochastic optimization algorithms, since they are of stochastic nature, meaning we do not have any guaranty that the result will be optimal for every run. Moreover, typically even the path leading to the final solution is often different. So to test the quality of the algorithm, it is not enough to performed just one run, but many of them from which we can draw some conclusions.

The second scenario or the multiple-problem analysis is the scenario when several stochastic optimization algorithms are compared on multiple problems, multiple functions. In this case, in most papers the authors use the averaged results for each function to compose a sample of results for each algorithm.

3. Case Study: Black-Box Optimization Benchmarking 2015

In order to go through the recommendations of how to perform statistical comparisons of stochastic optimization algorithms and to see the possible problems that appear, the results from the Black-Box Benchmarking 2015 [1] are used. The Black-Box Benchmarking 2015 is a competition that provides single-objective functions for benchmarking. In addition, it enables analyses of the performance of the competing algorithms, and makes it understandable what are the advantages and disadvantages for each algorithm.

From the competition the algorithms *BSif*, *BSifeg*, *BSrr*, and *Srr* are used for statistical comparisons. The capital letters, S or BS, denote STEP or Brent-STEP method, respectively. The lowercase letters denote the dimension selection strategy: "rr" for round-robin, "if" for the EWMA estimate of the improvement frequency, and "ifeg" for "if" combined with ϵ-greedy strategy [21]. For each of them the results for 24 different noiseless test functions in 5 dimensionality (2, 3, 5, 10, and 20) are selected. At the end, the statistical comparison is performed by comparing the algorithms on 22 different noiseless functions because some of them do not provide data for two functions of the benchmark when the dimension is 20.

The test functions are from 5 groups: separable functions, functions with low or moderate conditioning, function with hight conditioning and unimodal, multi-modal functions with adequate global structure, and multi-modal functions with week global structure. More details about them can be found in [15].

We have done the statistical comparisons in *"R programming language"*, by using the *"lawstat" package* [11] for the *Levene's Test*, the *"stats" package* [22] for the *Kolmogorov-Smirnov Test*, the *Paired-T Test* [16], the *Shapiro-Wilk Test*, and the *Wilcoxon Signed-Rank Test* [17], and the *"scmap" package* [2] for the *Iman-Davenport Test* [6], the *Friedman Test* [6], and the *Friedman Algined-Rank Test* [6].

3.1 Pairwise and Multiple Comparisons for Single-Problem Analysis

In this section pairwise comparison for single-problem analysis is presented together with comments for multiple comparisons for single-problem analysis. The *BSif* and *BSifeg* algorithms are the two algorithms used for pairwise comparison. The pairwise comparisons between these two algorithms for single-problem analysis are performed on 22 benchmark functions when the dimension is 10.

At the beginning of each statistical comparison, the required conditions for the safe use of the parametric tests are checked.

In case of the single-problem analysis the multiple runs of the algorithm on the same function are independent.

To check for normality, the *Shapiro-Wilk Test* and graphical representations by representing the data using histograms and quantile-quantile plots are used. The p-values from the *Shapiro-Wilk Test* are presented in the Table 2, and when the p-value is smaller than the significance level (we used 0.05), then the null hypothesis is rejected, and we assume the data is not normally distributed.

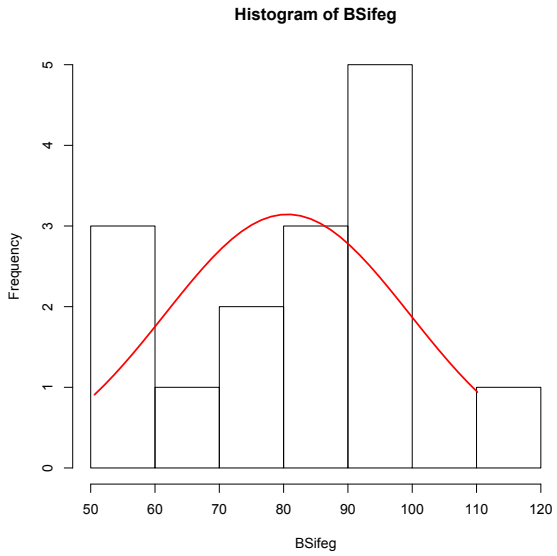
Table 2: Test of normality using *Shapiro-Wilk Test*

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
p-value BSif	-	(.61)	*(.00)	(.70)	*(.01)	*(.02)	*(.01)	(.05)
p-value BSifeg	-	*(.03)	*(.00)	(.47)	*(.01)	*(.00)	(.28)	*(.00)
	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
p-value BSif	*(.00)	*(.00)	(.49)	(.23)	*(.01)	*(.00)	*(.02)	(.05)
p-value BSifeg	*(.00)	*(.01)	(.28)	*(.00)	*(.00)	*(.00)	*(.02)	(.24)
	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}		
p-value BSif	(.21)	*(.04)	(.16)	*(.01)	*(.00)	*(.00)		
p-value BSifeg	(.24)	(.13)	(.07)	(.10)	*(.00)	*(.00)		

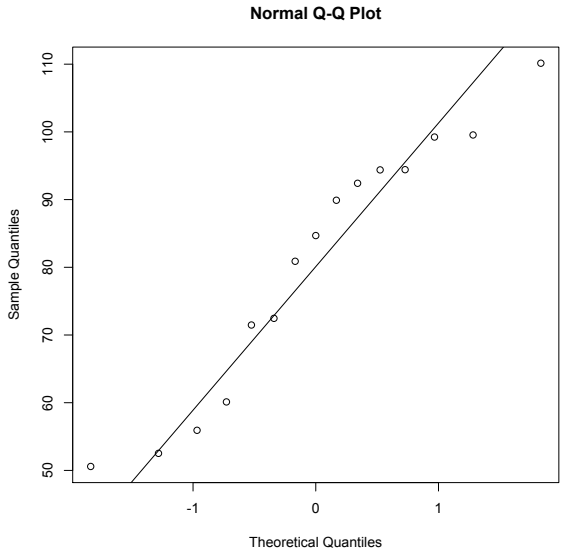
* indicates that the normality condition is not satisfied.

From the same table we can see that there are 6 cases in which the data from both algorithms comes from normal distribution (f_1, f_4, f_{11}, f_{16}, f_{17}, f_{19}), 6 cases in which the data only from one of the algorithms comes from normal distribution (f_2, f_7, f_8, f_{12}, f_{18}, f_{20}), and 10 cases in which the data from both algorithms is not normally distributed (f_3, f_5, f_6, f_9, f_{10}, f_{13},f_{14}, f_{15}, f_{21}, f_{22}).

In Fig. 1 and Fig. 2, the graphical representation of the data with normal and non-normal distribution is presented, respectively. Using the histograms, we can see if the distribution of the data is close to the shape of the distribution we are interested. The red line corresponds to the normal curve with the mean value and the standard deviation obtained from the data. The Q-Q plot is a probability plot, which is a graphical method for comparing two probability distributions by plotting their quantiles against each other. In our case the distribution of the data is compared with the normal distribution. If the data is normally

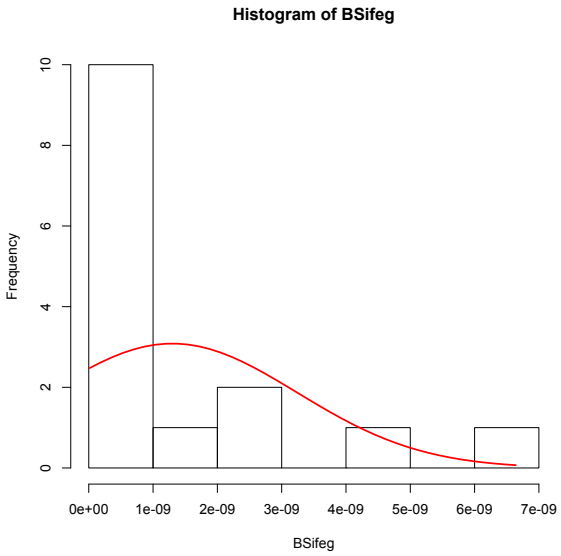


a) Histogram of *BSifeg*.

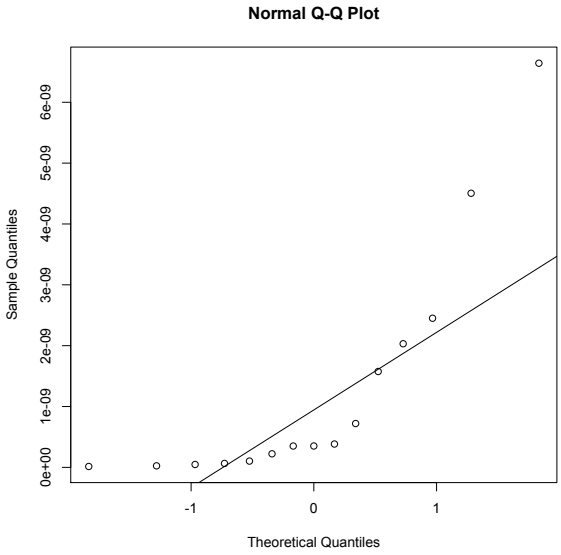


b) QQ-plot of *BSifeg*.

Figure 1: Example of normal distribution for the *BSifeg* algorithm for f_{11} with dimension 10.



a) Histogram of *BSifeg*.



b) QQ-plot of *BSifeg*.

Figure 2: Example of non-normal distribution for the *BSifeg* algorithm for f_3 with dimension 10.

distributed, the data points in the Q-Q normal plot can be approximated with a straight diagonal line.

The next step of the analysis is to check the homoscedasticity. In Table 3 the p-values from the *Levene's Test* for checking homoscedasticity, based on means, are presented. When the p-value obtained by this test is smaller than the significance level (we used 0.05), then the null hypothesis is rejected, and this indicates the existence of a violation of the hypothesis of equality of variances.

Table 3: Test of homoscedasticity using the *Levene's Test*

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
p-value	-	(.08)	(.98)	(.99)	1	(.05)	(.57)	(.07)	*(.01)
	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}
p-value	(.97)	(.37)	*(.00)	*(.00)	*(.04)	(.26)	(.27)	(.85)	(.29)
	f_{19}	f_{20}	f_{21}	f_{22}					
p-value	(.77)	(.94)	(.41)	(.66)					

* indicates that the homoscedasticity condition is not satisfied.

After checking the required conditions for the safe use of the parametric tests, the pairwise comparison of these two algorithms on each function separately is performed using the *Paired-T Test* as parametric test and the *Wilcoxon Signed-Rank Test* as non-parametric test. The p-values obtained by these two tests for the pairwise comparison of the two algorithms are presented in Table 4, where the p-value smaller than the significance level of 0.05, indicates that there is a significant statistical difference between the performance of the two algorithms.

From the Table 4 we can see that for functions f_9 and f_{22} we obtained different results according to the *Paired-T Test* and the *Wilcoxon Signed-Rank Test*. In order to select the true result, first we need to check the results for the validity of the required conditions for the safe use of parametric tests. If we look at Table 2, we can see that for these two functions the normality condition is not satisfied, so we can not use the parametric tests because they could lead to incorrect conclusions, and we need to consider the result obtained by the *Wilcoxon Signed-Rank Test*. Using this test in the case of both functions the null hypothesis is rejected using a significance level of 0.05, so there is a significant statistical difference between the performance of the two algorithms, *BSif* and *BSifeg*, over functions, f_9 and f_{22}.

Table 4: Statistical comparison of *BSif* and *BSifeg* algorithms using *Paired-T Test* and *Wilcoxon Signed-Rank Test*

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
p-value $_{\text{Paired}-\text{T}}$	-	(.19)	(.32)	(.76)	-	*(.02)	(.81)	* (.02)
p-value $_{\text{Wilcoxon}}$	-	(.08)	(.72)	(.72)	-	*(.03)	(.60)	*(.00)
	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
p-value $_{\text{Paired}-\text{T}}$	(.08)	(.40)	(.18)	(.11)	*(.00)	*(.03)	(.06)	(.64)
p-value $_{\text{Wilcoxon}}$	*(.00)	(.17)	(.26)	(.42)	*(.00)	*(.00)	(.05)	(.39)
	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}		
p-value $_{\text{Paired}-\text{T}}$	(.34)	(.95)	(.76)	(.87)	(.14)	(.14)		
p-value $_{\text{Wilcoxon}}$	(.68)	(.71)	(.98)	(.93)	(.14)	*(.01)		

* indicates that the null hypothesis is rejected, using $\alpha = 0.05$.
p-value $_{\text{Paired}-\text{T}}$, and p-value $_{\text{Wilcoxon}}$ indicate the p-values obtained by *Paired-T Test* and *Wilcoxon Signed-Rank Test*, respectively.

If we want to perform multiple comparisons for single-problem analysis, we need to go through the same steps as in the pairwise comparison, but we need to use the *repeated-masures ANOVA* as parametric test [12], and the *Friedman Test* or *Iman-Davenport Test* as non-parametric tests. If there is significance statistical difference between the algorithms we can continue with some post-hoc procedures relevant to the test we used [6].

3.2 Pairwise and Multiple Comparisons for Multiple-Problem Analysis

In this section multiple comparisons for multiple-problem analysis are presented togehter with comments for pairwise comparison. Following the recommendations from some papers [6, 10] that addressed the same topic, the averaged results for each function with dimension 10 are used to compose a sample of results for each algorithm. The *BSifeg*, *BSrr*, and *Srr* are the algorithms used for comparison over multiple functions.

First, the conditions for the safe use of the parametric test are checked. The condition for independence is satisfied, as we explained above.

The p-values for normality condition obtained by using the *Shapiro-Wilk Test* are presented in Table 5, from where we can see that neither of the algorithms assumes that the data comes from normal distribution.

Table 5: Test of normality using *Shapiro-Wilk Test*

	BSifeg	*BSrr*	*Srr*
p-value	*(.00)	*(.00)	*(.00)

* indicates that the normality condition is not satisfied.

The homoscedasticity is checked by applying the *Levene's Test*. The p-value obtained from the *Levene's Test* is 0.63, from which it follows that the homoscedasticity is satisfied.

Because the normality condition is not satisfied, we cannot use the *repeated-measures ANOVA* as parametric test, and we can continue the analysis by using the *Friedman Test*, the *Iman-Davenport Test*, and the *Friedman Aligned-Rank Test* as non-parametric tests. The differences between these three tests and the recommendations when to use them are explained in [6], and the p-values we obtained are presented in Table 6.

Table 6: Multiple comparisons for multiple-problem analysis

	Friedman Test	*Iman-Davenport Test*	*Friedman Aligned-rank Test*
p-value	*(.03)	*(.02)	*(.04)

* indicates that the null hypothesis is rejected, using $\alpha = 0.5$.

Using the p-values reported in Table 6, according to the three tests that are used, the null hypothesis is rejected, and there is a significant statistical difference between these three algorithms.

In order to see the difference that appears between the performance of the three algorithms, the distributions of the data for each algorithm are presented in Fig. 3. In the figure we can see that there is no difference between the distributions of the data of the three algorithms that are used in multiple-problem analysis. To confirm this, we introduced the *Kolmogorov-Smirnov Test* to compare the distributions between the pairs of algorithms, and the p-values are presented in Table 7, from where we can see that the p-values obtained are greater than 0.05, so we can not reject the null hypothesis, therefor the distributions of the data between the pairs of the algorithms are the same.

The question that arises here is, if this difference between the algorithms obtained by the use of the non-parametric tests is inforced by averaging the results from multiple runs for each function to compose a sample of results for each algorithm. Averages are known to be sensitive to outliers. For example, in machine learning, different techniques that can be used to remove outliers are presented [20, 14]. It happened for

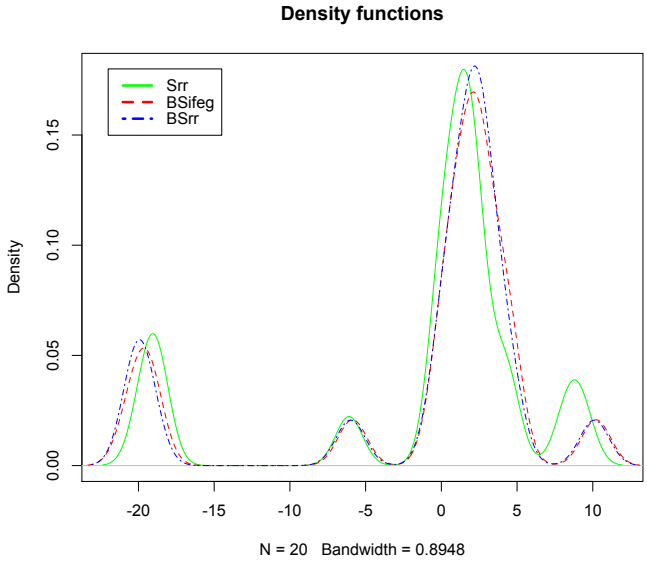


Figure 3: Probability distributions of the data of *BSifeg*, *BSrr*, and *Srr* used in multiple-problem analysis.

Table 7: Two-sample *Kolmogorov-Smirnov Test*

	(BSifeg,BSrr)	*(BSifeg,Srr)*	*(BSrr,Srr)*
p-value	1	(.86)	(.99)

example that in 15 runs the average result of one function for a given algorithm was better than another algorithm, but in new 15 runs the average result of the same function and the same algorithm could be worse than the other algorithm, and this happened because in the new 15 runs we have some outliers, or some poor runs. One solution could be to perform several multiple runs of an algorithm on the same problem, and then to average the averages results obtained by the runs. But in stochastic optimization we are not interested to have so many runs, because this is time-consuming. Another solution, also our further work, is to try to find what we can use as a measure for comparison of stochastic optimization algorithms that are robust on outliers, instead of using averaging of the results.

The pairwise comparison for multiple-problem analysis could be done using the same steps, but using the *Paired-T Tets* as parametric test,

and the *Wicoxon Signed-Rank Test* or the *The Sign Test* [8] as non-parametric tests.

4. Conclusion

In this paper a tutorial and a case study of statistical comparison between the behaviour of stochastic optimization algorithms are presented.

The main conclusion of the paper are the disadvantages that can appear in the multiple-problem analysis following the recommendations of other tutorials that address this topic. These disadvantages can happen by averaging the results from multiple runs for each function to compose a sample of results for each algorithm, in the case when the data includes outliers. In general, the outliers can be skipped using some techniques, but they need to be used with great care. But for multiple-problem analysis skipping outliers is really a question because only the results for certain problems would be changed and not for other problems. All this leads to a need of some new measures that will be robust to outliers and can be used to compose a sample for each algorithm over multiple problems, and after that to continue the analysis by using some standard statistical tests.

References

[1] Black-box benchmarking 2015. `http://coco.gforge.inria.fr/doku.php?id=bbob-2015`, accessed: 2016-02-01.

[2] B. Calvo and G. Santafe. scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, 2015.

[3] R. B. D'agostino, A. Belanger, and R. B. D'Agostino Jr. A suggestion for using powerful and informative tests of normality. *The American Statistician*, 44(4):316–321, 1990.

[4] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[5] J. Derrac, S. García, S. Hui, P. N. Suganthan, and F. Herrera. Analyzing convergence performance of evolutionary algorithms: a statistical approach. *Information Sciences*, 289:41–58, 2014.

[6] J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[7] J. Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage Learning, 2015.

[8] W. J. Dixon and A. M. Mood. The statistical sign test. *Journal of the American Statistical Association*, 41(236):557–566, 1946.

[9] S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational

intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010.

[10] S. García, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: a case study on the CEC2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009.

[11] J. L. Gastwirth, Y. R. Gel, W. L. Wallace Hui, V. Lyubchich, W. Miao, and K. Noguchi. *lawstat: Tools for Biostatistics, Public Policy, and Law*, 2015, r package version 3.0. [Online]. Available: `https://CRAN.R-project.org/package=lawstat`.

[12] E. R. Girden. *ANOVA: Repeated measures*. Sage, 1992.

[13] G. V. Glass. Testing homogeneity of variances. *American Educational Research Journal*, 3(3):187–190, 1966.

[14] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Elsevier, 2011.

[15] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2010: Experimental setup. Research Report RR-7215, INRIA, 2010.

[16] H. Hsu and P. A. Lachenbruch. Paired t test. In *Wiley Encyclopedia of Clinical Trials*, 2008.

[17] F. Lam and M. Longnecker. A modified wilcoxon rank sum test for paired data. *Biometrika*, 70(2):510–513, 1983.

[18] E. L. Lehmann, J. P. Romano, and G. Casella. *Testing statistical hypotheses*. Wiley, New York, 1986.

[19] H. W. Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402, 1967.

[20] M. Nikolova. A variational approach to remove outliers and impulse noise. *Journal of Mathematical Imaging and Vision*, 20(1-2):99–120, 2004.

[21] P. Pošík and P. Baudiš. Dimension selection in axis-parallel brent-step method for black-box optimization of separable continuous functions. *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, pages 1151–1158, 2015.

[22] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. [Online]. Available: `https://www.R-project.org/`.

[23] S. S. Shapiro and R. Francia. An approximate analysis of variance test for normality. *Journal of the American Statistical Association*, 67(337):215–216, 1972.

THE IMPACT OF QUALITY INDICATORS ON THE RATING OF MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS

Miha Ravber, Marjan Mernik, Matej Črepinšek
Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia
miha.ravber@um.si, marjan.mernik@um.si, matej.crepinsek@um.si

Abstract Comparing the results of single objective optimizers is an easy task in comparison to multi-objective optimizers for which the result is usually an approximation of the Pareto optimal front. These approximation sets must first be evaluated. One of the most popular methods for evaluation is the use of quality indicators, for which the result is a real valued number that reflects a certain aspect of quality. Evaluating and comparing multi-objective optimizers is an important issue. It has been empirically proven that chess ranking can be successfully applied to ranking and comparing single objective evolutionary algorithms. In this paper, the method was adapted to multi-objective evolutionary algorithms (MOEAs). The comparison of several different quality indicators in the chess rating system was conducted in order to get a better insight on their characteristics and how they affect the ranking of MOEAs. Although it is expected that quality indicators with the same optimization goals would yield a similar ranking of MOEAs, it has been shown that results can be contradictory.

Keywords: Chess rating, Evolutionary algorithms, Multi-objective optimization, Performance assessment, Quality indicator.

1. Introduction

The goal of multi-objective optimization (MOO) is to obtain the Pareto optimal front that contains the best trade-off solutions. Since many multi-objective optimization problems (MOP) are difficult to solve, the outcome of the optimization is usually an approximation of the Pareto front. In order to compare these approximations, they need to be evaluated. Evaluating the quality of these approximations is itself an MOP. Zitzler et al. [20] suggested three optimization goals that need to be measured: the distance of the resulting nondominated set to

the Pareto optimal front should be minimized; a good (in most cases uniform) distribution of the solutions found is desirable; the extent of the obtained nondominated front should be maximized. Comparing the performance of MOEAs remains an open problem. The most popular measures are quality indicators (QI); the term "performance metric" is also used to quantify the differences between approximation sets.

Many different QIs for measuring the quality of approximation sets have been proposed in the literature [1, 8, 9, 10, 12, 14, 15, 19, 20, 23, 24]. Each QI has been designed with a standpoint that takes one or more previously mentioned optimization goals into consideration. This means that no single indicator alone can reliably measure MOEA performance. It should be noted that several surveys and experiments have been conducted to analyze individual indicators [7, 8, 16, 24]. The results have shown inconsistencies and contradictions in the assessment of various approximation sets. It was argued in [7] and [16], that without established comparison criteria, claims based on heuristically chosen QIs do little to determine a given MOEAs actual efficiency and effectiveness. In addition, the conclusions are useless for answering the question of which algorithms are superior to others. Can it be argued that one algorithm is better than another even though the outcome depends on the selected QI?

The aim of this paper is to obtain better insight into the impact of the selected QI for the comparison of MOEAs. The focus is on the analysis of different QIs with the help of a chess rating system.

The remainder of the paper is organized as follows. In Section 2, some basic concepts of quality indicators are introduced. The chess rating system with Glicko-2 is presented in Section 3. In Section 4, the execution of the experiment and results are presented. Finally, the paper concludes in Section 5.

2. Quality Indicators

Approximation sets can be compared using dominance relations. However, there are numerous limitations to using this approach. For example, the extent to which one algorithm is better than another cannot be expressed nor can it be expressed in which aspects this is so. Furthermore, when using dominance relations, there are cases in which approximation sets are incomparable. In order to overcome these limitations, QIs have been designed. These indicators quantitatively measure approximations of Pareto optimal fronts. Therefore, QIs are in essence functions that assign each approximation set a real number that reflects different aspects of quality or quality differences. Zitzler et al.

[24] defined a quality indicator I as an m-ary function $I : \Omega^m \to \mathbb{R}$ that assigns each vector $(A_1, A_2, \ldots, A_m)$ of m approximation sets a real value $I(A_1, \ldots, A_m)$. Once the approximation sets are evaluated by indicators, different conclusions can be drawn about their relations. For different aspects of quality, different indicators need to be used.

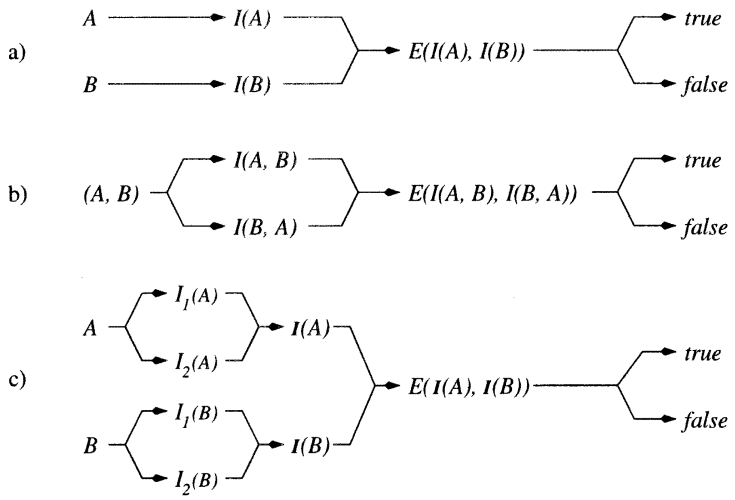


Figure 1: The concept of comparison methods adapted from [24].

Quality indicators have been categorized into different groups from different points of view to better understand their nature [24, 7, 10]. They are mainly categorized by the aspects of quality that they assess. These aspects include the closeness to the Pareto-optimal front, the number of elements of the Pareto-optimal front found, and the maximum spread of solutions. Quality indicators are also classified based on the number of approximation sets they take as an argument. Unary indicators accept one approximation and binary accept two. However, in principle indicators that accept an arbitrary number of arguments are also possible. When evaluating with unary indicators the resulting real values need to be compared in order to see which result set is better. Binary indicators, in contrast, compare two result sets to determine which one is better. Therefore, when comparing t sets using binary indicators $t(t1)$, comparisons need to be carried out to obtain the final ranking. Some unary indicators require a reference set to perform the evaluation, which must be taken into consideration since real-world problems have unknown Pareto-optimal fronts. When the reference set is available, any indicator can be converted from binary to unary. There are also other

categories that are not used as often, such as computational complexity, the sensitivity to scaling, the number of objectives, etc. It is also desirable that an indicator be compatible and complete with respect to dominance relations.
Quality indicators need interpretation, and different comparison methods can be used. This is best illustrated by Zitzler (Fig. 1) [24] where concepts of comparison methods using either only unary or only binary indicators are presented. Case (a) uses a single unary QI, (b) a single binary QI, and (c) a combination of two unary QIs. In cases (a) and (b), the indicator I evaluates the approximation sets A and B. The result is passed to the interpretation function E that decides the outcome. In case (c), two indicators are applied to A and B then the resulting two indicator values are combined in a vector, $I(A)$ for A and vector $I(B)$ for B. The vectors are passed to the interpretation function E that decides the outcome.

Table 1: Quality indicators and their properties.

Quality Indicator	Convergence	Uniformity	Spread	Requires reference set
CS [23]	√			
$I_{\epsilon+}$ [24]	√			√
GD [15]	√			√
HV [23]	√	√	√	
IGD [1], IGD+ [9]	√	√	√	√
MPFE [14]	√			√
MS [20]			√	√
R2 [8]	√			√
S [12]		√		
Generalized Spread Δ [19]			√	√

In this paper, eleven QIs are used, based on prevalence and different properties. Selected indicators are listed with their quality aspects in Table 1. When comparing algorithms, usually a handful of QIs are selected and then the experiment is performed and evaluated with selected statistical methodologies. In our case, the Chess Rating System for Evolutionary Algorithms (CRS4EAs) [13] is used. The outcome of the game was determined by methods a and b (Fig. 1), depending whether the indicator is unary or binary.

3. Chess Rating System for Evolutionary Algorithms (CRS4EAs)

In this paper, we use CRS4EAs based on the Glicko-2 system, in which each player receives his rating R, rating deviation RD and rating volatility σ [6]. The volatility measure indicates the degree of expected fluctuation in a players rating. When a player has an unpredictable performance such as exceptionally strong results after a period of stability, the volatility measure is high. If a player performs at a consistent level, the volatility measure is low. The rating deviation indicates how reliable a players rating is. A small rating deviation means a player plays often and has a reliable rating. In contrast, if the rating deviation is high, his rating is unreliable. A players strength can be summarized in the form of a 95% confidence interval. It can be said that we are 95% confident that the players rating R is within an interval $[R - 2RD, R + 2RD]$. To apply a rating, multiple games between multiple players within a rating period (tournament) need to be performed. Before the tournament starts the ratings, rating deviations and rating volatilities for all players need to be set. If a player is new or not established, his performance rating has to be defined first. The experiment was performed with the Evolutionary Algorithm Rating System (EARS) [5] framework that supports CRS4EAs. The codes for the different algorithms, problems, and calculation of quality indicators are available in the jMetal framework [4] and MOEA framework [11]. Each MOEA represents a chess player and searches for the best Pareto front approximation for a given problem represents a chess game. In a game, two MOEAs play against each other where the outcome is decided when each approximation set is evaluated with the given QI. Each player plays multiple games against all participants in the tournament.

4. Experiment

In this section, the experiment execution and results are presented. Chess ranking leaderboards of five algorithms with eleven QIs were compared.

4.1 Experimental Settings

In the experiment, five MOEAs were chosen for the tournament: IBEA [22], MOEA/D [17], NSGA-II [3], PESA-II [2] and SPEA2 [21]. The benchmark contains well-known unconstrained problems from the CEC 2009 special session and competition on the performance assessment of multi-objective optimization algorithms [18]. Population size

for all five MOEAs was set to be 100 for all of the 2-objective problems and 300 for the 3-objective problems, according to [16]. The rest of parameters setting of the algorithms are set according to the source code of [4, 11]. The maximum number of evaluations for a problem was set to 300,000. The number of independent runs of the tournament was set to 30. For the chess rating, Glickman recommended setting rating R to 1500, rating deviation RD to 350, and rating volatility σ to 0.06 [6]. A tournament was conducted for each QI. It should be noted that approximation sets were normalized prior to evaluation since different objective functions can have a different magnitude.

4.2 Experimental Execution

Figure 2 displays the flowchart of a single execution of the experiment in EARS. The experiment is conducted in the form of tournaments. Each tournament consists of $k = 5$ algorithms $\{a_1, a_2, \ldots, a_5\}$, $N = 10$ optimization problems and is performed in $n = 30$ independent runs. Each algorithm returns the best solution set for each optimization problem over n independent runs ($k * N * n$ results). These results are then evaluated with the QI that was given for the current tournament. After evaluation, the resulting two real values are passed to the interpretation function. The comparison methods use a single unary QI or a single binary QI (Fig. 1 a and b). A set $\{a_i, a_j\}_{l,m}$ is a single comparison or a single game between two algorithms a_i and a_j for the optimization problem F_l over run m where $i, j \in \{1, \ldots, k\}, i \neq j, l \in \{1, \ldots, N\}$ and $m \in \{1, \ldots, n\}$. The solution sets y_i and y_j from algorithms a_i and a_j for the problem F_l on run m are evaluated with the given quality indicator I and passed to the interpretation function E that defines the outcome of the comparison. Therefore, one tournament consists of $(k * (k-1)/2) * N * n$ games. At the end of the tournament, the results are gathered in the forms of wins, losses, and draws. Afterward, the ratings, rating deviations, and rating volatilities are updated. All the data is collected and presented on a leaderboard. The tournament was repeated for each QI, resulting in eleven leaderboards.

4.3 Results and discussion

The results for all QIs are presented in Table 2. All algorithms have played through the whole tournament for each indicator. For each indicator, there are two rows. The first row contains the final rating and rank for a given algorithm. The second row contains the 95% confidence intervals. For all players in all tournaments, the rating deviations reached their minimum value (50) [13]. The low value of RD was achieved with

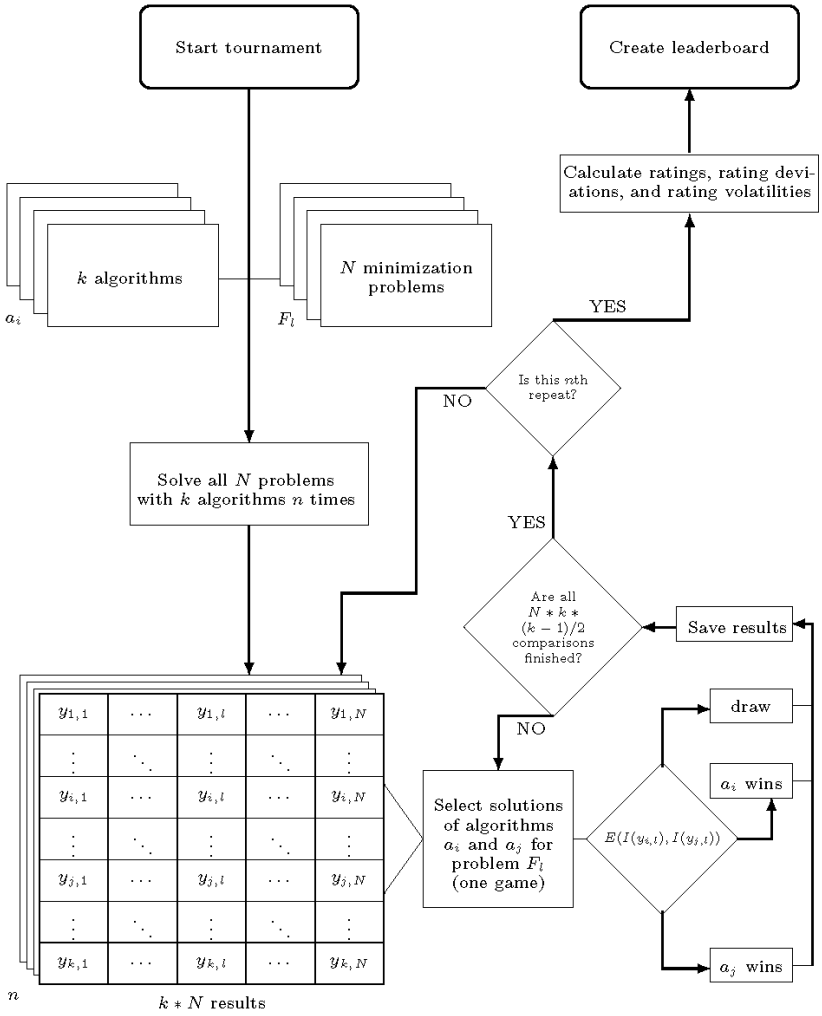


Figure 2: Flowchart of experiment execution in EARS [5].

an adequate number of tournaments, and it indicates that players competed frequently and have a stable rating. As expected, indicators are not unified in the ranking of algorithms, which is reflected in the deviation from the average rank displayed in the last row. Regardless of the incoherence in the ranking, some indicators assess the approximations similarly. Based on similarities in ranking, indicators can be divided into three groups. The biggest group contains seven indicators: HV, IGD, $IGD+$, $I_{\epsilon+}$, $R2$, MS and Δ. The first three indicators have the

Table 2: Leaderboards of five algorithms with eleven QIs on unconstrained CEC 2009 benchmark problems. For each indicator the rating intervals RI with 95% confidence ($RD \pm 2RD$) are presented.

	IBEA	MOEA/D	NSGA-II	PESAII	SPEA2
HV	*1380 (5)* [1280,1480]	**1649 (1)** [1549,1749]	1538 (2) [1438,1638]	1461 (4) [1361,1561]	1471 (3) [1371,1571]
IGD	*1228 (5)* [1128,1328]	**1700 (1)** [1600,1800]	1581 (2) [1481,1681]	1453 (4) [1353,1553]	1538 (3) [1438,1638]
$IGD+$	*1414 (5)* [1314,1514]	**1605 (1)** [1505,1705]	1566 (2) [1466,1666]	1437 (4) [1337,1537]	1478 (3) [1378,1578]
$I_{\epsilon+}$	*1390 (5)* [1290,1490]	**1599 (1)** [1499,1699]	1522 (3) [1422,1622]	1440 (4) [1340,1540]	1548 (2) [1448,1648]
$R2$	*1287 (5)* [1187,1387]	1598 (2) [1498,1698]	**1647 (1)** [1547,1747]	1400 (4) [1300,1500]	1567 (3) [1467,1667]
MS	*1218 (5)* [1118,1318]	1624 (2) [1524,1724]	**1770 (1)** [1670,1870]	1339 (4) [1239,1439]	1549 (3) [1449,1649]
Δ	*1266 (5)* [1166,1366]	1570 (3) [1470,1670]	1628 (2) [1528,1728]	1370 (4) [1270,1470]	**1667 (1)** [1567,1767]
CS	**1818 (1)** [1718,1918]	1399 (4) [1299,1499]	*1287 (5)* [1187,1387]	1525 (2) [1425,1625]	1471 (3) [1371,1571]
GD	**1848 (1)** [1748,1948]	1292 (4) [1192,1392]	*1291 (5)* [1191,1391]	1622 (2) [1522,1722]	1447 (3) [1347,1547]
$MPFE$	**1954 (1)** [1854,2054]	*1170 (5)* [1070,1270]	1339 (4) [1239,1439]	1644 (2) [1544,1744]	1392 (3) [1292,1492]
S	**1831 (1)** [1731,1931]	*1158 (5)* [1058,1258]	1421 (4) [1321,1521]	1633 (2) [1533,1733]	1457 (3) [1357,1557]
$\bar{x}$	3.9	2.9	3.1	3.6	3

same ranking. The remaining indicators ($I_{\epsilon+}$, $R2$, MS and Δ) have ranked differently, but there is no significant difference between the algorithms that switched ranks. The other two groups ranked the MOEAs very differently than the bigger group. If only the ranking is considered, two pairs of indicators are obtained, which differ only in the rank of $MOEA/D$ and $NSGA-II$. Since there is no significant difference between the fourth and fifth ranking algorithm with the $MPFE$ indicator, we grouped it with CS and GD. Although S achieved the same ranking as $MPFE$, it is in a separate group because there is a significant difference between $MOEA/D$ and $NSGA-II$. The bigger group contains all three compliant indicators: one strictly Pareto-compliant indicator (HV) and two weakly Pareto-compliant indicators ($IGD+$ and the unary $I_{\epsilon+}$). Since compliant indicators are deemed to be more reliable,

we conclude that the ranking of the bigger group is also more reliable. It is also interesting to note that indicators within the same group do not evaluate the same aspects of quality. This can be interpreted as indicating that the resulting approximation sets do not dominate only in one optimization goal. In Table 2, the rating was used to show the absolute power of the algorithm over other algorithms, however, the rating interval should also be considered. If the confidence intervals do not overlap, the algorithms have provided significantly different results, whereas the converse is not necessarily true.

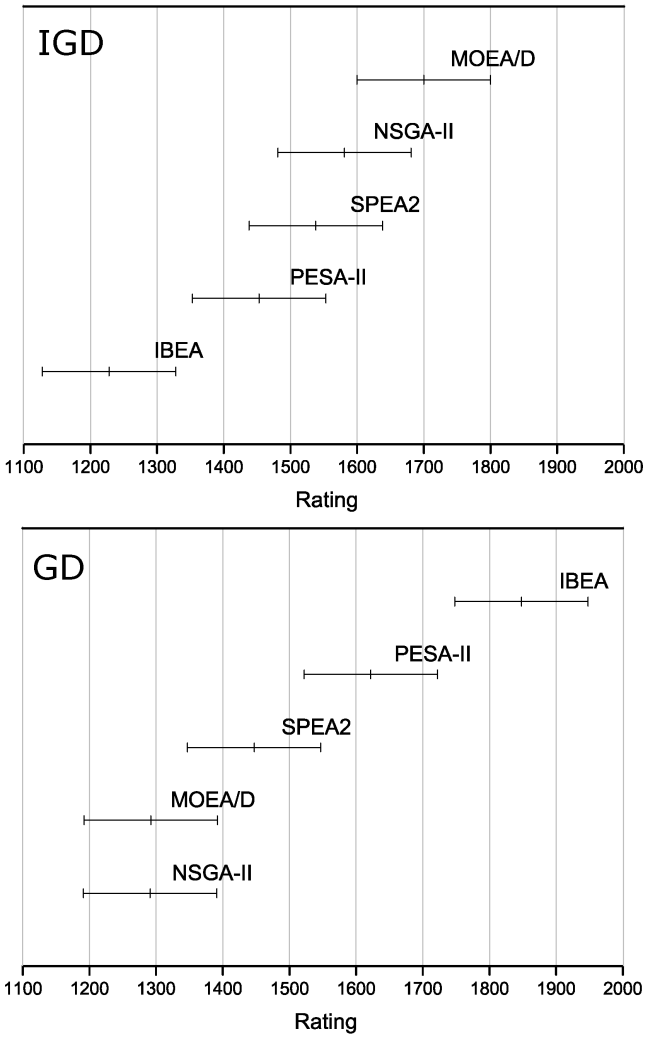


Figure 3: 95% confidence intervals for IGD (top) and GD (bottom) QI in table 2.

Due to space constraints we plotted the confidence intervals for IGD and GD (e.g., Fig. 3), which are some of the more popular QIs in literature. The results can be interpreted by observing ratings and rating interval. As we can see with the IGD indicator, $MOEA/D$ performed the best, being significantly better than $PESAII$ and $IBEA$. On the last place is $IBEA$, which performed the worst, being significantly outperformed by all other algorithms. In contrast, with GD $IBEA$ performed the best by significantly outperforming $MOEA/D$, $NSGAII$, $SPEA2$, and $PESAII$. On the second place is $PESAII$, outperforming $MOEA/D$ and $NSGAII$, which shares the last place with one point of difference. It is important to observe that the selected QI ranked the MOEAs almost in reverse order. This result can be explained by the property of GD indicator that measures only the convergence of the approximation set regardless of its spread and uniformity. Furthermore, the experiment was limited by selected set of problems, MOEAs, and QIs.

5. Conclusion

In this paper, eleven QIs were compared with CRS4EAs on five different MOEAs, solving unconstrained MOP from the CEC 2009 benchmark. For the given experiment, it has been shown that individual QIs differently rank algorithms even if they evaluate the same aspects of quality. Therefore, picking coherent indicators is very important. Selected QIs were categorized into three groups that have insignificant differences in MOEAs ranking. The biggest group with the state-of-the-art indicator contains Δ, $I_{\epsilon+}$, HV, IGD, $IGD+$, $R2$ and MS indicator. The other two groups containing CS, GD, $MPFE$ and S indicator returned very different ranking orders and are not recommended. Because of the disparity in rankings between indicators, a desired ranking of algorithms can be achieved with a carefully assembled set of indicators [16]. Therefore, in order to claim that one algorithm is better, a balanced and fair set of indicators is recommended. For future work, we would like to integrate this approach into CRS4EAs and test it on additional diverse problems for more detailed analysis of QIs.

References

[1] P. A. N. Bosman and D. Thierens. The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):174–188, 2003.

[2] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates. PESAII: Region-based selection in evolutionary multiobjective optimization. *Proceedings of the*

Genetic and Evolutionary Computation Conference (GECCO), pages 124-130, 2001.

[3] K. Deb, A. Pratab, S. Agrawal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197, 2002.

[4] J. J. Durillo, and A. J. Nebro. jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10): 760–771, 2011.

[5] Evolutionary Algorithms Rating System (Github). `https://github.com/matejxxx/EARS`, 2016.

[6] M.E. Glickman. Example of the Glicko-2 System. Boston University, 2012.

[7] D. Guoqiang, Z. Huang, and M. Tang. Research in the Performance Assessment of Multi-objective Optimization Evolutionary Algorithms. *Proceedings of the International Conference on Communications, Circuits and Systems (ICCCAS)*, pages 915–918, 2007.

[8] M. P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the nondominated set. Technical Report IMM-REP-1998-7, 1998.

[9] H. Ishibuchi, H. Masuda, Y. Tanigaki, and Y. Nojima. Difficulties in specifying reference points to calculate the inverted generational distance for many-objective optimization problems. *Proceedings of the IEEE Symposium on Computational Intelligence in Multi-Criteria Decision Making*, pages 170–177, 2014.

[10] M. Li, S. Yang, and X. Liu. Diversity comparison of Pareto front approximations in many-objective optimization. *IEEE Transactions on Cybernetics*, 44(12): 2568–2584, 2014.

[11] MOEA Framework - A Free and Open Source Java Framework for Multiobjective Optimization. `http://www.moeaframework.org`, 2016.

[12] J. R. Schott. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master Thesis, MA: Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 1995.

[13] N. Veček, M. Mernik, and M. Črepinšek. A chess rating system for evolutionary algorithms: A new method for the comparison and ranking of evolutionary algorithms. *Information Sciences*, 277(1): 656–679, 2014.

[14] D. A. Van Veldhuizen. Multiobjective Evolutionary Algorithms: Classifications, Analysis, and New Innovations. Ph.D. dissertation, Faculty of the Graduate School of Engineering , Air Force Institute of Technology, 1997.

[15] D. A. Van Veldhuizen and G. B. Lamont. Evolutionary computation and convergence to a Pareto front. *Proceedings of the Genetic Programming Conference*, pages 221-228, 1998.

[16] G. G. Yen and Z. He. Performance metric ensemble for multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 18(1):131–144, 2014.

[17] Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[18] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari. Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition. Technical Report CES-487, 2009.

[19] A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, and E. Tsang. Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 892-899, 2006.

[20] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[21] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: improving the strength Pareto evolutionary algorithm. Technical Report TIK-Report 103, 2001.

[22] E. Zitzler and K. Simon. Indicator-based selection in multiobjective search. *Proceedings of International Conference on Parallel Problem Solving form Nature (PPSN)*, pages 832–842, 2004.

[23] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257-271, 1999.

[24] E. Zitzler and L. Thiele. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 117–132, 2003.

BUILDING ENSEMBLES OF SURROGATES BY OPTIMAL CONVEX COMBINATION

Martina Friese, Thomas Bartz-Beielstein
SPOTSeven Lab, TH Köln, Gummersbach, Germany
martina.friese@th-koeln.de, thomas.bartz-beielstein@th-koeln.de

Michael Emmerich
LIACS, Leiden University, The Netherlands
m.t.m.emmerich@liacs.leidenuniv.nl

Abstract When using machine learning techniques for learning a function approximation from given data it can be difficult to select the right modelling technique. Without preliminary knowledge about the function it might be beneficial if the algorithm could learn all models by itself and select the model that suits best to the problem, an approach known as automated model selection. We propose a generalization of this approach that also allows to combine the predictions of several surrogate models into one more accurate ensemble surrogate model. This approach is studied in a fundamental way, by first evaluating minimalistic ensembles of only two surrogate models in detail and then proceeding to ensembles with more surrogate models. The results show to what extent combinations of models can perform better than single surrogate models and provide insights into the scalability and robustness of the approach. The focus is on multi-modal functions which are important in surrogate-assisted global optimization.

Keywords: Ensemble Methods, Function Approximation, Global Optimization, Model Selection, Surrogate Models.

1. Introduction

Surrogate models are mathematical functions that, basing on a sample of known objective function values, approximate the objective function, while being cheaper in terms of evaluation. Such surrogate models can then be used to partially replace expensive objective function evaluations. Expert systems like SPOT [1] come with a large variety of models that has to be chosen from when initiating an optimization process.

The choice of the right model implies the quality of the the optimization process.

Often expert knowledge is needed to decide which model to select for a given problem. If there is no preliminary knowledge about the objective function it might be beneficial if the algorithm could learn all by itself which model suits best to the problem. This can be done by evaluating different models on test data a priori and using a statistical model selection approach to select the most promising model.

Some occurrences imply that there might also be a benefit in linearly combining predictors from several models into a more accurate predictor. In Fig. 1 such an occurrence is happening. Predictions with two different (Kriging) models are shown and results obtained by a convex combination of the predictors of these models. Different errors seem to be compensated by the combined model's predictions.

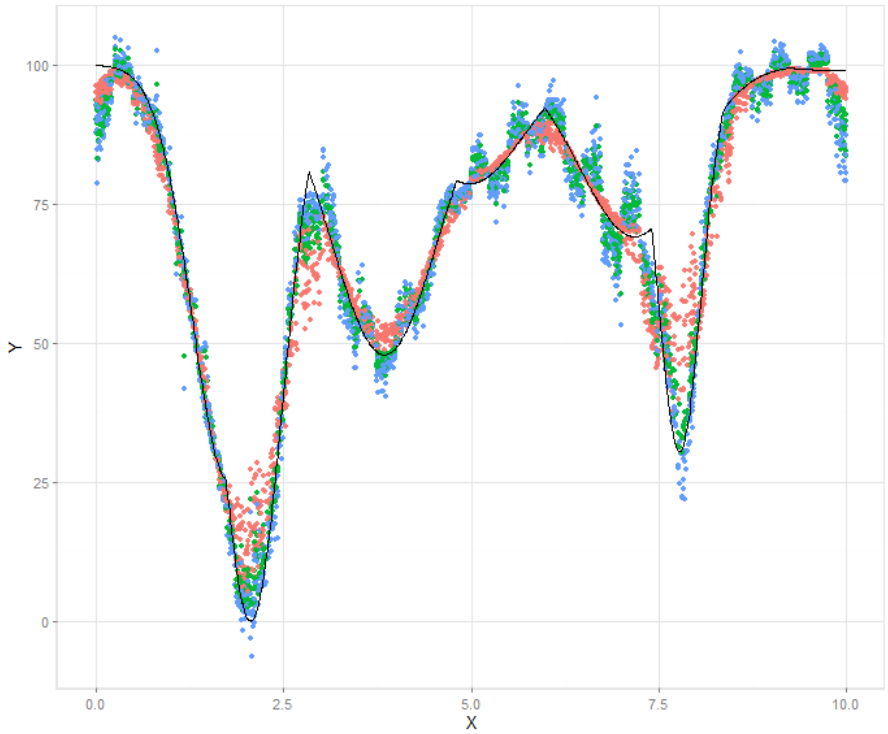


Figure 1: The black line marks the actual objective function value. The dots show the results obtained in a leave-one-out cross-validation. Blue and red dots mark the predictions of single models. The green dots shows predictions obtained with an optimal convex linear combination of the two predictors.

Such occurrences show that a predictor based on a single modeling approach is not always the best choice. On the other hand, complicated expressions based on multiple predictors might not be a good choice, either, due to overfitting and lack of transparency. Using convex combinations of predictors from available models seems to be a 'smart' compromise. Given s surrogate models $\hat{y}_i : \mathbb{R}^d \to \mathbb{R}, i = 1, \ldots, s$ and d the dimension of the search space, by a *convex combination of models* we understand a model given by $\sum_{i=1}^{s} \alpha_i \hat{y}_i$ with $\sum \alpha_i = 1$ and $\alpha_i \geq 0, i = 1, \ldots, s$. Finding an optimal convex combination of models can be viewed as a generalization of model selection, where selecting only one model is a special case. Convex combinations of predictors have also the advantage that they combine only predictions and can be used for heterogeneous model ensembles. The main research questions are:

(Q-1) Can convex combinations of predictors improve as compared to (single) model selection?

(Q-2) Given the answer is positive, what are explanations of the observed behavior?

(Q-3) How can a system be build that finds the optimal convex combination of predictions on training data?

In order to answer these questions, detailed empirical studies are conducted, starting from simple examples and advancing to more complex ones. This paper follows a structure, where the discussion of experimental results follows directly the introduction of the modeling extensions.

2. General Approach and Related Work

To base a decision or build a prediction from multiple opinions is common practice in our everyday live. It happens in a democratic government, or when in TV shows the audience is asked for help. One also might use it when we try to build an opinion on a topic that is new to us. Naturally, such tools already found their way into statistical prediction and machine learning. In statistics and machine learning an *ensemble* is a prediction model from several models, aiming for better accuracy. A comprehensive introduction to ensemble-based approaches in decision making is given in [5] and [9]. Generally, there are two groups of ensemble approaches: the first group's approaches, the so-called *single-evaluation* approaches, only choose and build one single model, whereas the second group's approaches, the so-called *multi-evaluation* approaches, build all models, and use the derived information to decide which output to use. For each of these two approaches, several model selection strategies can be implemented. Well-known strategies are:

- *Round robin* and *randomized choosing* are the most simplistic implementations of ensemble-based strategies. In the former approach, the models are chosen in a circular order independent of their previously achieved gain. In the latter approach, the model to be used in each step is selected randomly from the list of available models. The previous success of the model is not a decision factor.

- *Greedy strategies* choose the model that provided the best function value so far, while the SoftMax strategy uses a probability vector, where each element represents the probability for a corresponding model to be chosen [13]. The probability vector is updated depending on the reward received for the chosen models.

- *Ranking strategies* try to combine the responses of all meta models to one response, where all meta models contributed to, rather than to choose one response.

- *Bagging* combines results from randomly generated training sets and can also be used in function approximation, whereas

- *Boosting* combines several weak learners to a strong one in a stochastic setting.

- *Weighted averaging* approaches do not choose a specific model's result but rather combine it by averaging. Since bad models should not deteriorate the overall result, a weighting scheme is introduced. Every model's result for a single design point is weighted by its overall error, the sum over all models yields the final value assigned to the design point. A similar approach is *stacking*, where the weights are chosen by an additional training step.

The convex model combinations in this paper can be viewed as an elegant stacking approach and as such is similar to 'ensembles of surrogates' [7], which however used a fixed rule for determining weights. In our work weights are optimized globally and the approach is analysed in a controlled and detailed way. Since most of the black-box real-world problems considered to be difficult are multimodal, the focus for this work also is on multimodal function approximation (cf. [8, 10, 12, 14]).

3. Preliminaries

By a *surrogate model*, we understand here a function $\hat{y} : \mathbb{R}^d \to \mathbb{R}$ that is an approximation to the objective function $y : \mathbb{R}^d \to \mathbb{R}$, learned from a finite set of evaluations of the objective function. Kriging surrogate

models are used in our study. A set of three different kernels is used to implement the ensemble strategies. Following the definitions from [11], the correlation models can be described as follows. We consider stationary correlations of the form $\mathcal{R}(\theta, w, x) = \prod_{j=1}^{n} \mathcal{R}(\theta_j, w_j - x_j)$. The first model uses the *exponential* kernel $\mathcal{R}(\theta, w, x) = \exp(-\theta_j |w_j - x_j|)$ the second model uses an *Gaussian* kernel $\mathcal{R}(\theta, w, x) = \exp(-\theta_j |w_j - x_j|^2)$, whereas the third model is based on the *spline correlation* function $\mathcal{R}(\theta, w, x) = \zeta(\theta_j |w_j - x_j|)$ with

$$\zeta(\epsilon_j) = \begin{cases} 1 - 15\epsilon_j^2 + 30\epsilon_j^3 & \text{for} \quad 0 \le \epsilon_j \le 0.2 \\ 1.25(1 - \epsilon_j)^3 & \text{for} \quad 0.2 < \epsilon_j < 1 \\ 0 & \text{for} \quad \epsilon_j \ge 1. \end{cases}$$

Here, ϵ and θ are hyperparameters estimated by likelihood maximization.

For generating *test functions* we use the *Max-Set of Gaussian Landscape Generator* (MSG). It computes the upper envelope of m weighted Gaussian process realizations and can be used to generate continuous, bound-constrained optimization problems [6].

$$G(x) = \max_{i \in 1,2,\ldots,m} (w_i g_i(x)),$$

where $g : \mathbb{R}^n \to \mathbb{R}$ denotes an n-dimensional Gaussian function

$$g(x) = \left(\frac{\exp\left(-\frac{1}{2}(x - \mu)\Sigma^{-1}(x - \mu)^T\right)}{(2\pi)^{n/2}|\Sigma|^{1/2}} \right)^{1/n},$$

μ is an n-dimensional vector of means, and Σ is an $(n \times n)$ covariance matrix. Implementation details are presented in [2]. For the generation of the objective function the `spotGlgCreate` method of the SPOT package has been used. The options used for our experiments are shown in Table 1. With the parameter d the dimension of the objective function is specified. The lower and upper bounds (l and u, respectively) specify the region where the peaks are generated. The value max specifies the function value of the global optimum, while the maximum function value of all other peaks is limited by t, the ratio between the global and the local optima.

4. Binary Ensembles

This Section analyses models which combine only two models. Convex combinations of models will be referred to as ensemble models, while the original models will be referred to as base models. We focus on positive weights, since we do not want to select models that make predictions which are anti-correlated with the results.

Table 1: Gaussian landscape generator options

Parameter	*Description*	*Value*
d	Dimension	$2 - 10$
m	Number of peaks	$10 - 40$
l	Lower bounds of the region, where peaks are generated	$\{0_1, \ldots, 0_d\}$
u	Upper bounds of the region, where peaks are generated	$\{5_1, \ldots, 5_d\}$
max	Max function value	100
t	Ratio between global and local optima	0.8

A sample of points (design) is evaluated on the objective function (MSG, for parameters see Table 1). For the sampling of the points a latin hypercube design featuring 40 design points is generated. The two base models are Kriging with exponential correlation function (referred to as a) and Gaussian correlation function (referred to as b). Both base models are fitted to the data and then asked to do a prediction on the testdata. The predictions $\hat{y}$ of the ensemble models are calculated as convex combinations of the predictions of the base models.

Given a weight α_i, where $\alpha_i \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0\}$, the ensemble models can be defined as the linear combinations of the models a and b as follows:

$$\hat{y}_n = \alpha_n \times \hat{y}_a + (1 - \alpha_n) \times \hat{y}_b \tag{1}$$

The models are evaluated by calculating the root mean squared error (RMSE) of the predictions made during a leave-one-out cross-validation on the 40 design points.

Since randomness has been induced into the experiment by using the latin hypercube design, the evaluation process has been repeated 50 times. With each model returning one prediction for each design point in every repetition this results in a total of 2000 prediction values (40 design points $\times$ 50 repetitions) for each model.

To get a first quick insight into the result data, for each repetition the rankings of the RMSE's have been calculated. The models with $\alpha = 0.6$, $\alpha = 0.8$ and $\alpha = 0.9$ dominate this comparison, each performing best 8 out of 50 times. The base models, a and b, performed best only in four respectively two cases out of 50. Never an ensemble model with positive weights was performing worst.

In order to achieve some comparability between the RMSE's of different repetitions all RMSE's have been repetition-wise scaled to values between zero and one, so that the scaled RMSE of the best model in one repetition is always zero and the scaled RMSE of the worst model

for one repetition is always 1.0. Figure 2 shows the boxplot over these

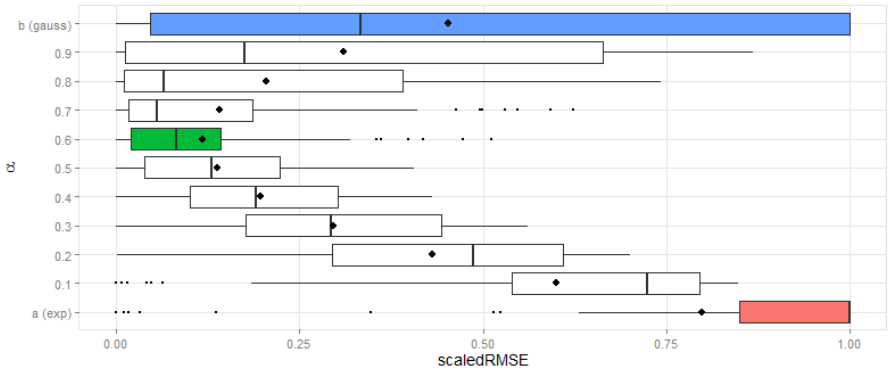


Figure 2: Boxplot over the scaled RMSE's of all models. The models are defined by an α-weighted linear combination of the two base models. The results of the base models depicted on the outer rows and colored red (exponential kernel), respectively blue (Gaussian kernel). The model combination chosen as best with $\alpha = 0.6$ is colored green. The mean value of each result bar is marked by a dot.

scaled RMSE's. It can be seen that the model a (exponential) in most of the cases performs worst since its median is 1.0 - only some outliers come closer to zero.

Model b (Gaussian) shows a larger variation in its performance. It has been the best- as well as the worst performing model each at least once. Its median and mean performances are average in comparison with all models evaluated. A parabolic tendency can be seen in the performance.

Due to the convex combination of the predictor, a prediction by the ensemble model cannot be worse but it might be better than both base models. An ensemble can only be better, if one model overestimates and the other model underestimates the objective function value. In the experiment this happens in 649 out of 2000 cases.

As a *consistent method for evaluating the performance and automatically choosing the best model* the following approach is proposed: Model-wise mean-, median- and 3rd quartile-values are calculated. The resulting values are ranked and the rankings summed up to one final ranking. The model that achieved the lowest value is recommended as best choice. In Fig. 2 the model recommended as best choice by this method is colored green.

5. Detailed Analysis on Transparent Test Cases

It can clearly be stated that for this first experiment setup the combination of two models is beneficial for the overall prediction. In this section we're going to have a closer look at possible explanations for

the successful result. Are there problem features that encourage using ensembles and is this result generalizable.

We chose a 1D objective function to allow for a better understanding of the underlying process. This is the only change in the experimental setup. The Figs. 1 and 2 from Section 4 depict the main results of this second experiment setup. Figure 2 shows the scaled RMSE's for all models. Applying the rule defined in Section 4 names the model obtained by a linear combination with $\alpha = 0.7$ as best choice.

Figure 1 shows only the performance of the best choice model and the base models. Each dot marks a single prediction made during the leave-one-out cross-validation. As can be seen in the plot, the predictions of the model a (exponential), marked by red dots, seem to smooth the objective function - straight segments are well met while curved segments are smoothed out.

The predictions of the model b (Gaussian), marked by the blue dots show signs of overfitting. Again straight segments are well met but when approaching local extrema the predictions start to oscillate. So the linear combination of both predictions averages positive as well as negative outliers of base models. This seems to provide some benefit to the overall experiment outcome.

Since the curves and corners in the objective function seem to make the game here, two additional experiments are set up. For these experiments two objective functions are specified featuring corners that are not continuous differentiable. For one experiment a triangle objective function is used while the other features a piecewise assembled objective function. Figure 4a shows the results for the piecewise assembled objective function. Looking at these results, we again find a strong parabolic tendency in the boxplot. Both base models have a rather large variance in their performance. The ensemble model marked as best choice has a smaller variance and performed better than the base models in nearly all cases.

The results on the triangle objective function happened to show a clear tendency towards base model b, which clearly outperformed basemodel a and thus was chosen best.

6. Ternary Ensembles

Next, the experiments are extended to a larger scale: The dimensionality of the objective function is increased and three base models are combined. As before Kriging models with different kernels are used, but now a third model using the spline correlation function is added.

$$\alpha_n, \beta_n, \gamma_n \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0\}, \quad \alpha_n + \beta_n + \gamma_n = 1 \qquad (2)$$

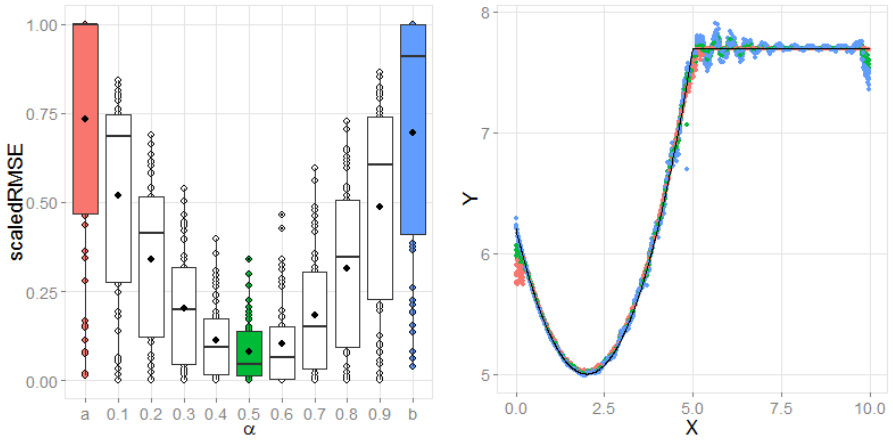


Figure 3: Results on a piecewise assembled objective function. Left hand side plot shows the scaled RMSE's. The α value defines the weight for the linear combination. The ensemble obtained by a linear combination with $\alpha = 0.5$, here colored green, is suggested best for this experiment setup. On the right hand side all predictions done during the leave-one-out cross validation for the base models and the best model are plotted against the objective function.

For the linear combination of three base models three weights are needed, that sum up to one as specified in (2). With a step size of 0.1 for the linear combinations this results in 66 models.

Figure 4a shows the results of the first experiment using three base models. The only change that has been made to the original experiment setup, besides the number of base models, is the dimension d of the objective function and the number of peaks m generated in the Gaussian landscape. As a first step towards objective functions of higher complexity, the dimension of the objective function has been set to 4. But this change alone is not sufficient to gain a larger complexity, since without adjusting the number of Gaussian components used for generating the objective function, it rather gets less complex. Thus the number of Gaussians process trajectories is adjusted to ten times the dimension.

With the points getting smaller when approaching the center of the triangle, it can be stated, that again it is beneficial to use a convex combination of the base models.

7. Scaling-up to multiple models

By now, only experiments with up to three models are carried out, but the underlying goal is to evolve a system that is able to handle quite a large set of available base models. But at this point quickly

another approach is needed, since the number of possible discretised convex combinations between a higher number of base models grows exponentially. A recursive formula is given below: There is only one setting where the first model gets all the weight (first factor in sum). In all other settings the remaining weight must be distributed on the remaining models.

$$f(r,s) = 1 + \sum_{r^*=1}^{r-1} f(r - r^*, s - 1), \quad f(r,1) = 1, f(1,s) = s \qquad (3)$$

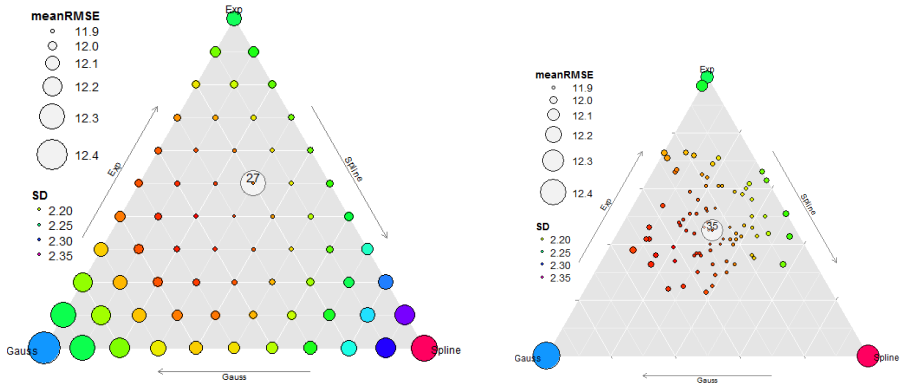
The relation between number of models, the step size for the discretised convex combinations and the resulting number of linear combinations can be expressed as function of r the reciprocal of the step size and s the number of models as defined in (3). Using three base models and a step size of 0.1 as defined in (2) this results in $f(10,3) = 66$ linear combinations. Now thinking of combinations of 10 base models already results in $f(10,10) = 92378$ linear combinations.

The complexity of the search space, when increasing the number of models, quickly gets too large to do a complete evaluation of all possible convex combinations with a fixed step size of 0.1. Looking at previous results, the function that describes the performance of the models built by convex combinations up to this point only showed unimodal characteristics. This seems to be expectable due to the nature of convex combinations. We expect the function to show this characteristic also when combining larger number of models.

Thus, instead of a complete evaluation of all linear combinations, an optimization step is implemented to find the best combination. The allowed weights are restricted to a precision of two decimal places. Since the area around the optimum tends to build a plateau. This reduces the possible search space without loosing the possible best solution.

For the sake of comparability, the experiment setup here is exactly the same as the one used in Section 6. Only the process itself changed. Prior to this experiment, all convex combinations have been evaluated. Now, only the base models are evaluated initially. Other models are only evaluated during the optimization. We also stuck to the method used by the (1+1)-ES of comparing the offspring only to the parent rather than to the whole population as we did it before.

For the mutation of the weights vector $\mathbf{v} = (\alpha, \beta, \gamma)^T$ three random samples of a normal distribution function with standard deviation of 0.16 have been drawn and added to the weight vector. Since this alone does not meet the requirements needed for a valid weight vector, the resulting vector has been adjusted in three steps:



a) The optimal linear combination has been found by a complete evaluations of all linear combinations using a fixed step size of 0.1.

b) QQ-plot of *BSifeg*.

Figure 4: The plots show the results of the experiment set up with three base models. Each circle depicts the performance results for one model. The three base models are located on the corners of the triangle, models gained by linear combinations of only two models are located on the outer border. Circles on the inner area of the area show the results for models that were gained by linear combinations of all three base models. The size of the circles denotes the mean RMSE value, the color the standard deviation. The model proposed as best choice is marked by an additional white circle.

1 If $\min(\alpha, \beta, \gamma) < 0$ then $\mathbf{v} \leftarrow \mathbf{v} - (\min(\alpha, \beta, \gamma), \ldots, \min(\alpha, \beta, \gamma))^T$,

2 $\mathbf{v} \leftarrow \mathbf{v}/(\alpha + \beta + \gamma)$,

3 Round the values α, β, γ to two decimal places so, that $\alpha + \beta + \gamma = 1$.

For this experiment we allowed a maximum of 100 individuals to be evaluated. Within these bounds already the 35th evaluated individual has been the best individual found in this run. Figure 4b depicts the results of this optimization step. As before, the best individual is marked by a white circle. However, since determination of optimal weights in the linear model is a non-linear optimization problem, we cannot guarantee the optimality of the proposed weights. So far, we have achieved similar results in repeated runs and on different objective functions. Due to space constraints, statistical validation is however left to future work.

8. Discussion and Outlook

Reconsidering the research questions from Section 1, it was shown that convex linear combinations of predictors can generate better results than model selection (Q-1). A system, which finds optimal linear combinations, was presented in Section 4. As a possible explanation a compensation of outliers was found, an effect that occured in particular in non-smooth objective functions (Q-2). The corresponding experiments were extended to a larger scale, in terms of dimensionality as well as number of models, in Section 6 with results indicating that the methods are scalable (Q-3). Finally, in Section 7, we proposed a method to include even more base models to the system, showing that evolutionary optimization can be an effective tool for finding optimal convex combinations. With this method the foundation has been created for a larger system including all available models. Although research questions (Q-2) and (Q-3) could be partially answered, larger studies are required to statistically confirm scalability and find in depth explanations.

In summary, convex combination of models are a promising approach in situtations where several types of models are available. if the user does not know, which model to choose, a linear combination might be a promising approach. An interesting aspect about convex combinations is that they are easy to interpret and that weights in the linear model can shed some light on the relevance of certain models and illustrate, which model is active.

Ideas and questions that will be discussed in future work are:

- Experiments featuring more base models, also including other types of models.
- Extensive analysis of the influence of objective function attributes on the experiment outcome. The results of Section 5 suggest, that particularly piecewise assembled objective functions might be of special interest.
- Studies also allowing other operations than simple convex combinations only: Does increasing the model complexity of model combinations yield much better results?
- Comparing to approaches that chose fixed weights [7].

Acknowledgement: This work has been supported by the *Bundesministeriums für Wirtschaft und Energie* under the grants KF3145101WM3 und KF3145103WM4. This work is part of a project that has received funding from the *European Unions Horizon 2020 research and innovation program* under grant agreement No 692286.

References

[1] T. Bartz-Beielstein. Spot: An r package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. Technical Report 05/10, Research Center CIOP (Computational Intelligence, Optimization and-Data Mining), Cologne University of Applied Science, Faculty of Computer Scienceand Engineering Science, 2010. Comments: Related software can be downloaded from `http://cran.r-project.org/web/packages/SPOT/index.html`.

[2] T. Bartz-Beielstein. How to Create Generalizable Results. In J. Kacprzyk and W. Pedrycz (Eds.) *Springer Handbook of Computational Intelligence*, pages 1127–1142. Springer, Berlin Heidelberg, 2015.

[3] H. G. Beyer and H. P. Schwefel. Evolution strategies - A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] M. Friese, M. Zaefferer, T. Bartz-Beielstein, O. Flasch, P. Koch, W. Konen, and B. Naujoks. Ensemble-Based Optimization and Tuning Algorithms. *Proceedings of the 21st Workshop Computational Intelligence*, pages 119–134, 2011.

[6] M. Gallagher and B. Yuan. A general-purpose tunable landscape generator. *IEEE Transaction on Evolutionary Computation*, 10(5):590–603, 2006.

[7] T. Goel, R. T. Haftka, W. Shyy, and N. V. Queipo. Ensemble of surrogates. *Structural and Multidisciplinary Optimization*, 33(3):199–216, 2007.

[8] W. Jakob, M. Gorges-Schleuter, I. Sieber, W. Süß, and H. Eggert. Solving a Highly Multimodal Design Optimization Problem Using the Extended Genetic Algorithm GLEAM. In: S. Hernandez, A. J. Kassab, and C. A. Brebbia (Eds.) *Computer Aided Design of Structures VI*, pages 205–214, WIT Press, Southhampton, 1999.

[9] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[10] L. Qing, W. Gang, Y. Zaiyue, and W. Qiuping. Crowding clustering genetic algorithm for multimodal function optimization. *Applied Soft Computing*, 8(1):88–95, 2008.

[11] J. S. Søren, N. Lophaven, and H. Bruun Nielsen. Dace - a matlab kriging toolbox. Technical report, Technical University of Denmark, 2002.

[12] C. Stoean, M. Preuss, R. Stoean, and D. Dumitrescu. Multimodal optimization by means of a topological species conservation algorithm. *IEEE Transactions on Evolutionary Computation*, 14(6):842–864, 2010.

[13] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[14] K.-C. Wong, K.-S. Leung, and M.-H. Wong. Protein structure prediction on a lattice model via multimodal optimization techniques. *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 155–162, 2010.

THE EXPONENTIAL CROSSOVER IN L-SHADE ALGORITHM

Radka Poláková
Centre of Excellence IT4Innovations, Institute for Research and Applications of Fuzzy Modeling, University of Ostrava, Czech Republic
radka.polakova@osu.cz

Abstract Differential evolution is popular and efficient algorithm for global optimization. L-SHADE algorithm is one of the most successful adaptive versions of the algorithm. It uses only binomial crossover. We study employing the exponential crossover in the algorithm. Our tests are carried out on CEC2015 benchmark set for learning-based optimization competition. According to our results, the employing of the exponential crossover together with binomial one into L-SHADE algorithm is beneficial.

Keywords: Binomial crossover, Differential evolution, Evolutionary algorithm, Experimental comparison, Exponential crossover.

1. Introduction

Differential evolution (DE) is one of the most known and used stochastic algorithms for solving of real-parameter optimization tasks. The algorithm was proposed by Storn and Price in 1997 [10]. There are many researchers who are interested in the algorithm, its behavior, and improvement of its performance [3, 8]. Effectiveness of the algorithm depends on values of its parameters and different values of control parameters are often more beneficial in different stage of search process. These facts are the reasons why many adaptive versions of the algorithm were proposed since differential evolution algorithm appeared, e.g., [1, 2, 7, 14, 18, 21]. The algorithm presented by Tanabe and Fukunaga in 2013 called Success-history based adaptive differential evolution with linear reduction of population size algorithm (L-SHADE) [13] is one of the most effective versions of DE up to these days. L-SHADE employs the most common used type of crossover, binomial one. There are some studies in which types of crossover used in DE are studied

and discussed, e.g., [15, 16, 17, 19]. Mentioned studies and the facts claimed in the papers inspired us to study the impact of including the exponential crossover into L-SHADE algorithm.

In the following two sections, the DE algorithm and L-SHADE algorithm are described. In Section 4, two new versions of L-SHADE algorithm are introduced. Carried out experiments are described and results of them are specified and discussed in Section 5. Conclusions are given in the last section.

2. Differential Evolution

Differential evolution algorithm [10] is one of the most known evolutionary algorithms. DE solves global optimization tasks with continuous search space. Let us have a real function $f : S \to R$, $S \subset R^D$, f is objective function and DE's aim is to find global minimum point of f in S, i.e. such point $\mathbf{x}^*$ that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ holds for all $\mathbf{x} \in S$, S is search space, D is problem dimension. DE works with population P of NP-points from search space S. The points are chosen randomly from S with uniform distribution at the beginning of the search process. NP is the size of population P. Population P then evolves generation by generation. A new generation is created in the following way. A new point $\mathbf{y} \in S$, so called trial point, is created for each member $\mathbf{x}_i$ of population P. If $f(\mathbf{y}) \leq f(\mathbf{x}_i)$ holds, $\mathbf{y}$ replaces $\mathbf{x}_i$ in population P. Otherwise $\mathbf{x}_i$ stays to be a member of population P for next generation. A trial point $\mathbf{y}$ is built up by evolutionary operators mutation and crossover from some points of current generation of population P. A combination of a mutation and a crossover is called DE-strategy. An abbreviation DE/a/b/c is commonly used for a DE-strategy, a is mutation, b is the number of difference vectors used in the mutation, and c is employed crossover. The search process is interrupted, when a stopping condition holds, for example when maximal count of objective function evaluations is reached.

A mutant $\mathbf{u}$ is created by operation mutation for a point $\mathbf{x}_i$. There are many types of mutation used in DE. A mutation is abbreviated by a/b, where a is type of mutation and b is a count of difference vectors used in the mutation. The most common mutation is rand/1 (1). The mutant $\mathbf{u}$ is developed from three random points $\mathbf{x}_{r1}$, $\mathbf{x}_{r2}$, $\mathbf{x}_{r3}$ of current generation of P ($r_1 \neq r_2 \neq r_3 \neq i$). Scaling factor $F \in (0, 2]$ is input parameter usually set as $F \in (0, 1]$. Current-to-pbest/1 mutation (2) was proposed by Zhang and Sanderson [21]. The point $\mathbf{x}_{pbest}$ is randomly chosen point from $p \times 100\%$ best points of population P, p is input

parameter, $0 < p < 1$.

$$\mathbf{u} = \mathbf{x}_{r1} + F \times (\mathbf{x}_{r2} - \mathbf{x}_{r3}), \tag{1}$$

$$\mathbf{u} = \mathbf{x}_i + F \times (\mathbf{x}_{pbest} - \mathbf{x}_i) + F \times (\mathbf{x}_{r1} - \mathbf{x}_{r2}). \tag{2}$$

The trial point $\mathbf{y}$ is created from $\mathbf{x}_i$ and mutant $\mathbf{u}$ by a crossover.

Binomial crossover combines coordinates of $\mathbf{x}_i$ with coordinates of mutant $\mathbf{u}$ into trial point $\mathbf{y}$ according to formula (3).

$$y_j = \begin{cases} u_j & \text{if} \quad U_j \leq CR \quad \text{or} \quad j = l \\ x_{ij} & \text{if} \quad U_j > CR \quad \text{and} \quad j \neq l\,, \end{cases} \tag{3}$$

where l is a number randomly chosen from set $\{1, 2, \ldots, D\}$, $U_1, U_2, \ldots, U_D$ are independent random variables uniformly distributed in $[0,\ 1)$. Input parameter $CR \in [0, 1]$ is crossover parameter.

In exponential crossover, L $(1 \leq L \leq D)$ consecutive coordinates are moved from mutant $\mathbf{u}$ into trial point $\mathbf{y}$. A m is randomly chosen with uniform distribution from set $\{1, 2, \ldots, D\}$. Probability of moving of k-coordinate in sequence $\{u_{m+k}\}$, $0 \leq k \leq (L-1)$ into trial point is CR^k. Coordinates, which are not copied into trial point $\mathbf{y}$ from $\mathbf{u}$, are copied from $\mathbf{x}_i$.

Binomial crossover is more often employed type of crossover than exponential one in adaptive versions of DE [1, 7, 9, 12]. The CR parameter influences the number of elements to be put into $\mathbf{y}$ from $\mathbf{x}_i$ and from $\mathbf{u}$ for both mentioned types of crossover. Let p_m is a mutation probability for a coordinate of $\mathbf{x}_i$ to be replaced by respective coordinate of mutant $\mathbf{u}$. Zaharie found [20] that the relation between probability p_m and CR is linear for binomial crossover and strongly non-linear (described by equality (4)) for exponential crossover.

$$CR^D - \ D\ p_m\ CR + \ D\ p_m - 1 = 0. \tag{4}$$

Figure 1 illustrates the relations between CR and p_m for binomial and exponential crossover and differences amongst them for dimension $D = 30$. The DE algorithm is described by pseudo-code in Algorithm 1.

3. L-SHADE Algorithm

The success-history based adaptive differential evolution algorithm with linear reduction of population size (L-SHADE) proposed by Tanabe and Fukunaga in 2014 [13] is based on SHADE algorithm [11, 12].

SHADE algorithm [11, 12] uses DE/current-to-pbest/1/bin DE-strategy, archive A, and an adaptation of control parameters F and CR. It is

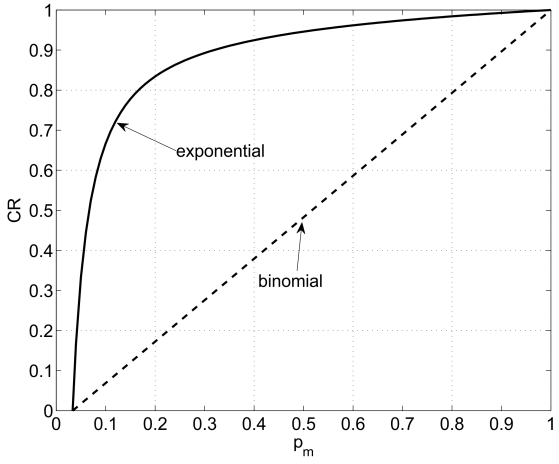


Figure 1: The dependence of CR on probability of mutation p_m for binomial and exponential crossover, $D = 30$

Algorithm 1 Differential evolution algorithm

generate an initial population $P = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{NP})$, distributed uniformly in search space
evaluate $f(\mathbf{x}_i)$, $i = 1, 2, \ldots, NP$
while stopping condition not reached **do**
  $Q := \emptyset$
  for $i = 1$ to NP **do**
    generate a mutant point $\mathbf{u}$ by mutation
    create a trial point $\mathbf{y}$ from mutant $\mathbf{u}$ and $\mathbf{x}_i$ by crossover
    evaluate $f(\mathbf{y})$
    if $f(\mathbf{y}) \leq f(\mathbf{x}_i)$ **then**
      insert $\mathbf{y}$ into new generation Q
    else
      insert $\mathbf{x}_i$ into new generation Q
    end if
  end for
  $P := Q$
end while

based on JADE [21] algorithm. Archive A is initialized as empty set and each point $\mathbf{x}_i$, which is replaced by its better trial point $\mathbf{y}$, is included into archive A during the search process. The archive A is adjusted after each generation to have maximal size of NP, where members for

removing from archive are chosen randomly. $\mathbf{x}_{r1}$ is randomly selected point from P and $\mathbf{x}_{r2}$ is randomly selected point from $P \cup A$ for current-to-pbest/1 mutation (2). The p for the mutation is chosen new for each trial point randomly from interval $p \in (1/NP, 0.2]$ in SHADE algorithm.

F and CR are generated before each new trial point is created from Cauchy and normal distribution, respectively. So called historical circle memories M_F and M_{CR} are implemented for storing several values of the first parameters of F and CR distributions, respectively. Recommended value of the memories' size is $H = NP$ for the SHADE algorithm. Each member of these both memories is set to value 0.5 at the beginning of the search process. The memories are updated in the way described below.

When F and CR are needed for a point $\mathbf{x}_i$ for creation of $\mathbf{y}$, an uniform random number r is chosen from the set $\{1, 2, \ldots, H\}$ and F is random number with Cauchy distribution with parameters $(M_{F_r}, 0.1)$ and CR is random number from normal distribution $N(M_{CR_r}, 0.1)$. If generated F is higher than 1 then F is set to $F = 1$ and if $F \leq 0$ then new value of F is generated. Generated value of CR is trimmed into interval $[0, 1]$, i.e. if $CR > 1$ then $CR = 1$ and if $CR < 0$ then $CR = 0$. If $f(\mathbf{y}) < f(\mathbf{x}_i)$ holds for trial point $\mathbf{y}$ made by the pair of parameters ($\mathbf{y}$ is successful), the F and CR are stored into sets S_F and S_{CR}, respectively. The sets S_F and S_{CR} are set as empty sets at the beginning of a generation.

New values of M_{F_k} and M_{CR_k} (k is current position in M_F and M_{CR}) are computed at the end of the generation from values stored in S_F and S_{CR}, respectively. They are created as weighted means and they are weighted by differences between values of objective function. The M_{F_k} and M_{CR_k} are computed only if there is at least one successful trial point $\mathbf{y}$ in the generation. Then $S_F = \{F_1, F_2, \ldots, F_{|S_F|}\}$ and $S_{CR} = \{CR_1, CR_2, \ldots, CR_{|S_F|}\}$, note that $|S_F| = |S_{CR}|$ holds. The computation of new values M_{F_k} and M_{CR_k} is done according to equalities (5)-(7). $mean_{WL}$ is weighted Lehmer mean and $mean_{WA}$ is weighted arithmetic mean (6). $\mathbf{y}_m$ in (7) is successful trial point generated by F_m and CR_m and $\mathbf{x}_m$ is point of population, which was replaced by trial point $\mathbf{y}_m$.

$$M_{F_k} = mean_{WL}(S_F), \qquad M_{CR_k} = mean_{WA}(S_{CR}), \quad when \;\; S_F \neq \emptyset, \tag{5}$$

$$mean_{WL}(S_F) = \frac{\sum_{m=1}^{|S_F|} w_m \; F_m^2}{\sum_{n=1}^{|S_F|} w_n \; F_n}, \;\; mean_{WA}(S_{CR}) = \sum_{m=1}^{|S_F|} w_m \; CR_m, \tag{6}$$

$$w_m = \frac{\Delta f_m}{\sum_{h=1}^{|S_F|} \Delta f_h}, \quad \Delta f_m = |f(\mathbf{x}_m) - f(\mathbf{y}_m)|, \tag{7}$$

At the beginning of the search process, parameter k is set to $k = 1$. k is increased by 1 after each computation of M_{F_k} and M_{CR_k}. If $k > H$ for such increased k, k is set to $k = 1$.

L-SHADE algorithm [13] is very similar to SHADE algorithm. The only differences are the use of linear reduction of the population size NP and the different setting of some input parameters. The parameter p for current-to-pbest/1 mutation is set on a constant value, $p = 0.11$. The population size is linearly decreasing generation by generation with the increasing number of the objective function evaluations (FES) during the search process from NP^{init} at the beginning to NP^{min} at the end of the search process, i.e. if allowed number of the function evaluations ($MaxFES$) is reached. Relatively big value of NP^{init} is very useful, in order to ensure the most possible exploration of search space. On the other hand, small value of NP^{min} is recommended in order to increase the length of computation (count of created generations) as possible and to let the algorithm to specify the solution as possible. The size of population for generation G+1 is computed according to the formula:

$$NP_{G+1} = round\left[NP^{init} - \frac{FES}{MaxFES}\left(NP^{init} - NP^{min}\right)\right], \qquad (8)$$

where FES is the current number of the objective function evaluations. Whenever $NP_{G+1} < NP_G$, the $(NP_G - NP_{G+1})$ worst individuals are deleted from the population.

Values of the size of archive A, NP^{init}, NP^{min}, and the size of historical circle memories H recommended by authors of [13] are $2.6 \times NP$, $NP^{init} = 18 \times D$, $NP^{min} = 4$, and $H = 6$, respectively. L-SHADE with mentioned parameter setting was the best DE-version in optimization competition on CEC2014 [4, 5]. The L-SHADE algorithm proposed in [13] is described by pseudo-code in Algorithm 2.

4. Exponential Crossover in L-SHADE

In L-SHADE algorithm, each trial point is created from original point $\mathbf{x}_i$ and mutant $\mathbf{u}$ by binomial crossover, the more often used type of crossover in DE. There are studies which focus on the comparison of using of binomial and exponential crossovers in DE in the literature. The influence of employing the exponential crossover in competitive differential evolution adaptive version of DE was studied in [15, 16]. The author found that applying both types of crossover brings improvement for standard functions in comparison with applying only the binomial crossover. For composition functions, the improvement appeared only for part of problems in the study. Tvrdík [17] claimed that the use of both types of crossover together makes DE algorithm more robust.

Algorithm 2 L-SHADE algorithm

1: initialization: NP^{init}, NP^{min}, circle memories M_F and M_{CR}, archive $A = \emptyset$
2: $NP := NP^{init}$
3: generate an initial population $P = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{NP})$
4: evaluate $f(\mathbf{x}_i)$, $i = 1, 2, \ldots, NP$
5: **while** stopping condition not reached **do**
6: set S_F and S_{CR} empty; $Q = \emptyset$
7: **for** $i = 1$ to NP **do**
8: generate F and CR, use circle memories M_F and M_{CR}
9: generate a trial vector $\mathbf{y}$
10: evaluate $f(\mathbf{y})$
11: **if** $f(\mathbf{y}) < f(\mathbf{x}_i)$ **then**
12: save F and CR into S_F and S_{CR}
13: insert $\mathbf{x}_i$ into archive A
14: **end if**
15: **if** $f(\mathbf{y}) \leq f(\mathbf{x}_i)$ **then**
16: insert $\mathbf{y}$ into new generation Q
17: **else**
18: insert $\mathbf{x}_i$ into new generation Q
19: **end if**
20: **end for**
21: $P := Q$
22: modify circle memories if needed, use S_F and S_{CR}
23: $NP_{old} := NP$, recompute size of population NP, eq. (8)
24: **if** $NP < NP_{old}$ **then**
25: remove superfluous points from population
26: **end if**
27: **end while**

Weber and Neri [19] designed a new type of crossover called contiguous crossover. The crossover is similar to exponential crossover and it was shown on a benchmark set that the DE algorithm with the contiguous crossover is either of the same performance or of slightly better performance than DE with binomial crossover in the paper. Based on these works we decided to study a possibility to improve the efficiency of L-SHADE algorithm by employing the other type of crossover, the exponential one, in this paper.

The binomial crossover in L-SHADE can be replaced by exponential crossover or both crossovers could be used together in the algorithm. Proposed versions of L-SHADE algorithm follow:

- L-SHADEexp – the exponential crossover is employed instead of the binomial one,
- L-SHADEcom – both types of crossover compete.

L-SHADEexp version of L-SHADE algorithm employs the exponential crossover instead of the binomial one. In historical circle memories, it stores first parameter of distribution of F and not CR but p_m, the probability which was discussed in Section 2. The memories are labeled M_F and M_{p_m} here. So, when a CR is needed in L-SHADEexp algorithm, it generates p_m from normal distribution $N(M^k_{p_m}, 0.1)$ and then CR is computed from polynomial (4). Each value included in memory M_{p_m} is computed similarly as a value included into memory M_{CR} in L-SHADE, based on successful values of p_m. The other features are the same as for the original L-SHADE algorithm.

Binomial and exponential crossovers compete in L-SHADEcom algorithm. Four historical circle memories, M_{F_b}, M_{CR}, M_{F_e}, and M_{p_m}, and four sets are used for storing the first parameters of distributions and for storing successful values of F and CR of binomial crossover and of F and p_m of exponential crossover, respectively. The crossovers are employed in dependence on probabilities p_b and p_e during the search process.

A crossover is chosen according to current values of probabilities p_b and p_e independently for each point $\mathbf{x}_i$ before its trial point $\mathbf{y}$ is created. Both crossovers have the same probability, $p_b = p_e = 0.5$, at the beginning of the search process. Let s_{b_G} and s_{e_G} are counts of successful trial points generated by DE/current-to-pbest/1/bin and DE/current-to-pbest/1/exp DE-strategy during the generation G, respectively. They are set to $s_{b_G} = s_{e_G} = 0$ at the beginning of each generation. The first change of the probabilities p_b and p_e in the search process and after each resetting of probabilities occurs when a generation, in which the first successful trial point was generated, ends. In such case, at least one of s_{b_G}, s_{e_G} are not equal to zero and probabilities p_b and p_e can be adopted according to (9), (10). The s_b and s_e in (9) and (10) are cumulative counts of successful trial points since the beginning of the search process or since the last reset of probabilities p_b and p_e generated by DE/current-to-pbest/1/exp and DE/current-to-pbest/1/bin DE-strategy, respectively.

$$s_b = s_b + s_{b_G}, \qquad s_e = s_e + s_{e_G}, \tag{9}$$

$$p_b = \frac{s_b}{s_b + s_e}, \qquad p_e = 1 - p_b. \tag{10}$$

If p_b or p_e is less then δ after such re-computation, probabilities p_b and p_e are reset to $p_b = p_e = 0.5$ (δ is input parameter) and also cumulative

Table 1: Improvement and deterioration of L-SHADE algorithm by including exponential crossover

dimension	D=10		D=30		D=50		D=100		all	
algorithm	exp	com	exp	com	exp	com	exp	com	exp	com
improvement	9	7	5	5	3	4	1	2	18	18
deterioration			2	1	4	1	5	2	11	4

counts are reset to $s_b = s_e = 0$. We use value of $\delta = 0.1$ in our experiments. The approach of crossovers' competition used in L-SHADEcom is undertaken from competition of DE-strategies which was proposed for competitive-adaptive version of DE [14].

5. Experiments and Results

Two proposed modifications L-SHADEexp and L-SHADEcom were compared experimentally with L-SHADE algorithm on benchmark set developed for learning-based real-parameter optimization competition on CEC2015 according to conditions defined in [6]. This benchmark set includes 15 different problems of different complexity. Tests were done in dimensions $D = 10, 30, 50, 100$. 51 independent runs were curried out for each function, each dimension, and for each of studied algorithms. The algorithms were stopped according to conditions defined in [6] when *MaxFES* was reached, $MaxFES = D \times 10^4$. Tested algorithms were implemented in Matlab 2010a and this environment was used for experiments. All computations were carried out on a standard PC with Windows 7, Intel(R) Core(TM)i7-4600U CPU, 2.10GHz, 2.70GHz, 8 GB RAM.

The results of our experiments are summarized in Table 2 for $D = 10$ and $D = 30$ and Table 3 for $D = 50$ and $D = 100$. In these tables, best and worst error value, median, mean, and standard deviation of error value of 51 solutions are shown for each benchmark function. The L-SHADE algorithm is referred as *orig*, L-SHADEexp as *exp*, and L-SHADEcom as *com* in the tables. Some results are released from tables. In each released case, the algorithm found the optimum in all runs.

After experiments, we compared statistically results of each of proposed algorithms, L-SHADEexp and L-SHADEcom, with results of original L-SHADE algorithm. Results were compared by Wilcoxon two-sample test. We did the experiments in that way, because we want to

Table 2: Results of L-SHADE versions, D=10 and D=30

	D=10						D=30					
f/Alg.	Best	Worst	Median	Mean	Std	W	Best	Worst	Median	Mean	Std	W
3/orig	1.2410	20.017	20.007	17.599	6.092		20.061	20.147	20.105	20.108	0.0207	
exp	0	20.034	20.001	17.330	6.778	+	20	20.233	20.144	20.123	0.0682	−
com	0	20.020	20.003	16.901	7.275	=	20.002	20.209	20.102	20.107	0.0400	=
4/orig	1.9903	5.9714	3.9800	3.4941	1.002		17.050	31.191	25.097	24.945	3.203	
exp	0.9950	4.9748	2.9849	3.1409	0.8982	+	18.930	33.997	25.880	25.840	3.216	=
com	0	4.9748	2.9849	2.8873	1.076	+	15.931	31.841	24.878	24.461	3.458	=
5/orig	3.7571	141.20	15.539	29.952	38.81		741.77	1720.1	1242.3	1291.4	224.0	
exp	0.24982	229.91	21.825	44.752	54.35	=	1037.8	1899.8	1436.2	1437.0	224.4	−
com	0.18736	137.10	18.597	40.783	45.16	=	911.91	1764.5	1332.7	1345.9	195.6	=
6/orig	0.58746	9.2417	3.2384	3.5703	1.853		36.118	376.13	197.06	205.30	81.01	
exp	0	3.1930	0.4163	0.7743	0.7122	+	32.137	504.58	181.66	194.85	94.57	=
com	0	11.381	1.2031	1.2895	1.680	+	40.539	522.78	188.30	195.37	109.3	=
7/orig	0.06386	0.48082	0.23807	0.24376	0.09122		5.7896	7.4667	6.8496	6.8049	0.3729	
exp	0.02683	0.35244	0.09512	0.12774	0.08196	+	3.9795	7.0493	5.8237	5.7711	0.8452	+
com	0.03655	0.34547	0.12036	0.14226	0.07815	+	4.1992	7.5321	6.5885	6.4734	0.6624	+
8/orig	0.19264	2.9792	0.7688	0.9609	0.6209		15.473	270.83	52.018	55.845	38.11	
exp	5.84E-04	0.8094	0.1052	0.2100	0.2217	+	8.9945	262.85	31.222	42.878	42.21	+
com	4.47E-05	0.6461	0.0348	0.1126	0.1586	+	7.7778	100.43	33.710	38.370	18.72	+
9/orig	100	101.052	100	100.073	0.2523		105.911	108.333	107.076	107.100	0.5177	
exp	100	100	100	100	1.05E-06	+	101.552	106.51	105.908	105.722	0.8040	+
com	100	100	100	100	5.27E-06	+	104.969	106.833	106.175	106.162	0.3326	+
10/orig	140.701	179.010	143.109	148.704	10.88		516.284	741.962	616.434	623.758	56.01	
exp	140.701	152.292	140.708	141.895	2.364	+	533.549	681.669	612.689	604.962	42.55	=
com	140.701	152.292	140.708	142.302	3.429	+	516.315	761.666	614.993	619.535	53.46	=
11/orig	2.1630	4.2013	3.0709	3.0603	0.4503		316.02	623.18	556.30	527.88	82.78	
exp	1.8998	300	2.7454	8.5156	41.63	+	300.39	586.63	418.90	415.99	70.62	+
com	1.5598	301.22	2.8664	8.7553	41.78	=	300.35	613.67	492.17	471.17	93.43	+
12/orig	110.918	112.724	112.166	112.106	0.3543		108.57	110.17	109.36	109.39	0.3637	
exp	109.863	112.135	111.766	111.665	0.3870	+	107.55	109.95	108.95	108.94	0.4547	+
com	111.284	112.445	111.890	111.870	0.2944	+	108.40	110.02	109.17	109.21	0.3842	+
13/orig	0.0925	0.1072	0.0927	0.0940	0.0034		0.0104	0.0109	0.0107	0.0107	1.08E-04	
exp	0.0927	0.1072	0.0927	0.0938	0.0032	=	0.0104	0.0115	0.0107	0.0107	1.99E-04	=
com	0.0925	0.1028	0.0927	0.0942	0.0033	=	0.0104	0.0115	0.0107	0.0107	1.59E-04	=
14/orig	6662.87	6677.01	6670.66	6667.95	4.606		33760	42628	42559	41524	2863	
exp	6662.87	8706.43	6670.66	6707.38	285.6	=	33760	43477	42572	40052	4107	=
com	6662.87	6677.01	6670.66	6668.97	4.616	=	33760	43507	42564	41124	3453	−
15/orig	100	100	100	100	1.04E-13		100	100	100	100	1.54E-13	
exp	100	100	100	100	1.11E-13	=	100	100	100	100	1.15E-13	=
com	100	100	100	100	1.16E-13	=	100	100	100	100	1.28E-13	=

match the comparisons and to know which algorithm (L-SHADEexp or L-SHADEcom) is better compared to original L-SHADE algorithm. The results of curried out statistical tests are depicted in the last column W of Tables 2 and 3. The symbol + means the proposed algorithm reached statistically better results than L-SHADE algorithm. The symbol − means proposed algorithm reached statistically worse results than original algorithm and the symbol = means the null hypothesis of equality of compared results was not reject, which means the compared results are statistically the same. The significance level for all these statistical tests was set to 0.05.

Table 3: Results of L-SHADE versions, D=50 and D=100

	D=50						D=100					
f/Alg.	Best	Worst	Median	Mean	Std	W	Best	Worst	Median	Mean	Std	W
1/orig	78.652	14738	1219.4	2179.8	2739		71420.7	458246	190437	209140	78742	
exp	2.6667	13937	308.18	683.52	1971	+	78354.4	425687	179440	191968	73104	=
com	48.845	6529.6	628.30	1146.0	1371	+	76537.9	446651	180135	196822	73819	=
2/exp	0	0	0	0	0	=	2.07E-08	0.0003	1.75E-06	1.15E-05	3.95E-05	−
3/orig	20.170	20.272	20.228	20.227	0.0253		20.455	20.566	20.507	20.507	0.0248	
exp	20.014	20.363	20.291	20.273	0.0855	−	20.33	20.684	20.607	20.594	0.0682	−
com	20.070	20.309	20.235	20.235	0.0473	−	20.44	20.602	20.531	20.527	0.0343	−
4/orig	38.346	59.506	48.637	49.333	4.645		88.206	127.595	108.236	109.062	8.663	
exp	37.377	70.059	56.652	55.402	6.650	−	104.374	171.201	143.391	145.008	15.07	−
com	32.170	60.880	50.514	50.250	5.762	=	94.656	551.98	120.002	159.407	124.1	−
5/orig	2224.5	3567.9	3052.0	3003.6	284.0		9186.21	11384.5	10640.7	10620.2	495.6	
exp	2579.1	4052.2	3278.0	3296.8	345.3	−	9834.50	13177.3	11706.4	11787.3	794.5	−
com	2212.3	3687.3	2935.1	2995.7	319.1	=	9618.73	11910.4	10728.7	10716.0	496.3	=
6/orig	1108.7	2836.1	1900.9	1936.9	405.9		3421.72	6566.71	5256.00	5153.89	679.0	
exp	783.01	2987.6	1896.5	1926.0	420.2	=	3112.02	6514.91	5084.10	5003.51	722.4	=
com	941.08	2808.0	1762.7	1816.9	397.4	=	3538.54	6507.46	4878.57	4937.39	646.9	+
7/orig	39.677	41.762	40.773	40.725	0.4668		94.895	145.193	138.019	129.696	17.73	
exp	39.620	43.621	40.532	40.591	0.6663	=	96.743	145.853	136.749	124.063	19.89	=
com	38.893	42.902	40.514	40.542	0.7200	=	96.502	147.274	138.728	128.748	18.64	=
8/orig	15.663	758.21	401.87	398.05	178.4		1197.98	3897.73	2567.07	2515.14	631.2	
exp	19.681	916.09	409.65	420.22	210.6	=	1326.36	4183.82	2441.61	2526.49	573.9	=
com	129.57	976.46	395.16	424.10	222.1	=	1215.19	3818.94	2469.14	2427.91	599.0	=
9/orig	102.50	103.09	102.789	102.80	0.1308		110.513	111.694	111.143	111.102	0.3254	
exp	102.15	102.85	102.55	102.52	0.1767	+	107.8	110.451	109.122	109.130	0.5063	+
com	102.43	103.02	102.71	102.71	0.1511	+	109.009	110.974	109.808	109.883	0.4875	+
10/orig	642.96	1689.8	1212.0	1185.9	241.0		3376.98	5300.79	3972.90	3987.88	456.5	
exp	783.37	1765.7	1209.4	1240.7	254.9	=	2888.30	4936.66	3919.99	3874.98	471.00	=
com	791.32	1916.9	1221.5	1233.8	238.0	=	2704.49	4759.48	4001.3	3948.59	451.8	=
11/orig	400	453.65	421.21	416.31	15.78		433.244	655.634	514.352	514.825	41.00	
exp	400	457.41	433.12	424.47	18.62	−	412.963	693.160	560.684	559.924	59.53	−
com	400	473.15	421.21	418.78	19.97	=	447.131	666.403	510.977	526.472	52.46	=
12/orig	115.20	201.54	115.83	127.53	29.81		112.507	200.406	113.395	116.759	17.07	
exp	114.97	201.55	115.68	129.15	31.54	=	112.561	200.409	113.373	118.504	20.68	=
com	115.19	201.54	115.59	122.45	23.32	+	112.476	200.406	113.363	116.721	17.08	=
13/orig	0.0248	0.0256	0.0250	0.0251	1.60E-04		0.0613	0.0658	0.0633	0.0632	9.82E-04	
exp	0.0245	0.0257	0.0250	0.0250	2.62E-04	=	0.0603	0.0661	0.0635	0.0633	0.0014	=
com	0.0246	0.0255	0.0250	0.0250	1.70E-04	+	0.0610	0.0652	0.0633	0.0633	0.001	=
14/orig	52657	52714	52682	52682	15.55		108833	108955	108875	108874	20.46	
exp	52657	52716	52678	52673	15.87	+	108833	108955	108874	108875	19.52	=
com	52657	52727	52681	52680	16.58	=	108855	108910	108871	108873	13.44	=
15/orig	100	100	100	100	1.35E-13		100	100	100	100	1.23E-13	
exp	100	100	100	100	1.24E-13	=	100	100.388	100	100.014	0.0721	=
com	100	100	100	100	1.19E-13	=	100	100.388	100	100.008	0.0547	=

Summarization of statistically significant improvement and deterioration of L-SHADE algorithm by including exponential crossover is depicted in Table 1. The replacement of binomial crossover in L-SHADE algorithm by exponential crossover caused significant improvement for almost two thirds of benchmark functions in dimension $D = 10$, but the count is less for higher dimensions. Additionally, the count of benchmark functions, for which deterioration of results appeared, increases with increasing dimension for the L-SHADEexp algorithm. In case of

including of the crossover competition, L-SHADEcom algorithm, the count of problems, where solution is significantly better than solution of original algorithm, is less than in case of mere replacement of binomial crossover by exponential one in $D = 10$ and for higher dimensions, the count is again less than the count for dimension $D = 10$. However, there is less count of problems, which results are statistically worse than results of L-SHADE algorithm for L-SHADEcom algorithm than for algorithm L-SHADEexp. Thus, we can conclude that including of the exponential crossover into algorithm L-SHADE in the way of competition with binomial one brings better results than only the replacement of binomial crossover by exponential one, which was expected. The algorithm, in which the competition of strategy is employed, can choose appropriate crossover type to solved optimization problem or current stage of process. L-SHADEcom significantly improved the results of L-SHADE in 18 of 60 problems, significant deterioration of results appeared in 4 of 60 tested problems.

Nevertheless, the employing both type of crossover does not increase the performance of L-SHADE algorithm too substantially and does not solve the issue of algorithm's stagnation. The issue of stagnation relates not only to L-SHADE algorithm and our version L-SHADEcom of the algorithm but also to the other adaptive versions of DE. To solve the phenomenon of optimization algorithms' stagnation is not easy task. Discussing adaptive DE-versions, maintenance of population diversity which would be useful for searching different solutions in the search space stands against the convergency of algorithm.

6. Conclusion

The L-SHADE algorithm is successful version of differential evolution algorithm, only the binomial crossover is employed in the algorithm. In this paper, we studied the including of other type of crossover introduced by authors of DE algorithm, the exponential one. We proposed two versions of L-SHADE algorithm in which we used either only exponential crossover or both crossovers together and tested them on CEC2015 benchmark set. According to our results, the employing of the two types of crossover into L-SHADE algorithm is beneficial but it does not solve the issue of DE's stagnation, which remains for further work.

Acknowledgments This work was supported by the project LQ1602 IT4Innovations excellence in science and partially supported by University of Ostrava from the project SGS08/UVAFM/2016.

References

[1] J. Brest, S. Greiner, B. Boškovič, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10:646–657, 2006.

[2] S. Das, A. Ghosh, and S. S. Mullick. A switched parameter differential evolution for large scale global optimization – simpler may be better. *Proceedings of the International Conference on Soft Computing MENDEL*, pages 103–125, 2015.

[3] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15:27–54, 2011.

[4] J. J. Liang, B. Qu, and P. N. Suganthan. Problem definitions and evaluation criteria for the CEC2014 special session and competition on single objective real-parameter numerical optimization. [online] `http://www.ntu.edu.sg/home/epnsugan/`, 2013.

[5] J. J. Liang, B. Qu, and P. N. Suganthan. Ranking results of CEC2014 special session and competition on real-parameter single objective optimization, 2014.

[6] J. J. Liang, B. Y. Qu, and P. N. Suganthan. Problem definition and evaluation criteria for the CEC2015 competition on learning-based real-parameter single objective optimization, 2014.

[7] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11:1679–1696, 2011.

[8] F. Neri and V. Tirronen. Recent advances in differential evolution: A survey and experimental analysis. *Artificial Intelligence Review*, 33:61–106, 2010.

[9] A. K. Qin, V. L. Huang, and P. N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13:398–417, 2009.

[10] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[11] R. Tanabe and A. Fukunaga. Evaluating the performance of SHADE on CEC 2013 benchmark problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1952–1959, 2013.

[12] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for differential evolution. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 71–78, 2013.

[13] R. Tanabe and A. Fukunaga. Improving the search performance of SHADE using linear population size reduction. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1658–1665, 2014.

[14] J. Tvrdík. Competitive differential evolution. In R. Matoušek and P. Ošmera (Eds.), *Proceedings of the International Conference on Soft Computing MENDEL*, pages 7–12, 2006.

[15] J. Tvrdík. Adaptive differential evolution and exponential crossover. *Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT)*, pages 927–931, 2008.

[16] J. Tvrdík. Adaptation in differential evolution: A numerical comparison. *Applied Soft Computing*, 9(3):1149–1155, 2009.

[17] J. Tvrdík. Self-adaptive variants of differential evolution with exponential crossover. *Analele of West University Timisoara, Series Mathematics-Informatics*, 47:151–168, 2009.

[18] J. Tvrdík and R. Poláková. Competitive differential evolution applied to CEC 2013 problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1651–1657, 2013.

[19] M. Weber and F. Neri. Contiguous binomial crossover in differential evolution. *Lecture Notes in Computer Science*, 7269:145–153, 2012.

[20] D. Zaharie. Influence of crossover on the behavior of differential evolution algorithms. *Applied Soft Computing*, 9:1126–1138, 2009.

[21] J. Zhang and A. C. Sanderson. JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13:945–958, 2009.

ENHANCED SHADE AND REAL-WORLD OPTIMIZATION PROBLEMS

Petr Bujok, Josef Tvrdik
Department of Computer Science, University of Ostrava, Czech Republic
petr.bujok@osu.cz, josef.tvrdik@osu.cz

Abstract An enhanced adaptive differential evolution algorithm is described and applied to the CEC 2011 test suite of real-world optimization problems. The new algorithm combines success-history based adaptation known from SHADE and the competition of the various strategies in differential evolution. The comparison of the newly proposed algorithm (called SHADE4) with the algorithm winning in CEC 2011 competition shows that the new algorithm performs significantly better in 11 out of 22 test problems and worse only in four problems. The performance of SHADE4 was also significantly better than the original SHADE. The new algorithm is almost control-parameter free, which is helpful to its usage in the solution of the real-world problems.

Keywords: CEC 2011 real-world test problems, Competing strategies, Differential evolution, Experimental comparison, Success-history adaptation.

1. Introduction

A new adaptive variant of differential evolution (DE) (called SHADE4 hereafter) is applied to the real-world optimization problems collected in the CEC 2011 test suite [5]. Our experimental results are compared with the results of the standard DE/randrl/1/bin algorithm, the winner of CEC 2011 competition of algorithm [7] and the original SHADE algorithm. SHADE4 algorithm combines two adaptive mechanisms, namely the success-history based parameter adaptation used in SHADE [17] and the competition of strategies proposed in [18]. The aim of the paper is to present the performance of SHADE4 algorithm on the real-world optimization test suite and to demonstrate that the research of new adaptive variants of evolutionary algorithms is beneficial for real-world applications.

We consider the single-objective global optimization problem with the bound constraints defined as follows. The cost function to be minimized is $f(\mathbf{x}),\ \mathbf{x} = (x_1,\ x_2,\ \ldots, x_D) \in \mathbb{R}^D$. The domain of a feasible solutions Ω is constrained by bounds, a lower limit (a_j) and an upper limit (b_j), $\Omega = \prod_{j=1}^{D}[a_j,\ b_j]$, $a_j < b_j$, $j = 1,\ 2,\ \ldots, D$. The global minimum point $\mathbf{x}^*$ satisfying condition $f(\mathbf{x}^*) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in \Omega$, is the solution of the problem.

The rest of the paper is organized in the following manner. The basic scheme of DE algorithm is shown in Section 2. An adaptive SHADE, from which the new algorithm was inspired, is presented in Section 3. A newly proposed SHADE4 algorithm is described in Section 4. Settings of experiments and the results are given in Section 5 and 6. Finally, the conclusions are made in Section 7.

2. Differential Evolution

Differential evolution introduced by Storn and Price in [15] is a population-based evolutionary algorithm for problems with a real-valued cost function. The population P of the size N is developed step-by-step from a generation to a generation. DE uses evolutionary operators, i.e., mutation, crossover, and selection that are applied in the development of new generation of P.

The new trial point $\mathbf{y}$ is created from a mutant point $\mathbf{u}$ generated by using a kind of the mutation and from the current point of the population $\mathbf{x}_i$ by the application of the crossover. A combination of the mutation and the crossover variant is usually called DE strategy. A better point from the pair of $\mathbf{x}_i,\ \mathbf{y}$, based on the value of the cost function, is selected for the new generation.

The DE algorithm has only a few parameters whose settings significantly influence the ability to solve various optimization problems. These control parameters and their settings have been studied intensively in recent years. A comprehensive summary of advanced results in DE research is available in [6, 11], where several kinds of the mutation and the crossover were listed and some adaptive or self-adaptive DE variants are described. Adaptive variants of DE, e.g., [1, 2, 8, 13, 23] enable the change of DE control-parameters during the run of the algorithm to the current problem without trial-and-error tuning of the control parameters. Some other adaptive DE variants like [10, 14, 21] use several DE strategies or control-parameter settings and select among them adaptively with respect to the previous progress of the search.

3. SHADE Variant of Differential Evolution

Success-History Based Parameter Adaptation for Differential Evolution (SHADE) algorithm was proposed by Tanabe and Fukunaga in 2013 [17] and proved to be the best performing DE variant in CEC 2013 competition. SHADE was derived from the original algorithm Adaptive Differential Evolution With Optional External Archive (JADE) proposed by Zhang and Sanderson in 2009 [25]. The main extension of SHADE compared to original JADE is in a history-based adaptation of the control parameters F and CR. Both JADE and SHADE variants use an efficient *current-to-pbest* mutation variant, where the new mutant vector $\mathbf{u}_i$ is generated from four mutually different individuals of P – the current individual $\mathbf{x}_i$, randomly chosen individual from the p best points $\mathbf{x}_{pbest}$, randomly selected point $\mathbf{x}_{r1}$ from P and randomly selected point $\mathbf{x}_{r2}$ from $P \bigcup A$, as it is formed in the equation (1). A unification of the current population P and archive population A of old outperformed promising solutions aims at preventing DE algorithm to stuck in the local minimum.

$$\mathbf{u}_i = \mathbf{x}_i + F_i\ (\mathbf{x}_{pbest} - \mathbf{x}_i) + F_i\ (\mathbf{x}_{r1} - \mathbf{x}_{r2}) \tag{1}$$

where the point $\mathbf{x}_{pbest}$ is randomly selected from the p best points of P and F_i is a scale factor ($0 \leq F \leq 1$) which value is adapted during the search as is described in Algorithm 1.

Differently to JADE, the parameter of p for selecting the $\mathbf{x}_{pbest}$ in SHADE is generated randomly for each point of the population from uniform distribution according to the equation (2):

$$p_i = rand[2/N,\ 0.2], \tag{2}$$

and $\mathbf{x}_{pbest}$ is a point selected from the $p_i \times 100\,\%$ best points of P. After the mutation, a binomial crossover operation with CR parameter, ($0 \leq CR \leq 1$) which value is adapted during the search as is described in Algorithm 1, is used for generating the trial point $\mathbf{y}_i$ (3).

$$\mathbf{y}_{i,j} = \begin{cases} \mathbf{u}_{i,j} & \textit{if } rand_j(0,1) \leq CR \textit{ or } j = rand(1,D) \\ \mathbf{x}_{i,j} & \textit{otherwise.} \end{cases} \tag{3}$$

The changed coordinates are dispersed uniformly over the dimensions $1\ , 2, \ldots,\ D$. The crossover operator combines selected elements of mutant vector $\mathbf{u}_i$ and the current individual $\mathbf{x}_i$ in order to get a trial vector $\mathbf{y}_i$. SHADE algorithm uses only *current-to-pbest/bin* DE strategy.

If the function value of the newly composed trial point $f(\mathbf{y}_i)$ is better than the $f(\mathbf{x}_i)$, the new point $\mathbf{y}_i$ replaces the old one and the old solution

$\mathbf{x}_i$ is inserted into archive A. The values of the successfully used control parameters F and CR are also stored into auxiliary memories S_F and S_{CR}.

When the size of the archive exceeds the size of the population P, excessive randomly chosen points are deleted from the archive A. The purpose of the archive is to store old solutions from the previous generations and use them for the mutation according to (1).

The circle memories M_F and M_{CR} for generating new values of the control parameters F and CR are updated on the current position t, based on the stored successful values from auxiliary memories S_F and S_{CR}. In particular, the new value of M_{CR} is computed as a weighted arithmetic mean of the current values in S_{CR} (4) with the weights w_k (6) and the new value of M_F is computed as weighted Lehmer mean of the current values from S_F (5).

$$M_{F,t} = \frac{\sum_{k=1}^{|S_F|} w_k \; S_{F,k}^2}{\sum_{k=1}^{|S_F|} w_k \; S_{F,k}}, \tag{4}$$

$$M_{CR,t} = \sum_{k=1}^{|S_{CR}|} w_k \; S_{CR,k}, \tag{5}$$

$$w_k = \frac{\Delta f_k}{\sum_{j=1}^{|S_{CR}|} \Delta f_j}. \tag{6}$$

The values of Δf_k are computed as a difference between the cost function of the current point $\mathbf{x}_k$ and the new trial point $\mathbf{y}_k$. The values of M_F and M_{CR} remain the same if no successful point was created in the last generation. Update of values in M_F and M_{CR} is carried out at the position t, $t = 1$ at the beginning of the search and t is incremented by one after each updating of an element of M_F and M_{CR}. If $t > H$ then t is reset to $t = 1$, where H is the size of M_F and M_{CR}.

SHADE algorithm was the best performing DE variant in CEC 2013 competition [16]. However, even SHADE fails in some hard problems defined by composition of several multi-modal functions, especially in rotated functions of high level of dimension.

4. Competing Strategies in SHADE Algorithm

We studied different types of mutation and crossover. In SHADE, only one DE strategy, namely the *current-to-pbest/bin* DE strategy with the binomial crossover (3) is used. Some results [19, 24] show that the exponential type of crossover (9) or a different variant of mutation can be efficient in problems where the standard variant with binomial crossover

performs poorly. It was shown that even only replacement of the binomial crossover with the exponential one can increase performance of the SHADE algorithm [4]. Based on these results, we propose an enhanced SHADE algorithm with the competition of DE strategies. We suppose that the competition of different DE strategies can increase the efficiency of the search in situations, where variant with only one strategy fails.

The scheme for selecting variant of the DE strategies is taken from the competitive DE [18, 20]. Let us have K strategies and each DE strategy has its value of probability q_k to be used and the values of q_k are updated according to the success of the corresponding strategy in previous generations. Thereafter, a more successful strategy is used more likely than a strategy which is rarely able to generate successful individuals.

$$q_k = \frac{n_k + n_0}{\sum_{j=1}^{K} (n_j + n_0)}, \tag{7}$$

where q_k is probability of use of the kth DE strategy, n_k is the current count of the kth DE strategy successes, n_0 is an input parameter to prevent from a dramatic change in q_k and all probabilities are reset to starting uniformly distributed values if any q_k decreases below a given δ, $\delta > 0$.

Beside the *current-to-pbest/bin* strategy used in SHADE the strategies using different mutation and different crossover are selected for competition. The mutation (called *randrl/1*) which was proposed by Kaelo and Ali [9] has appeared efficient [3, 12, 19]. This mutation scheme (8) is based on the popular *rand/1* mutation, where three mutually different points $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ that are randomly selected from P.

$$\mathbf{u} = \mathbf{r}_1^x + F\,(\mathbf{r}_2^x - \mathbf{r}_3^x), \tag{8}$$

where in contrast to *rand/1* the point $\mathbf{r}_1^x$ is tournament best among $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$, i.e., $f(\mathbf{r}_1^x) \leq f(\mathbf{r}_j^x),\ j = 2, 3$.

The exponential crossover in DE is defined by the following rule:

$$\mathbf{y}_{i,j} = \begin{cases} \mathbf{u}_{i,j} & \textit{for } j = \langle n \rangle_D,\ \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D \\ \mathbf{x}_{i,j} & \textit{otherwise}, \end{cases} \tag{9}$$

where the brackets $\langle\rangle_D$ represent the modulo function with modulus D. The starting position of crossover (n) is chosen randomly from $\{1, \ldots, D\}$, and L consecutive elements (counted in a circular manner) are taken from the mutant vector $\mathbf{u}_i$. Thus, L adjacent elements are changed in exponential crossover and the probability of replacing the kth element in the sequence $1, 2, \ldots, L$, $L \leq D$, decreases exponentially

with increasing k in dependence on CR, next element is replaced if $rand_j(0,1) < CR$. The relationship between CR and the mean value of L was derived by Zaharie [24].

When we combine the aforementioned types of mutation and crossover, we obtain four different DE strategies to compete. Other components of SHADE algorithm remain unchanged, i.e., archive A of outperformed good solutions and way of adaptation of the control parameters F and CR. All the strategies in the competition use the common M_F and M_{CR} memories.

The advantage of this scheme is ability to adapt the search to the currently solved problem. A pseudo-code of SHADE variant with competition of four DE strategies is illustrated in Algorithm 1. Operators randn and randc generate random numbers from normal and Cauchy distribution, respectively.

At the beginning, a population of N potential solutions is generated distributed uniformly in the search area. Then all H values of both historical memories $M_{CR,t}$ and $M_{F,t}$, $t = 1, 2, \ldots, H$ for generating the control parameters CR and F are initialized to 0.5, where the learning rate H is an input parameter. The archive A for storing the old good solutions is set as empty.

Temporary memories S_F and S_{CR} for saving the successful values of the control parameters F and CR are set empty in each generation. The values of distribution parameters needed for random updating of F and CR are chosen randomly from M_F and M_{CR} for each point of the population. F is generated from Cauchy distribution with the parameters $(M_{F,r_i}, 0.1)$ and CR is generated from the normal distribution with the mean value M_{CR,r_i} and standard deviation 0.1, where r_i is randomly selected index from $\{1, 2, \ldots, H\}$. The unfeasible values of F and CR are regenerated according to the rules of JADE algorithm [25].

After setting the control parameters of mutation and crossover, a new trial vector $\mathbf{y}_i$ is constructed applying the selected DE strategy (one of the four in the competition) to the current parent vector $\mathbf{x}_i$. It is obvious that only *current-to-pbest* mutation variant can use archive A. On the other hand, any of applied the four DE strategies can insert outperformed old parent vector $\mathbf{x}_i$ into archive A.

At the beginning of SHADE4, the counters of successes of the DE strategies are set to zero, which results in equal values $q_k = 1/K$ for $k = 1, 2, \ldots, K$. $K = 4$ is the number of the DE strategies. After each successful generating a trial point $\mathbf{y}_i$, the counter of DE strategy currently used is increased by one. After each generation, the probabilities are updated based on the counters of the successes (7).

Algorithm 1 SHADE with competition of DE strategies (SHADE4)

initialize population $P = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$
evaluate $f(\mathbf{x}_i),\ i = 1, 2, \ldots, N$
set all M_{CR} and M_F to 0.5
initialize empty archive A
while $FES < MaxFES$ **do**
  initialize S_{CR}, S_F empty
  for $i = 1, 2, \ldots, N$ **do**
    select r_i randomly from $\{1, 2, \ldots, H\}$
    $CR_i = \text{randn}(M_{CR,r_i}, 0.1)$
    $F_i = \text{randc}(M_{F,r_i}, 0.1)$
    $p_i = \text{rand}[2/N,\ 0.2]$
    generate trial vector $\mathbf{y}_i$ by selected DE strategy (roulette wheel)
    if $f(\mathbf{y}_i) \leq f(\mathbf{x}_i)$ **then**
      insert $\mathbf{y}_i$ into Q
    else
      insert $\mathbf{x}_i$ into Q
    end if
    if $f(\mathbf{y}_i) < f(\mathbf{x}_i)$ **then**
      $\mathbf{x}_i \rightarrow A$
      $CR_i \rightarrow S_{CR}$, $F_i \rightarrow S_F$
      increase counter of the DE strategy success
    end if
  end for
  if size(A) > size(P) **then**
    delete randomly chosen individuals from A
  end if
  if size$(S_{CR}) > 0$ **and** size$(S_F) > 0$ **then**
    update M_{CR} and M_F based on S_{CR} and S_F
  end if
  $P \leftarrow Q$
end while

Finally, M_F and M_{CR} memories are updated using the stored successful values in S_F and S_{CR} as it is described in Section 3. The values of M_F and M_{CR} remain the same if no successful point was created in the last generation. The main cycle of the SHADE4 algorithm is repeated until the stopping condition is achieved. Matlab source code of SHADE4 algorithm is available online on web site `www1.osu.cz/~bujok/`.

Table 1: Function values obtained from 25 runs of standard DE/randrl/1/bin

Problem	*D*	*Best*	*Worst*	*Median*	*Mean*	*Std*
T01	6	0	1.04E-20	0	4.34E-22	2.09E-21
T02	30	-9.63396	-4.61191	-7.17815	-7.18629	0.99431
T03	1	1.15E-05	1.15E-05	1.15E-05	1.15E-05	5.19E-21
T04	1	0	0	0	0	0
T05	30	-22.9184	-17.9343	-19.277	-19.5956	1.30424
T06	30	-18.3201	-11.9093	-14.0873	-14.4051	1.61769
T07	20	1.49985	1.91389	1.72156	1.72729	0.10434
T08	7	220	220	220	220	0
T09	126	288886	525927	392200	403142	52960.3
T10	12	-21.537	-21.0853	-21.3571	-21.351	0.12976
T11.1	120	2.15E+07	7.48E+08	4.62E+07	4.56E+07	1.29E+07
T11.2	240	5.06E+06	6.66E+06	5.62E+06	5.74E+06	4.78E+05
T11.3	6	15444.2	15444.2	15444.2	15444.2	5.57E-12
T11.4	13	18982.1	19691	19211	19221.7	160.566
T11.5	15	32914.3	33108.8	32972.6	32975.4	44.5107
T11.6	40	135262	142690	138479	138718	2296.81
T11.7	140	3.56E+06	3.70E+07	1.15E+07	1.24E+07	7.69E+06
T11.8	96	4.00E+06	7.65E+06	5.84E+06	5.73E+06	959208
T11.9	96	3.13E+06	8.27E+06	5.53E+06	5.82E+06	1.30E+06
T11.10	96	2.68E+06	7.24E+06	4.78E+06	4.73E+06	1.17E+06
T12	26	24.7192	32.1438	28.0203	28.1726	1.80743
T13	22	11.3761	26.1714	13.3083	14.3622	3.12482

5. Experiments

The test suite of 22 real-world problems proposed for CEC 2011 Special Session on Real-Parameter Numerical Optimization is used as a benchmark in the experimental comparison. The test functions are described in [5], including the experimental settings required for the competition and the source code of the functions is available on the web site given in this report.

Four algorithms are compared in this paper. One of them is GA-MPC [7], which is the winner of CEC 2011 competition, and its results are taken from the cited paper.

The standard DE (DE/randrl/1/bin), the original SHADE and SHADE4 algorithms are the other algorithms in the comparison. They are implemented in Matlab 2010a and this environment was used for experiments. All computations were carried out on a standard PC with Windows 7, Intel(R) Core(TM)2 CPU 6320, 1.86GHz 1.87GHz, 2GB RAM.

The control parameters of the standard DE are set to $F = 0.8$, $CR = 0.8$. The control parameters of the original SHADE and SHADE4 are set

Table 2: Function values obtained from 25 runs of GA-MPC [7]

Problem	*Best*	*Worst*	*Median*	*Mean*	*Std*
T01	0	**0**	0	**0**	0
T02	-28.4225	**-27.113**	**-27.4797**	**-27.7007**	0.46731
T03	1.15E-05	1.15E-05	1.15E-05	1.15E-05	0
T04	13.77076	14.32911	13.77076	13.8154	0.1546
T05	-36.8454	-34.1076	-35.0098	-35.0388	0.83293
T06	-29.1661	-21.2585	-27.4298	-27.4881	1.78214
T07	0.5	0.93343	0.75806	0.74841	0.12491
T08	220	220	220	220	0
T09	**466.763**	**1818.26**	**1171.82**	**1220.59**	361.119
T10	-21.8425	**-21.4757**	-21.6445	-21.7022	0.11635
T11.1	**50925.1**	**52745.0**	**52002.8**	**52054.6**	449.912
T11.2	**1.07E+06**	1.08E+06	**1.07E+06**	**1.07E+06**	1617.59
T11.3	15444.2	15444.2	15444.2	15444.2	1.75E-07
T11.4	18100.6	18375.8	18278.2	18261.0	69.8163
T11.5	32723.8	32823.0	32770.8	32769.8	26.8249
T11.6	129213	136059	133186	133230	1878.80
T11.7	1.92E+06	1.97E+06	1.96E+06	1.95E+06	14083.6
T11.8	949500	995043	969605	971289	10387.4
T11.9	972102	1.21E+06	1.05E+06	1.06E+06	57041.6
T11.10	946598	995008	975758	975109	11848.9
T12	**7.09556**	**16.9249**	**13.7260**	**12.8182**	3.24134
T13	**8.39869**	**10.8102**	**8.62031**	**9.35934**	0.94543

to the values given in Section 3 and 4 including the learning rate the same as the population size, $H = N$. The population size is set to $N = 90$ (SHADE4) and $N = 100$ (the original SHADE) for all the test problems without respect to the problem dimension. The parameters controlling the competition of DE strategies in SHADE4 are set as follows: $n_0 = 2$ and $\delta = 1/(4 * 5) = 0.05$.

The tests were carried out with 25 independent runs per each test function. The run of the algorithm stops if the prescribed amount of function evaluation $MaxFES = 150000$ is reached. The point in the terminal population with the smallest function value is the solution of the problem found in the run.

6. Results

The basic characteristics of the function values found by the compared algorithms are presented in Table 1, 2, 3 and 4. The dimensions of the test problems are shown in Table 1. The function values, where an algorithm achieved less value of the characteristic than remaining

Table 3: Function values obtained from 25 runs of SHADE4

Problem	*Best*	*Worst*	*Median*	*Mean*	*Std*	*Sign.*
T01	0	1.97E-15	0	7.98E-17	3.93E-16	≈
T02	-28.4225	-22.4173	-24.1182	-24.7689	1.52554	−
T03	1.15E-05	1.15E-05	1.15E-05	1.15E-05	5.19E-21	≈
T04	**0**	**0**	**0**	**0**	0	+
T05	**-36.8955**	**-35.5469**	**-36.5439**	**-36.5116**	0.37292	+
T06	-29.1661	**-29.1388**	**-29.1661**	**-29.165**	5.46E-03	+
T07	0.5	**0.85883**	**0.61107**	**0.62462**	0.10125	+
T08	220	220	220	220	0	≈
T09	1393.46	3207	1979.9	2104.18	571.735	−
T10	-21.8425	-21.4421	**-21.8424**	**-21.7321**	0.12753	≈
T11.1	51003.9	53613.4	52513.6	52374.8	687.639	≈
T11.2	1.07E+06	**1.08E+06**	1.07E+06	1.07E+06	2071.30	≈
T11.3	15444.2	15444.2	15444.2	15444.2	5.57E-12	≈
T11.4	18056.1	18260.4	**18114.9**	**18128.9**	46.6809	+
T11.5	**32694.6**	**32798.7**	**32746**	**32740.8**	27.8249	+
T11.6	127718	**131474**	129911	129722	1095.99	+
T11.7	1.89E+06	1.94E+06	1.90E+06	1.91E+06	12769.8	+
T11.8	**936218**	946720	939786	**940219**	2540.09	+
T11.9	941254	1.15E+06	977446	987332	47146.4	+
T11.10	**933823**	**944030**	**939680**	**939613**	2610.84	+
T12	13.185	22.8099	18.5719	18.2831	2.30866	−
T13	8.62086	21.6398	14.9047	15.6816	3.88541	−

two others, are emphasized by bold print. Notice that the standard DE/randrl/1/bin never outperforms the GA-MPC, the original SHADE and SHADE4 simultaneously. The differences in the mean performance of GA-MPC and SHADE4 were assessed statistically by two-sample t-test at the significance level of 0.01. The t-test was used due to the fact that only summary results from [7] were available for GA-MPC. The results are depicted in the last column of Table 3. The symbol of "+" denotes that SHADE4 outperforms GA-MPC significantly, the symbol of "−" marks better performance of GA-MPC, and "≈" is used when there is no significant difference between the two algorithms.

The performance of SHADE4 and the original SHADE were compared by Wilcoxon two-sample test. The results are shown in the last column of Table 4. The symbol of "+" denotes that SHADE outperforms SHADE4 significantly, the symbol of "−" marks better performance of SHADE4. SHADE4 outperformed significantly the original SHADE in 7 out of 22 test problems, while the original SHADE was better only in one problem. In the rest of the problems, the performance was not significantly different.

Table 4: Function values obtained from 25 runs of SHADE

Problem	*Best*	*Worst*	*Median*	*Mean*	*Std*	*Sign.*
T01	0.02737	11.5882	0.13659	0.75903	2.29845	−
T02	-24.4943	-21.6473	-22.9994	-23.0316	0.74605	−
T03	1.15E-05	1.15E-05	1.15E-05	1.15E-05	5.19E-21	≈
T04	0	0	0	0	0	≈
T05	-36.6566	-34.1495	-36.3187	-36.0052	0.63807	−
T06	-29.142	-28.609	-29.092	-29.0486	0.12389	−
T07	0.90641	1.23144	1.13808	1.12130	0.08344	−
T08	220	220	220	220	0	≈
T09	1141.14	4169.19	2132.08	2228.75	714.195	≈
T010	-21.8425	-21.216	-21.6444	-21.6289	0.14074	−
T11.1	51559.6	71320.6	52580.5	53225.2	3803.80	≈
T11.2	1.07E+06	1.30E+06	1.08E+06	1.10E+06	53420.6	≈
T11.3	15444.2	15444.2	15444.2	15444.2	5.57E-12	≈
T11.4	**18028.6**	**18224.9**	18139.9	18130.7	51.7471	≈
T11.5	32742.9	32867.6	32749.6	32760	28.0246	≈
T11.6	**126951**	132462	**129061**	**129231**	1585.63	≈
T11.7	**188E+06**	**1.94E+06**	**1.90E+06**	**1.91E+06**	14075.4	≈
T11.8	937267	**944437**	**939593**	940475	2161.43	≈
T11.9	**939771**	**987681**	**948911**	**952462**	12130.8	+
T11.10	934264	945774	941238	940667	2991.61	≈
T12	14.1104	20.9105	17.5435	17.5903	1.42842	≈
T13	13.1146	23.6202	20.1044	19.9439	2.87591	−

7. Conclusion

The newly proposed SHADE4 algorithm outperformed the winner of CEC 2011 competition in 11 out of 22 problems significantly and the performance was the same in 7 problems. SHADE4 was outperformed only in four test problems. Compared to the original SHADE, SHADE4 performed better in 7 problems and it was defeated only in one problem. The results demonstrate that the development of new adaptive algorithms is beneficial to the utilization in solving the real-world optimization problems.

Moreover, SHADE4 has much less control parameters to set compared to GA-MPC when solving an optimization problem. The application of such almost control-parameter free algorithms is more convenient to the real-world users.

Acknowledgment: This work was supported by University of Ostrava from the project SGS08/UVAFM/2016.

References

[1] J. Brest, S. Greiner, B. Bosković, M. Mernik and V. Žumer. Self-adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. In *IEEE Transactions on Evolutionary Computation*, 10:646–657, 2006.

[2] J. Brest, A. Zamuda, B. Bosković, M. Maučec and V. Žumer. Dynamic optimization using self-adaptive differential evolution. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 415–422, 2009.

[3] P. Bujok and J. Tvrdík. A Comparison of Various Strategies in Differential Evolution. *Proceedings of the International Conference on Soft Computing MENDEL*, pages 48–55, 2011.

[4] P. Bujok and J. Tvrdík. Adaptive differential evolution: SHADE with competing crossover strategies. *Lecture Notes in Computer Science*, 9119:329–339, 2015.

[5] S. Das and P. N. Suganthan. Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems. Technical report, Jadavpur University, India and Nanyang Technological University, Singapore, 2010.

[6] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15:27–54, 2011.

[7] S. Elsayed, R. Sarker and D. Essam. GA with a New Multi-Parent Crossover for Solving IEEE-CEC2011 Competition Problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1034–1040, 2011.

[8] S. M. Islam, S. Das, S. Ghosh, S. Roy and P. N. Suganthan. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: (Cybernetics)*, 42(2):482–500, 2012.

[9] P. Kaelo and M. M. Ali. A numerical study of some modified differential evolution algorithms. *European Journal of Operational Research*, 169:1176–1184, 2006.

[10] R. Mallipeddi, P. N. Suganthan, Q. K. Pan and M. F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11:1679–1696, 2011.

[11] F. Neri and V. Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33:61–106, 2010.

[12] R. Poláková. A Comparison of Two Similar Mutation Operators in Differential Evolution. *Proceedings of the International Conference on Soft Computing MENDEL*, pages 1–6, 2015.

[13] W. Qian and A. Li. Adaptive differential evolution algorithm for multiobjective optimization problems. *Applied Mathematics and Computation*, 201(12):431–440, 2008.

[14] A. K. Qin, V. L. Huang and P. N. Suganthan. Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[15] R. Storn and K. V. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[16] R. Tanabe and A. Fukunaga. Evaluating the performance of SHADE on CEC 2013 benchmark problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1952–1959, 2013.

[17] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for differential evolution. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 71–78, 2013.

[18] J. Tvrdík. Competitive differential evolution. *Proceedings of the International Conference on Soft Computing MENDEL*, pages 7–12, 2006.

[19] J. Tvrdík. Exponential crossover in competitive differential evolution. *Proceedings of the International Conference on Soft Computing MENDEL*, pages 44–49, 2008.

[20] J. Tvrdík. Adaptation in differential evolution: A numerical comparison. *Applied Soft Computing*, 9(3):1149–1155, 2009.

[21] Y. Wang, Z. Cai and Q. Zhang. Differential Evolution with Composite Trial Vector Generation Strategies and Control Parameters. *IEEE Transactions on Evolutionary Computation*, 15:55–66, 2011.

[22] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[23] Z. Yang, K. Tang and X. Yao. Self-adaptive differential evolution with neighborhood search. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1110–1116, 2008.

[24] D. Zaharie. Influence of crossover on the behavior of differential evolution algorithms. *Applied Soft Computing*, 9:1126–1138, 2009.

[25] J. Zhang and A. C. Sanderson. JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13:945–958, 2009.

WORST CASE OPTIMIZATION USING CHEBYSHEV INEQUALITY

Kiyoharu Tagawa
School of Science and Engineering, Kindai University, Japan
tagawa@info.kindai.ac.jp

Abstract In real-world optimization problems, a wide range of uncertainties have to be taken into account. The presence of uncertainty leads to different results for repeated evaluations of the same solution. Therefore, users may not always be interested in the so-called best solutions. In order to find the robust solutions which are evaluated based on the predicted worst case, Worst Case Optimization Problem (WCOP) is formulated by using Chebyshev inequality from samples. Besides, a new evolutionary algorithm based on Differential Evolution is proposed to solve WCOP efficiently. The difference between the nominal solutions and the robust solutions is demonstrated through engineering design problems.

Keywords: Differential evolution, Prediction interval, Robust optimization.

1. Introduction

Considering the worst case is important, in particular, if the decision maker is very risk averse, or if the stakes are high. Therefore, we formulate Worst Case Optimization Problem (WCOP) to obtain the robust solution evaluated based on the predicted worst case. In order to predict the worst case from samples, we employ the upper bounds of the uncertain function values in the objective and constraints. In WCOP, we can specify the probability of risk with a significance level. We also show the minimum sample size necessary for the significance level.

We propose an Evolutionary Algorithm (EA) based on Differential Evolution (DE) [13] to solve WCOP efficiently. In order to save the number of samplings, or the number of repeated evaluations of the same solution, for calculating the upper bounds, the accumulative sampling [12] and the U-cut [21] are introduced into DE. Finally, we apply the proposed methodology to practical engineering design problems.

2. Related Work

Optimization in uncertain environments is an active research area in EAs [9]. A number of EAs have been reported to solve single- and multi-objective optimization problems under uncertainties [2, 3]. However, in most of the methods, the average performance has been considered.

Generally speaking, approaches for the worst case optimization can be categorized into two classes. Let $\boldsymbol{x} \in \Re^D$ be the vector of design variables x_j, $j = 1, \cdots, D$. In non-statistical approaches for the worst case optimization [17, 24], the uncertainty is given by the upper and lower bounds of each design variable x_j. Then the worst case of the function included in an objective or a constraint is evaluated based on vertex analysis. On the other hand, in statistical approaches for the worst case optimization, the design variable x_j is given by a random variable. WCOP described in this paper is a statistical approach.

In our previous papers [20, 21], we considered another WCOP in which the objective function $f(\boldsymbol{x})$ was subject to noise: $\mathcal{F}(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$. Supposing that noise ϵ was distributed normally, we predicted the upper bound of the objective function value $\mathcal{F}(\boldsymbol{x})$ with the quantile of the normal distribution. The new methodology proposed in this paper can handle two types of uncertainties: $\mathcal{F}(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{\delta}) + \epsilon$, where $\boldsymbol{\delta} \in \Re^D$ denotes the vector of perturbations, or random variables. Besides, it doesn't presume particular distribution for random variables.

DE [5, 13] is a recently developed EA. DE is arguably one of the most powerful stochastic real-parameter optimization algorithms in current use. Due to its simple but powerful searching capability, DE has been used successfully in many scientific and engineering applications [19, 25]. Some variants of DE have also been reported for optimization problems under uncertainties [11, 14, 15]. However, most of them don't evaluate solutions with the worst case, but with the average performance.

3. Prediction Interval from Samples

Chebyshev inequality [23] in (1) is well known to statisticians. If $\mathcal{F}$ is a random variable with mean μ and variance σ^2, then for $\lambda > 1$,

$$P_r(|\mathcal{F} - \mu| \geq \lambda\,\sigma) \leq \frac{1}{\lambda^2}. \tag{1}$$

Chebyshev inequality has great utility because it can be applied to completely arbitrary distributions. Let $\alpha = 1/\lambda^2$ be a significance level. Then, from (1), we can derive a prediction interval as

$$P_r([\mu - \lambda\,\sigma,\ \mu + \lambda\,\sigma] \ni \mathcal{F}) \geq 1 - \alpha. \tag{2}$$

Since the mean μ and variance σ^2 in (1) are usually unknown, they have to be estimated from samples. Let $\mathcal{F}^1$, $\mathcal{F}^2$, $\cdots$, $\mathcal{F}^N$ and $\mathcal{F}^{N+1}$ be weakly exchangeable samples from some unknown distribution such that $P_r(\mathcal{F}^1 = \mathcal{F}^2 \cdots = \mathcal{F}^N = \mathcal{F}^{N+1}) = 0$. From a set of observed N samples, the sample mean $\overline{F}$ and variance s^2 are obtained as

$$\overline{F} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{F}^n, \tag{3}$$

$$s^2 = \frac{1}{N-1} \sum_{n=1}^{N} (\mathcal{F}^n - \overline{F})^2. \tag{4}$$

By using the sample mean $\overline{F}$ and variance s^2, Saw et al. [18] have presented Chebyshev inequality estimated from N samples as

$$P_r \left(|\mathcal{F} - \overline{F}| \geq \lambda \sqrt{\frac{N+1}{N}}\, s \right) \leq \frac{1}{N+1} \left\lfloor \frac{(N+1)\,(N-1+\lambda^2)}{N\,\lambda^2} \right\rfloor, \tag{5}$$

where $\lfloor \cdot \rfloor$ denotes the floor function. By using the upper bound of the floor function, or its argument, we simplify the right-hand side as

$$P_r \left(|\mathcal{F} - \overline{F}| \geq \lambda \sqrt{\frac{N+1}{N}}\, s \right) \leq \frac{N-1+\lambda^2}{N\,\lambda^2}. \tag{6}$$

Now, we will derive a prediction interval from (6). We set κ as

$$\kappa = \lambda \sqrt{\frac{N+1}{N}}. \tag{7}$$

By using κ in (7), Chebyshev inequality in (6) is rewritten as

$$P_r(|\mathcal{F} - \overline{F}| \geq \kappa\, s) \leq \frac{N^2 - 1 + N\,\kappa^2}{N^2\,\kappa^2}. \tag{8}$$

Let us suppose that a significance level α is given as

$$\alpha = \frac{N^2 - 1 + N\,\kappa^2}{N^2\,\kappa^2}. \tag{9}$$

From (8) and (9), we can derive the prediction interval as

$$P_r([\overline{F} - \kappa\, s,\ \overline{F} + \kappa\, s] \ni \mathcal{F}) = P_r([\mathcal{F}^L,\ \mathcal{F}^U] \ni \mathcal{F}) \geq 1 - \alpha. \tag{10}$$

From (9), κ can be specified by the significance level α as

$$\kappa = \sqrt{\frac{N^2 - 1}{N\,(\alpha\, N - 1)}}. \tag{11}$$

Since $\kappa > 1$ holds from (7), the minimum sample size $N_{\min}$ is

$$N_{\min} = \left\lfloor \frac{1}{\alpha} + 1 \right\rfloor. \tag{12}$$

Incidentally, from (11), the lower bound of κ can be estimated as

$$\lim_{N\to\infty} \kappa = \lim_{N\to\infty} \sqrt{\frac{N^2-1}{N\,(\alpha\,N-1)}} = \sqrt{\frac{1}{\alpha}}. \tag{13}$$

In this paper, we call $\mathcal{F}^U$ in (10) the upper bound of $\mathcal{F}$. If we have enough number of samples $\mathcal{F}^1$, $\mathcal{F}^2$, $\cdots$, $\mathcal{F}^n$, $\cdots$, $\mathcal{F}^N$ that satisfies the condition $N \geq N_{\min}$ for a given significance level α, then we can calculate the upper bound $\mathcal{F}^U = \overline{F} + \kappa\, s$ of $\mathcal{F}$ from (3), (4), and (11).

4. Worst Case Optimization Problem (WCOP)

The traditional constrained optimization problem can be stated as

$$\left[\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) = f(x_1,\, x_2,\, \cdots,\, x_D) \\ \text{subject to} & g_m(\boldsymbol{x}) \leq 0,\ m \in \boldsymbol{I}_M = \{1,\, \cdots,\, M\} \\ & x_j^L \leq x_j \leq x_j^U,\ j = 1,\, \cdots,\, D \end{array}\right. \tag{14}$$

with the vector $\boldsymbol{x} = (x_1,\, \cdots,\, x_D) \in \Re^D$ of D design variables.

When uncertainties are introduced in the above optimization problem, functions $f(\boldsymbol{x})$ and $g_m(\boldsymbol{x})$ included in (14) are modified as

$$\left[\begin{array}{l} \mathcal{F}(\boldsymbol{x}) = f(\boldsymbol{x}+\boldsymbol{\delta}) + \epsilon = f(x_1+\delta_1,\, \cdots,\, x_D+\delta_D) + \epsilon, \\ \mathcal{G}_m(\boldsymbol{x}) = g_m(\boldsymbol{x}+\boldsymbol{\delta}) + \epsilon,\ m \in \boldsymbol{I}_M = \{1,\, \cdots,\, M\}. \end{array}\right. \tag{15}$$

where $\delta_j \in \Re$, $j = 1,\, \cdots,\, D$ denotes the disturbance to each $x_j \in \Re$, and $\epsilon \in \Re$ denotes a noise. Both $\boldsymbol{\delta}$ and ϵ are random variables.

The objective function value $\mathcal{F}(\boldsymbol{x})$ depends on a solution $\boldsymbol{x} \in \Re^D$. However, every time the solution is evaluated, different function values may be returned. Therefore, we predict the worst case of the solution $\boldsymbol{x} \in \Re^D$ by using the upper bound in (10). We evaluate the solution N ($N \geq N_{\min}$) times and make a sample set $\{\mathcal{F}^n(\boldsymbol{x}) \mid n = 1,\, \cdots,\, N\}$. Then, from the sample set, we calculate the upper bound $\mathcal{F}^U(\boldsymbol{x})$ of the objective function value $\mathcal{F}(\boldsymbol{x})$ for a given significance level α. In the same way, we calculate the upper bounds $\mathcal{G}_m^U(\boldsymbol{x})$, $m \in \boldsymbol{I}_M$ of the uncertain function values $\mathcal{G}_m(\boldsymbol{x})$ included in the inequality constraints.

The Worst Case Optimization Problem (WCOP) is formulated as

$$\left[\begin{array}{ll} \text{minimize} & \mathcal{F}^U(\boldsymbol{x}) = \mathcal{F}^U(x_1,\, x_2,\, \cdots,\, x_D) \\ \text{subject to} & \mathcal{G}_m^U(\boldsymbol{x}) \leq 0,\ m \in \boldsymbol{I}_M = \{1,\, \cdots,\, M\} \\ & x_j^L \leq x_j \leq x_j^U,\ j = 1,\, \cdots,\, D \end{array}\right. \tag{16}$$

with an arbitrary significance level α $(0 < \alpha < 1)$.

5. Differential Evolution for WCOP

5.1 Conventional Differential Evolution

Conventional DE can be applied to WCOP in (16). DE works by building a population $\boldsymbol{P}$ of vectors which is a set of possible solutions to WCOP. The initial population $\boldsymbol{x}_i \in \boldsymbol{P}$, $i = 1, \cdots, N_P$ is generated randomly. Evaluating each vector $\boldsymbol{x}_i \in \boldsymbol{P}$ N times, the upper bounds in (16), namely $\mathcal{F}^U(\boldsymbol{x})$ and $\mathcal{G}_m^U(\boldsymbol{x})$, $m \in \boldsymbol{I}_M$, are calculated. Classical DE [13] uses three control parameters: population size N_P, scale factor S_F, and crossover rate C_R. However, we employ a self-adapting mechanism of S_F and C_R [1]. Therefore, every vector $\boldsymbol{x}_i \in \boldsymbol{P}$ has its own parameter values $S_{F,i}$ and $C_{R,i}$ initialized as $S_{F,i} = 0.5$ and $C_{R,i} = 0.9$.

Within a generation of a population, each vector $\boldsymbol{x}_i \in \boldsymbol{P}$ is assigned to the target vector in turn. According to the following strategy of DE, the trial vector $\boldsymbol{u} \in \Re^D$ is generated from the target vector $\boldsymbol{x}_i \in \boldsymbol{P}$.

Through the recommendation [1], S_F and C_R are decided respectively from the parameters $S_{F,i}$ and $C_{R,i}$ of the target vector $\boldsymbol{x}_i \in \boldsymbol{P}$ as

$$S_F = \begin{cases} 0.1 + \mathrm{rand}_1\ 0.9, & \text{if} \quad \mathrm{rand}_2 < 0.1 \\ S_{F,i}, & \text{otherwise,} \end{cases} \tag{17}$$

$$C_R = \begin{cases} \mathrm{rand}_3, & \text{if} \quad \mathrm{rand}_4 < 0.1 \\ C_{R,i}, & \text{otherwise,} \end{cases} \tag{18}$$

where $\mathrm{rand}_n \in [0,\ 1]$ is a uniformly distributed random number.

Except for the target vector $\boldsymbol{x}_i \in \boldsymbol{P}$, three other distinct vectors $\boldsymbol{x}_{r1}$, $\boldsymbol{x}_{r2}$, and $\boldsymbol{x}_{r3}$ $(i \neq r1 \neq r2 \neq r3)$ are selected randomly from $\boldsymbol{P}$. From the four vectors, the j-th element $u_j \in \Re$ of $\boldsymbol{u} \in \Re^D$ is generated as

$$u_j = \begin{cases} x_{r1,j} + S_F\,(x_{r2,j} - x_{r3,j}), & \text{if} \quad (\mathrm{rand}_j < C_R) \vee (j = j_r) \\ x_{i,j}, & \text{otherwise,} \end{cases} \tag{19}$$

where the index of design variable $j_r \in [1,\ D]$ is selected randomly.

According to the direct constraint handling [13], the trial vector $\boldsymbol{u}$ is compared with the target vector $\boldsymbol{x}_i \in \boldsymbol{P}$. Exactly, if at least one of the following criteria is satisfied, $\boldsymbol{u}$ is judged to be better than $\boldsymbol{x}_i \in \boldsymbol{P}$.

- $\boldsymbol{u}$ is feasible $(\forall\, m \in \boldsymbol{I}_M;\ \mathcal{G}_m^U(\boldsymbol{u}) \leq 0)$ and $\mathcal{F}^U(\boldsymbol{u}) \leq \mathcal{F}^U(\boldsymbol{x}_i)$.
- $\boldsymbol{u}$ is feasible $(\forall\, m \in \boldsymbol{I}_M;\ \mathcal{G}_m^U(\boldsymbol{u}) \leq 0)$ and $\exists\, m \in \boldsymbol{I}_M;\ \mathcal{G}_m^U(\boldsymbol{x}_i) > 0$.
- $\boldsymbol{u}$ is infeasible $(\exists\, m \in \boldsymbol{I}_M;\ \mathcal{G}_m^U(\boldsymbol{u}) > 0)$, but $\forall\, m \in \boldsymbol{I}_M;\ \max\{\mathcal{G}_m^U(\boldsymbol{u}),\ 0\} \leq \max\{\mathcal{G}_m^U(\boldsymbol{x}_i),\ 0\}$.

As a result, if $\boldsymbol{u}$ is better than $\boldsymbol{x}_i \in \boldsymbol{P}$, $\boldsymbol{u}$ takes the place of $\boldsymbol{x}_i \in \boldsymbol{P}$. Besides, S_F in (17) and C_R in (18) are substituted for $S_{F,i}$ and $C_{R,i}$. If $\boldsymbol{u}$ is not better than $\boldsymbol{x}_i \in \boldsymbol{P}$, $\boldsymbol{u}$ is discarded. According to the asynchronous generation alternation model [7, 22], we use only one population $\boldsymbol{P}$.

5.2 Proposed Differential Evolution

Even though DE is applicable to WCOP, the multiple samplings of each vector to calculate the upper bounds are still expensive. Therefore, in order to examine a lot of vectors within a limited number of samplings, we introduce two techniques into the above DE, namely the accumulative sampling [12] and U-cut [21]. Both of the techniques have been contrived to solve multi-objective optimization problems. In this paper, we modify those techniques to solve WCOP efficiently. The proposed DE is called DEAU (DE with Accumulative sampling and U-cut), which can allocate the computing budget only to the promising solutions of WCOP.

Firstly, the accumulative sampling evaluates each $\boldsymbol{x}_i \in \boldsymbol{P}$ $N_{\min}$ times for calculating the upper bounds ($\mathcal{F}^U(\boldsymbol{x}_i)$ and $\mathcal{G}_m^U(\boldsymbol{x}_i)$, $m \in \boldsymbol{I}_M$), where $N_{\min}$ is the minimum sample size in (12). Thereafter, it re-evaluates $\boldsymbol{x}_i \in \boldsymbol{P}$ $N_{\min}$ times for taking additional samples and updates the upper bounds at regular generation intervals G_{int}. Let N_G be the sum total of evaluations of $\boldsymbol{x}_i \in \boldsymbol{P}$ depending on the current generation.

Secondly, the U-cut can judge hopeless trial vectors $\boldsymbol{u}$ only by few samplings and discard them. When a newborn $\boldsymbol{u}$ is compared with the target vector $\boldsymbol{x}_i \in \boldsymbol{P}$ at a generation, U-cut evaluates $\boldsymbol{u}$ N_G times for taking samples: $\mathcal{F}^n(\boldsymbol{u})$ and $\mathcal{G}_m^n(\boldsymbol{u})$, $m \in \boldsymbol{I}_M$, $n = 1 \cdots, N_G$. However, U-cut takes and examines those samples one by one. If at least one of the following criteria is satisfied along the way, $\boldsymbol{u}$ is judged to be worse than $\boldsymbol{x}_i \in \boldsymbol{P}$ and discarded immediately. That is because $\mathcal{F}^n(\boldsymbol{u}) < \mathcal{F}^U(\boldsymbol{u})$ and $\mathcal{G}_m^n(\boldsymbol{u}) < \mathcal{G}_m^U(\boldsymbol{u})$ are expected with a high probability $(1-\alpha)$.

- $\boldsymbol{x}_i$ is feasible ($\forall\, m \in \boldsymbol{I}_M$; $\mathcal{G}_m^U(\boldsymbol{x}_i) \leq 0$) and $\mathcal{F}^U(\boldsymbol{x}_i) \leq \mathcal{F}^n(\boldsymbol{u})$.
- $\boldsymbol{x}_i$ is feasible ($\forall\, m \in \boldsymbol{I}_M$; $\mathcal{G}_m^U(\boldsymbol{x}_i) \leq 0$) and $\exists\, m \in \boldsymbol{I}_M$; $\mathcal{G}_m^n(\boldsymbol{u}) > 0$.
- $\boldsymbol{x}_i$ is infeasible ($\exists\, m \in \boldsymbol{I}_M$; $\mathcal{G}_m^U(\boldsymbol{x}_i) > 0$), but $\forall\, m \in \boldsymbol{I}_M$; $\max\{\mathcal{G}_m^U(\boldsymbol{x}_i),\, 0\} \leq \max\{\mathcal{G}_m^n(\boldsymbol{u}),\, 0\}$.

If $\boldsymbol{u}$ survives, $\mathcal{F}^U(\boldsymbol{u})$ and $\mathcal{G}_m^U(\boldsymbol{u})$, $m \in \boldsymbol{I}_M$ are calculated. Then $\boldsymbol{u}$ is compared with $\boldsymbol{x}_i \in \boldsymbol{P}$ again in the same way with conventional DE.

6. Numerical Experiments

We demonstrated the proposed methodology using one test problem and two engineering design problems. Those were originally stated as

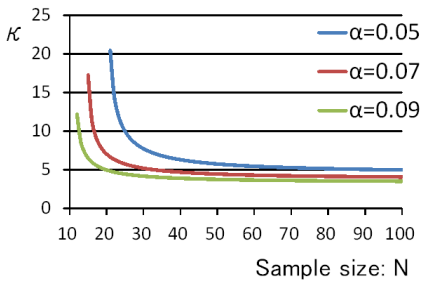


Figure 1: Change of κ for N and α.

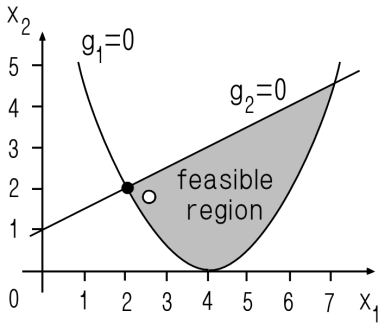


Figure 2: Feasible region of (20).

constrained optimization problems as shown in (14). We extended each of them into WCOP in which the normally distributed disturbances $\delta_j \sim \mathcal{N}(0, 0.01^2)$, $j = 1, \cdots, D$ were added to design variables x_j on the assumption that the machining error of tool was inevitable.

The conventional DE and the proposed DEAU were coded by the Java language. The population size was chosen as $N_P = 10\,D$. As the termination condition, the total number of samplings was limited to $D \times 10^5$. The generation interval $G_{int} = 10$ was used for DEAU.

6.1 Test Problem

The following test problem has a nominal solution $\boldsymbol{x}^\star = (2, 2)$ that realizes function values $f(\boldsymbol{x}^\star) = 4$, $g_1(\boldsymbol{x}^\star) = 0$, and $g_2(\boldsymbol{x}^\star) = 0$.

$$\left[\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) = f(x_1, x_2) = x_1^2 + (x_2 - 2)^2 \\ \text{subject to} & g_1(\boldsymbol{x}) = g_1(x_1, x_2) = (x_1 - 4)^2 - 2\,x_2 \le 0 \\ & g_2(\boldsymbol{x}) = g_2(x_1, x_2) = -x_1 + 2\,x_2 - 2 \le 0 \\ & -5 \le x_1,\ x_2 \le 10 \end{array}\right. \tag{20}$$

The test problem in (20) was extended to WCOP with $\alpha = 0.05$, 0.07, and 0.09. Figure 1 shows the value of κ in (11) that depends both on N ($N \ge N_{\min}$) and α. In order to obtain a robust solution $\boldsymbol{x}^\circ \in \Re^2$ of each WCOP, the proposed DEAU was used. Table 1 compares the robust solution $\boldsymbol{x}^\circ$ of each WCOP with the nominal solution $\boldsymbol{x}^\star$. From Table 1, $\mathcal{F}^U(\boldsymbol{x}^\star)$ is smaller than $\mathcal{F}^U(\boldsymbol{x}^\circ)$ in all WCOPs. However, the nominal solution is infeasible, while the robust ones are feasible.

Figure 2 illustrates the feasible region in the design variable space of the test problem. The nominal solution $\boldsymbol{x}^\star$ denoted by "•" exists on the boundary of the feasible region. On the other hand, every robust solution denoted by "○" exists on the inside of the feasible region.

From Table 1 and Fig. 2, the robust solution $\boldsymbol{x}^\circ$ parts from the boundary of the feasible region as the significance level α becomes small.

Table 1: Nominal solution and robust solutions of test problem.

α	solution	x_1	x_2	$\mathcal{F}^U(\boldsymbol{x})$	$\mathcal{G}_1^U(\boldsymbol{x})$	$\mathcal{G}_2^U(\boldsymbol{x})$
0.05	nominal: $\boldsymbol{x}^\star$	2.000	2.000	4.188	0.188	0.104
	robust: $\boldsymbol{x}^\circ$	2.061	1.979	4.444	−0.016	−0.001
0.07	nominal: $\boldsymbol{x}^\star$	2.000	2.000	4.157	0.154	0.086
	robust: $\boldsymbol{x}^\circ$	2.048	1.980	4.359	−0.002	−0.001
0.09	nominal: $\boldsymbol{x}^\star$	2.000	2.000	4.138	0.133	0.075
	robust: $\boldsymbol{x}^\circ$	2.045	1.981	4.323	−0.009	−0.006

From the value of κ in Fig. 1, the sample size of every function was set to $N = 150$.

Table 2: Nominal solution and robust solution of pressure vessel design problem.

solution	$\mathcal{F}^U(\boldsymbol{x})$	$\mathcal{G}_1^U(\boldsymbol{x})$	$\mathcal{G}_2^U(\boldsymbol{x})$	$\mathcal{G}_3^U(\boldsymbol{x})$	$\mathcal{G}_4^U(\boldsymbol{x})$
nominal: $\boldsymbol{x}^\star$	6259.854	0.049	0.047	2998.767	−39.974
robust: $\boldsymbol{x}^\circ$	6775.558	−0.004	−0.006	−1996.695	−40.421

6.2 Pressure Vessel Design Problem

The following problem is taken from [10]. A cylindrical vessel is capped at both ends by hemispherical heads as shown in Fig. 3. There are four design variables: thickness of the vessel x_1, thickness of the head x_2, inner radius x_3, and length of the vessel without heads x_4. The objective is to minimize the total cost $f(\boldsymbol{x})$ including the cost of the material, forming, and welding. The problem is stated as follows:

$$
\left[\begin{array}{ll}
\text{minimize} & f(\boldsymbol{x}) = 0.6224\,x_1\,x_3\,x_4 + 1.7781\,x_2\,x_3^2 \\
& \quad +3.1661\,x_1^2\,x_4 + 19.84\,x_1^2\,x_3 \\
\text{subject to} & g_1(\boldsymbol{x}) = -x_1 + 0.0193\,x_3 \le 0 \\
& g_2(\boldsymbol{x}) = -x_2 + 0.00954\,x_3 \le 0 \\
& g_3(\boldsymbol{x}) = -\pi\,x_3^2\,x_4 - \dfrac{4}{3}\,\pi\,x_3^3 + 1296000 \le 0 \\
& g_4(\boldsymbol{x}) = x_4 - 240 \le 0 \\
& 0.0625 \le x_1,\ x_2 \le 6.1875,\ 10 \le x_3,\ x_4 \le 200
\end{array}\right. \tag{21}
$$

Among the results of many optimization algorithms applied to the engineering design problem in (21) [4, 8], Garg [8] found the best nominal solution $\boldsymbol{x}^\star \in \Re^4$ with $f(\boldsymbol{x}^\star) \approx 5885.403$, $g_1(\boldsymbol{x}^\star) \approx -1.399 \times 10^{-6}$, $g_2(\boldsymbol{x}^\star) \approx -2.837 \times 10^{-6}$, $g_3(\boldsymbol{x}^\star) \approx -1.141$, and $g_4(\boldsymbol{x}^\star) \approx -40.019$.

The engineering design problem in (21) was extended to WCOP with $\alpha = 0.05$. By solving WCOP with DEAU, a robust solution $\boldsymbol{x}^\circ \in \Re^4$ could be obtained. Table 2 compares the robust solution $\boldsymbol{x}^\circ$ with the nominal solution $\boldsymbol{x}^\star$. From Table 2, the nominal solution is infeasible. Furthermore, since the value of $\mathcal{G}_3^U(\boldsymbol{x}^\star)$ is very large, the constraint $g_3(\boldsymbol{x}) \le 0$ in (21) is sensitive to the disturbances of design variables.

6.3 Welded Beam Design Problem

The following problem is taken from [16]. The welded beam structure is shown in Fig. 4. There are four design variables: thickness of the weld x_1, length of the welded joint x_2, width of the beam x_3, and thickness of the beam x_4. The length of beam is a constant $L = 14$. The objective is to minimize the fabricating cost of the welded beam subject to six constraints on shear stress $\tau(\boldsymbol{x})$, bending stress in the beam $h(\boldsymbol{x})$, end deflection on the beam $q(\boldsymbol{x})$, buckling load on the bar $\rho_c(\boldsymbol{x})$, and side constraints. The engineering design problem is stated as follows:

$$\left[\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) = 1.10471\, x_1^2\, x_2 + 0.04811\, x_3\, x_4\, (L + x_2) \\ \text{subject to} & g_1(\boldsymbol{x}) = \tau(\boldsymbol{x}) - 13600 \le 0 \\ & g_2(\boldsymbol{x}) = h(\boldsymbol{x}) - 3 \times 10^4 \le 0 \\ & g_3(\boldsymbol{x}) = x_1 - x_4 \le 0 \\ & g_4(\boldsymbol{x}) = 0.125 - x_1 \le 0 \\ & g_5(\boldsymbol{x}) = q(\boldsymbol{x}) - 0.25 \le 0 \\ & g_6(\boldsymbol{x}) = \rho - \rho_c(\boldsymbol{x}) \le 0 \\ & 0.1 \le x_1 \le 2,\ 0.1 \le x_2,\ x_3 \le 10,\ 0.1 \le x_4 \le 2 \end{array}\right. \tag{22}$$

$$\left[\begin{array}{l} J(\boldsymbol{x}) = \sqrt{2}\, x_1\, x_2 \left(\dfrac{x_2^2}{12} + \left(\dfrac{x_1 + x_3}{2}\right)^2\right), \quad R(\boldsymbol{x}) = \sqrt{\dfrac{x_2^2}{4} + \left(\dfrac{x_1 + x_3}{2}\right)^2} \\ \rho = 6000, \quad \tau_1(\boldsymbol{x}) = \dfrac{\rho}{\sqrt{2}\, x_1\, x_2}, \quad \tau_2(\boldsymbol{x}) = \rho \left(L + \dfrac{x_2}{2}\right) \dfrac{R(\boldsymbol{x})}{J(\boldsymbol{x})} \\ \tau(\boldsymbol{x}) = \sqrt{\tau_1(\boldsymbol{x})^2 + \dfrac{\tau_1(\boldsymbol{x})\, \tau_2(\boldsymbol{x})\, x_2}{R(\boldsymbol{x})} + \tau_2(\boldsymbol{x})^2}, \quad q(\boldsymbol{x}) = \dfrac{\rho\, L^3}{(75 \times 10^5)\, x_3^3\, x_4} \\ h(\boldsymbol{x}) = \dfrac{6\, \rho\, L}{x_3^2\, x_4}, \quad \rho_c(\boldsymbol{x}) = \dfrac{4013000 \sqrt{10}\, x_3\, x_4^3}{L^2} \left(1 - \dfrac{\sqrt{0.625}\, x_3}{2\, L}\right) \end{array}\right.$$

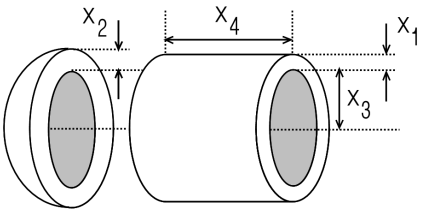


Figure 3: Pressure vessel.

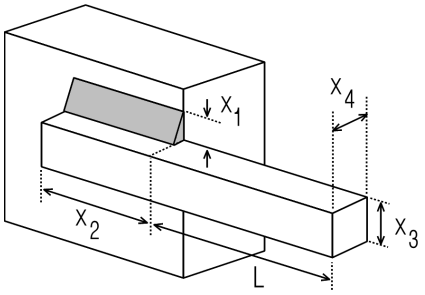


Figure 4: Welded beam.

Table 3: Nominal solution and robust solution of welded beam design problem.

	$\mathcal{F}^U(\boldsymbol{x})$	$\mathcal{G}_1^U(\boldsymbol{x})$	$\mathcal{G}_2^U(\boldsymbol{x})$	$\mathcal{G}_3^U(\boldsymbol{x})$	$\mathcal{G}_4^U(\boldsymbol{x})$	$\mathcal{G}_5^U(\boldsymbol{x})$	$\mathcal{G}_6^U(\boldsymbol{x})$
$\boldsymbol{x}^\star$	2.784	2701.887	5669.847	0.064	−0.072	−0.231	3292.867
$\boldsymbol{x}^\circ$	3.573	−85.129	−25.285	−0.007	−0.063	−0.233	−260.091

Garg [8] found the nominal solution $\boldsymbol{x}^\star$ of the problem in (22) with $f(\boldsymbol{x}^\star) \approx 2.380$, $g_1(\boldsymbol{x}^\star) \approx -0.100$, $g_2(\boldsymbol{x}^\star) \approx -1.170$, $g_3(\boldsymbol{x}^\star) \approx -6.84 \times 10^{-6}$, $g_4(\boldsymbol{x}^\star) \approx -0.119$, $g_5(\boldsymbol{x}^\star) \approx -0.234$, and $g_6(\boldsymbol{x}^\star) \approx -0.071$.

The engineering design problem in (22) was extended to WCOP with $\alpha = 0.05$. Besides disturbances $\delta_j \sim \mathcal{N}(0, 0.01^2)$ to design variables x_j, we assumed that the length of beam L was also a random value such as $L \sim \mathcal{N}(14, 0.01^2)$. By solving WCOP with DEAU, a robust solution $\boldsymbol{x}^\circ \in \Re^4$ could be obtain. Table 3 compares the robust solution $\boldsymbol{x}^\circ$ with the nominal solution $\boldsymbol{x}^\star$. From Table 3, the nominal solution is infeasible. Furthermore, the constraints $g_1(\boldsymbol{x}) \leq 0$, $g_2(\boldsymbol{x}) \leq 0$, and $g_6(\boldsymbol{x}) \leq 0$ in (22) seem to be very sensitive to the disturbances.

6.4 Comparison of DEAU with DE

The performance of DEAU was compared with the conventional DE. DEAU and DE were applied 50 times to each of the above three WCOPs with different initial populations. DE evaluated every solution $\boldsymbol{x}_i \in \boldsymbol{P}$ $N = 200$ times. The results of numerical experiments were summarized in Table 4 and Table 5. Table 4 shows the objective function value $\mathcal{F}^U(\boldsymbol{x})$ averaged over 50 runs. Similarly, Table 5 shows the number of solutions examined through the process of optimization.

From Table 4, we can confirm that DEAU outperforms conventional DE. As you can see in Table 5, by using the accumulative sampling and

Table 4: Objective function value $\mathcal{F}^U(\boldsymbol{x})$ of WCOP.

α	method	WCOP1	WCOP2	WCOP3
0.05	DEAU	4.434	6759.823	3.642
	DE	4.458	7320.667	3.796
0.07	DEAU	4.353	6615.113	3.411
	DE	4.388	7192.097	3.552
0.09	DEAU	4.306	6514.112	3.269
	DE	4.339	7003.265	3.432

- WCOP1: Test problem in (20) extended to WCOP
- WCOP2: Pressure vessel design problem in (21) extended to WCOP
- WCOP3: Welded beam design problem in (22) extended to WCOP

Table 5: Number of examined solutions.

α	method	WCOP1	WCOP2	WCOP3
0.05	DEAU	1932.0	5746.4	5224.0
	DE	1000.0	2000.0	2000.0
0.07	DEAU	2312.8	6886.4	6312.8
	DE	1000.0	2000.0	2000.0
0.09	DEAU	2683.6	8088.0	7424.0
	DE	1000.0	2000.0	2000.0

U-cut, DEAU has examined more solutions than DE in all WCOPs. As a result, DEAU could find better robust solutions than DE.

7. Conclusion

We formulated WCOP in which the worst case was estimated by using the upper bound of distributed function values. The upper bound was derived from Chebyshev inequality. Therefore, the upper bound could guarantee the worst case of unknown distributions with a significance level. For solving WCOP efficiently, we proposed a new algorithm called DEAU. We demonstrated the usefulness of the proposed methodology through one test problem and two engineering design problems.

Future work will include in-depth assessments of the methodology on a broad range of optimization problems under uncertainties. Besides, we would like to transform WCOP based on prediction intervals into various formulations including multi-objective optimization problems [6].

References

[1] J. Brest, S. Greiner, B. Bošković, M. Merink, and V. Žumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[2] R. R. Chan and S. D. Sudhoff. An evolutionary computing approach to robust design in the presence of uncertainties. *IEEE Transactions on Evolutionary Computation*, 14(6):900–912, 2010.

[3] R. F. Coelho. Probabilistic dominance in multiobjective reliability-based optimization: theory and implementation. *IEEE Transactions on Evolutionary Computation*, 19(2):214–224, 2015.

[4] C. A. C. Coello and E. M. Montes. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3):193–203, 2002.

[5] S. Das and P. N. Suganthan. Differential evolution: a survey of the state-of-the art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[6] K. Deb and H. Gupta. Introducing robustness in multi-objective optimization. *Evolutionary Computation*, 14(4):463–494, 2006.

[7] V. Feoktistov. *Differential Evolution in Search of Solutions*, Springer, 2006.

[8] H. Garg. Solving structural engineering design optimization problems using an artificial bee colony algorithm. *Journal of Industrial and Management Optimization*, 10(3):777–794, 2014.

[9] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments - a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[10] B. K. Kannan and S. N. Kramer. An augmented Lagrange multiple based method for mixed integer discrete continuous optimization and its applications to mechanical design. *Journal of Mechanical Design. Transactions of the ASME*, 116:318–320, 1994.

[11] B. Liu, X. Zhang, and H. Ma. Hybrid differential evolution for noisy optimization. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 587–592, 2008.

[12] T. Park and K. R. Ryu. Accumulative sampling for noisy evolutionary multi-objective optimization. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 793–800, 2011.

[13] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization*, Springer, 2005.

[14] S. Rahanamayan, H. R. Tizhoosh, and M. M. A. Salama. Opposition-based differential evolution for optimization of noisy problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 6756–6763, 2006.

[15] P. Rakshit, A. Konar, S. Das, L. C. Jain, and A. K. Nagar. Uncertainty management in differential evolution induced multiobjective optimization in presence of measurement noise. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(7):922–937, 2013.

[16] S. S. Rao. *Engineering Optimization.* John Wiley and Sons, 3rd ed., 1996.

[17] Z. Ren, M.-T. Pham, and C. S. Koh. Robust global optimization of electromagnetic devices with uncertain design parameters: comparison of the worst case optimization methods and multiobjective optimization approach using gradient index. *IEEE Transactions on Magnetics*, 48(2):851–859, 2013.

[18] J. G. Saw, M. C. K. Yang, and T. C. Mo. Chebyshev inequality with estimated mean and variance. *The American Statistician*, 38(2):130–132, 1984.

[19] K. Tagawa, Y. Sasaki, and H. Nakamura. Optimum design of balanced SAW filters using multi-objective differential evolution. *Lecture Notes in Computer Science*, 6457:466–475, 2010.

[20] K. Tagawa and T. Suenaga. Extended differential evolution algorithm for worst-case value minimization problems. *International Journal of Mathematical Models and Methods in Applied Sciences*, 8:262–272, 2014.

[21] K. Tagawa and S. Harada. Multi-noisy-objective optimization based on prediction of worst-case performance. *Lecture Notes in Computer Science*, 8890:23–34, 2014.

[22] K. Tagawa, H. Takeuchi, and A. Kodama. Memetic differential evolutions using adaptive golden section search and their concurrent implementation techniques. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2532–2539, 2015.

[23] P. Tchébychef. Des valeurs moyennes. *Journal de Mathématiques Pures et Appliquées*, 2(12):177–184, 1867.

[24] J.-T. Tsai. An evolutionary approach for worst-case tolerance design. *Engineering Applications of Artificial Intelligence*, 25:917–925, 2012.

[25] A. Zamuda, J. Brest, B. Bošković, and V. Žumer. Woody plants model recognition by differential evolution. *Proceedings of the 4th International Conference on Bioinspired Optimization Methods and their Applications (BIOMA)*, pages 205–215, 2010.

A HEURISTIC FOR THE JOB SHOP SCHEDULING PROBLEM

Hugo Zupan, Niko Herakovič
Faculty of Mechanical Engineering, University of Ljubljana, Slovenia
hugo.zupan@fs.uni-lj.si, niko.herakovic@fs.uni-lj.si

Janez Žerovnik
Faculty of Mechanical Engineering, University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia
janez.zerovnik@fs.uni-lj.si

Abstract Multistart local search heuristics Remove and Reinsert that is based on a simple schedule constructing heuristics is tested on several benchmark instances of the job shop scheduling problem. The heuristics provides very good near optimal solutions within reasonably short computation time. The implementation within a plant simulation software is compared to the build-in genetic algorithm.

Keywords: Digital factory, Discrete event simulation, Genetic algorithm, Job shop scheduling problem, Remove and reinsert heuristics.

1. Introduction

Optimization of assembly and handling systems and processes (AHSS) is important in terms of reducing costs, shortening lead times, delivery terms, etc., thus ensuring the competitiveness of enterprises. It has been clearly shown [25] that it can cause disturbances to reduce the overall effectiveness of equipment (OEE), which can represent up to 50% of the costs. For this reason, it is essential to optimize AHSS. Different approaches and methods have been used to optimize AHSS in order to effectively achieve the optimum, respectively nearly optimal solution [2, 9, 20].

One of the most famous optimization problems of AHSS is the Job-Shop Scheduling Problem (JSSP). Over the past few decades, a great number of studies have been made on the job-shop scheduling problem

(JSSP). JSSP can be regarded as a scheduling problem and it is one of the most challenging combinatorial optimization problems [20]. It is of both theoretical and practical interest, c.f. it is highly popular in production industry [8].

The JSSP is known to be an NP-hard optimization problem [17]. Therefore, application of metaheuristics for the JSSP is justified when looking for optimal or near optimal solutions in reasonable amount of time. This paper proposes such an algorithm, based on Remove and Reinsert algorithm (RaR) [1, 19].

The rest of the paper is organized as follows. In the next section we briefly explain the practical motivation for this research. In Section 3 the JSSP is defined, in Section 4 the metaheuristic RaR is outlined and its operation is illustrated with an example. Results of the preliminary experiments are given in Section 5. The paper ends with a summary of conclusions and ideas for future work.

2. Motivation

For conventional JSSP, it is usually assumed that all time parameters are known exactly and in deterministic values. An instance of JSSP can be described as follows: we have a set of n jobs that need to be operated on a set of m machines [22]. Each job has its own processing route; that is, jobs visit machines in different orders. Each job may need to be performed only on a fraction of m machines, not all of them. The task is to determine a processing order of all jobs on each machine that minimizes the total flow time.

Another usual assumption is that each job can be processed by at most one machine at a time and each machine can process at most one job at a time. When the process of an operation starts, it cannot be interrupted before the completion; that is, the jobs are non-preemptive. The jobs are independent; that is, there are no precedence constraints among the jobs and they can be operated in any sequence. All the jobs are available for their process at time 0. We assume that there is a buffer of unlimited size between machines for semi-finished jobs; meaning that if a job needs a machine that is occupied, the job must and can wait until the machine becomes available. There is no machine breakdown (i.e., machines are continuously available).

Besides the conventional JSSP, there is a number of variants in the literature. We mention here briefly the JSSP with setup times [15] as this was the problem on which we tested the RaR heuristic originally [27]. Setup times of machines are typically sequence dependent (or SDST), that is, the magnitude of setup strongly depends on both current and

immediately processed jobs on a given machine. For example, this may occur in a painting operation, where different initial paint colours require different levels of cleaning when being followed by other paint colours. We also assume that setup is non-anticipatory, meaning that the setup can only begin when the job and the machine are both available.

A lot of different methods and metaheuristics were used for solving the JSSP, from very simple heuristics based on priority rules to more complex methods. For example, we only mention here applications of Tabu-search (TS) [18, 21] and Simulated Annealing [3], which are among the most popular local search based metaheuristics used in combinatorial optimization. In the literature, there are also reports on applications of more advanced metaheuristics such as Genetic algorithms [10, 16], and recently popular heuristics based on Swarm intelligence including the Firefly algorithm [6], particle swarm optimization [6], Cuckoo search [13], the artificial bee colony algorithm [12, 26], ant colony optimization [6], and others.

In contrast to above metaheuristics that appear to be rather complex, in the sense that some very specific knowledge is needed for implementation and in particular for parameter tuning, our attempt was to design a conceptually simple heuristic. Therefore we started with a simple solution construction heuristic and used the same idea to define procedures for generating a new feasible solution with a perturbation of a previously given feasible solution. This naturally leads to a local search heuristic, or, more precisely, multi-start iterative improvement heuristic. As already mentioned, we initially tested the ideas on JSSP with setup times, and, as the heuristic has proven to be surprisingly competitive on JSSP with start-up times, at least on our dataset, we decided to study the same type of heuristic on the conventional JSSP, which is much more extensively studied. Consequently, there are many benchmark instances available. Below we first formally define the problem, and then introduce the RaR algorithm. In particular we define several procedures that slightly differ in the neighbourhood structure they imply on the set of feasible solutions. Finally, experimental comparison of our heuristic with genetic algorithms that are built-in as an object in Siemens Tecnomatix Plant Simulation is provided on several benchmark instances.

3. Job Shop Scheduling Problem

The Job Shop Scheduling Problem (JSSP) is formally defined as follows. Given is a set of jobs $J = \{j_1, j_2, \ldots, j_n\}$ and a set of machines $M = \{m-1, m_2, \ldots, m_n\}$. Every job is assigned a set of operations $O_i = \{O_{i1}, O_{i2}, \ldots, O_{im}\}$ with processing times $p_{ab} = p(O_{ab})$. The op-

erations are ordered, i.e., there is a binary relation A that induces a linear order on the set of operations O_i. No precedence exists between operations of different jobs. At any time, only one operation can be performed on a machine, and it may not be interrupted.

A schedule is a function $S : O \rightarrow IN \cup \{0\}$ that for each operation v defines a start time $S(v)$. A schedule S is feasible if:

$$\forall v \in O : S(v) \geq 0$$

$$\forall v, w \in O, (v, w) \in A : S(v) + p(v) \leq S(w)$$

$$\forall v, w \in O, v \neq w, M(v) = M(w) : S(v)+p(v) \leq S(w) \vee S(w)+p(w) \leq S(v)$$

The length of a schedule S is $len(S) = \max_{v \in O}(S(v)+p(v))$. The goal is to find an optimal schedule, that is a feasible schedule of minimum length, $\min(len(S))$.

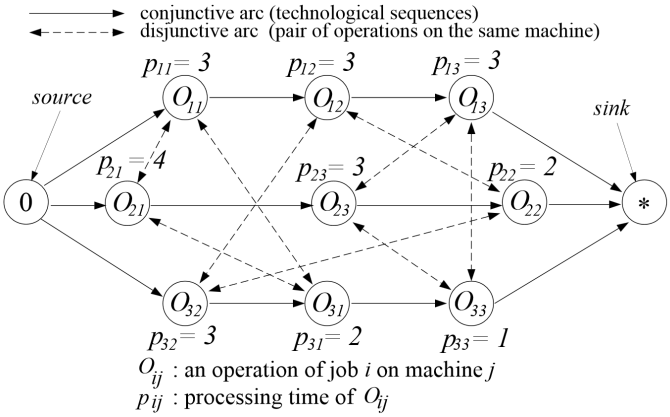


Figure 1: A disjunctive graph of a 3 × 3 problem [24].

An instance of the JSSP can be represented by means of a disjunctive graph $G = (V, C \cup D)$ where: V is a set of nodes representing operations of the jobs together with two special nodes, a source(0) and a sink(*), representing the beginning and the end of the schedule, respectively. C is a set of conjunctive arcs representing partial order of the operations. D is a set of disjunctive arcs (edges) representing pairs of operations that must be performed on the same machines. The processing time for each operation is the value attached to the corresponding node. Figure 1 shows the disjunctive graph for a simple example of JSSP with three jobs consisting of three operations each [24].

Often, the buffers (waiting queues) at machines are some standard queues, cf. First-In-First-Out (FIFO). Note that in this case, the schedule at a particular machine is fully determined by the arrival times of

the jobs. Consequently, assuming that the order of processing jobs at a machine is determined by the arrival times, it is sufficient to consider the order of jobs entering the virtual machine at the source. In this paper we study this variant of the problem.

4. Remove and Reinsert (RaR) Algorithm

Our algorithm is inspired by some applications of several similar heuristics that appeared under various names. These heuristics were successfully applied to the probabilistic traveling salesman problem (PTSP) [28], the asymmetric traveling salesman problem (ATSP) [1] and to the classical resource-constrained project scheduling problem (RCPSP) [19]. It may be rather surprising that such a simple heuristic outperforms much more complicated metaheuristics such as (in this study) a commercial implementation of a genetic algorithm. On the other hand, we think that this phenomena is not that unexpected, see [29] and the references there. In other words, as the basic idea of the heuristic is very simple and somehow natural for the particular problem, the present authors speculate that this may in fact be a reason for good results.

RaR can be regarded as a local search based on RaR neighbourhood or as a constructive heuristic. For example, on the traveling salesman problems (TSP, ATSP, PTSP) a new neighbour of a given solution is obtained by first removing a number of cities from the tour, and then reinserting them one by one into the best position that does not change the relative order of the other cities. The tour constructing heuristic on TSP (ATSP, PTSP) starts with a small subset of cities, computes their optimal permutation, and then inserts the other cities in arbitrary order. An iteration of iterative improvement consists of first removing some of the cities and then reinserting them in arbitrary order. The results on PTSP were encouraging [28], and extremely good on ATSP [1], where they were competitive with the best known heuristics of the same type. At the time, this was rather surprising as TSP is one of the most extensively studied problems in combinatorial optimization and operational research. On the resource-constrained project scheduling problem (RCPSP), the same idea with some obvious adaptations proved to be competitive with the best heuristics for the problem [19].

On the JobShop Scheduling Problem (JSSP) that is studied here, we apply the basic ideas above as follows. Below we first explain the basic neighbourhood which corresponds to a perturbation of a feasible solution into a new feasible solution. Given a parameter k and a feasible solution, k jobs are removed and reinserted, or, in this case we better say, the relative positions in the sequence of the selected k jobs may change. In

contrast to RaR on the traveling salesman problems, we do not remove all the jobs at the same time, but remove and reinsert the selected jobs one at a time. In a pseudo programming language, it could be written as

Procedure **GenerateNeighbourBasic(S0, k)** Returns(S1)
1. S1 = S0
2. Choose k jobs J_k
3. For i = 1 to k do
   a. Select a job j_i from J_k (J_k = J_k - j_i)
   b. Insert j_i into S1 on the best position
4. Return(S1)

Based on the basic neighbourhood given by procedure **GenerateNeighbourBasic**, several other neighbourhoods can be naturally defined. Here we first define a neighbourhood called large neighbourhood defined by procedure **GenerateNeighbourLarge** that can change the given solution substantially. The **GenerateNeighbourLarge** procedure first removes a subset of jobs, but, before reinserting them, it solves the subproblem to optimality. This implies that large k have to be used as otherwise the procedure would be very time consuming.

Procedure **GenerateNeighbourLarge(S0, k)** Returns(S1)
1. S1 = S0
2. Choose k jobs J_k
3. Remove jobs J_k from S1
4. Find optimal order of jobs J – J_k within S1
5. For i = 1 to k do
   a. Select a job j_i from J_k (J_k = J_k – j_i)
   b. Insert j_i into S1 on the best position
6. Return(S1)

Local search using different neighbourhoods is not a new idea. For example, it is extensively studied under name variable neighbourhood search [7, 14].

Note that **GenerateNeighbourLarge** can also be seen as procedure for generating an initial solution. Given any S0, **GenerateNeighbourLarge(S0,k)** is a heuristic that provides a good solution. (Note that there is some clear analogy to the well-known Arbitrary Insertion tour constructing heuristic for TSP.)

A version of **GenerateNeighbourLarge** that we use below allows selection of the k jobs to be perturbed outside the procedure. For this aim we define the jobs to be selected by their positions, therefore a dif-

ferent name for this variant:

Procedure **GenerateNeighbourLargePos(S0,P_k)** Returns(S1)
1. S1 = S0
2. Let J_k be k jobs at positions P_k
3. Find optimal order of jobs J – J_k within S1
4. For i = 1 to k do
   a. Select a job j_i at the next position among P_k
   b. Insert j_i into S1 on the first best position
5. Return(S1)

In our implementation of RaR, we run a multistart RaR based local search heuristic. However, as we avoid randomization after the initial solution is given, the sequence of selected neighbours is predefined. In more details, the heuristic used in the experiment is

1. Generate a Random initial solution S
2. While (there is time left) do
3. Repeat
4. S0 = S
5. Run a sequence of moves:
6. For w = 1 to n – m
7. P = positions w + m,. . . , n
8. S1 = **GenerateNeighbourLargePos(S, P)**
9. S = better between S and S1
10. Until S not better than S0

We conclude the Section with a list of remarks emphasizing some basic facts regarding our implementation of the heuristic.

1 We run a multistart of iterative improvements of the large neighbourhood **GenerateNeighbourLargePos**.

2 We speed up the implementation by avoiding randomization in the first experiments. In particular, the choices of the jobs that generate a subproblem are always a sequence of $(n-k)$ jobs starting at some position, say w. Again, this seemingly counterintuitive decision is based on the speculation that such subproblems may provide relatively good starting solution before reinsertion.

3 Removing from the set of jobs from solution is performed one at a time, thus all the jobs contribute to the cost of intermediate solutions. This decision was taken because we have observed that

completely removing many jobs may cause that the properties of the subproblems differ too much from the full problem and, consequently, even very good solutions of a subproblem may provide poor starting solution before reinsertion.

4 The testing environment is software program Siemens Tecnomatix Plant Simulation, which is used for evaluations of our scenarios and for comparison of our RaR algorithm with built-in genetic algorithms [8]. Therefore the most natural measure of time here is the number of scenarios. We also measure wall clock time, but note that this information is not very useful, as the software provides a user friendly front end with lots of graphics etc. that we do not control but tends to be very time consuming.

5. Results

For computational results the RaR algorithm was combined with discrete event simulation. It is well known [26] that using discrete event simulation or virtual factory is very effective tool for "what–if" scenarios, for every type of production system. In our case we have transformed JSSP with all the features and limitations into virtual factory. The idea is that the metaheuristic proposes initial and iteratively improved schedules of orders while the discrete event simulation performs "what–if" scenario for each proposed schedule thus providing the quality measure of the schedule. This process is repeated until the metaheuristic can no longer provide better schedule.

Recall that we assume that all queues are FIFO. Hence the algorithm optimizes only the schedule on the source (see Fig. 1).

The algorithm was tested on some well-known and one of the most used benchmark instances for the JSSP – LA01 to LA05 problems from Lawrence [11] and MT06, MT10 and MT20 problems from Fisher and Thompson [5]. We compared the RaR algorithm with the built-in "Siemens" genetic algorithm (SGA), which is already installed in the Siemens programming environment Plant Simulation [7]. The results are presented in Table 1. Note that both algorithms always started with the same initial scheduling $O_1, O_2, \ldots, O_n$.

From the results we see that RaR algorithm finds a very good solution in a relatively short time compared to SGA. The great advantage of RaR algorithm is that it is not necessary to store large amounts of data, since the algorithm works sequentially, and in almost every step takes only the best solution and while discarding the others.

We made also a test to see effect of different initial schedules on our RaR algorithm, to see how they effect on end results. The results showed

Table 1: Comparison of results between SGA and RaR.

	SGA		RaR	
Problem	best solution	time	best solution	time
LA01	705	13 s	705	3 s
LA02	758	17 s	778	3 s
LA03	679	21 s	681	2 s
LA04	660	10 s	660	2 s
LA05	593	6 s	593	2 s
MT06	59	5 s	59	1 s
MT10	1092	24 s	1092	4 s
MT20	1496	114 s	1496	9 s

that with different initial scheduling different best solutions are possible. The results were tested on benchmark LA02 and are presented in Table 2. Note that in 7 runs, the quality of the best solution by RaR has improved. In one of the runs it was even better than solution given by SGA.

Table 2: Effect of different initial scheduling on best solution found by RaR algorithm.

Initial scheduling	RaR best solution
O1, O2, O3, O4, O5, O6, O7, O8, O9, O10	778
O9, O8, O7, O10, O5, O3, O6, O1, O4, O2	758
O5, O8, O6, O3, O1, O7, O2, O4, O10, O9	766
O5, O8, O6, O3, O1, O9, O2, O4, O7, O10	754
O5, O9, O7, O10, O8, O3, O6, O1, O4, O2	758
O8, O6, O5, O9, O10, O7, O2, O1, O3, O4	778
O10, O9, O8, O7, O6, O5, O4, O3, O2, O1	778

As mentioned in Section 2, in real production there are setup times of machines which are sequence dependent - this problem is called Sequence Depended Setup Times JSSP (SDST JSSP) [15]. RaR algorithm was also tested for two SDST JSSP; with 10 jobs and with 100 jobs. In both cases we compared the algorithm with the genetic algorithm. The results are as presented in Table 3.

Characteristics of the genetic algorithm (GA):

- instance with 10 jobs: 50 generations: size of generation: 100;
- instance with 100 jobs: 500 generations; size of generation: 100.

According to the results of the first tests that have been carried out (reported above and some those that are not described in here), we can

Table 3: The results of GA and RaR algorithm on SDST JSSP.

Algorithm	Total time ...	for finding best solution	Quality of the best solution
Instance with 10 orders			
GA	1 min	33 sec	2468
RaR	20 sec	8 sec	2468
Instance with 100 orders			
GA	4h 5 min 22 sec	4h 5 min 22 sec	14385
RaR	29 min 30 sec	22 min 41 sec	13768

say that RaR algorithm works very well and in quick time gives good solutions.

6. Conclusion

This paper proposes Remove and Reinsert (RaR) heuristic for the job-shop scheduling problem. Preliminary results show that it provides very good solutions thus nearly minimizing the expected average flow time within a reasonable amount of calculation time.

For executions of "what–if" scenarios of the initial schedules, the discrete event simulation software Technomatix Plant Simulation was used. A comparison of RaR algorithm with the Genetic Algorithm which is built-in module in Technomatix Plant Simulation software showed that RaR algorithm finds good solution in shorter time compared to Genetic Algorithm. We have to mention that we did not change any parameters of the built-in GA. Maybe some better tuning of parameters for GA would improve its performance, but this was not possible as the software we use does not allow such user intervention.

Motivated by the promising results outlined here, we have also executed "what–if" scenarios for different priority rules inside the production processes. The results showed that by changing priority rules of processing the orders on the machines, we get even shorter flow time of all orders. Details will be given in the full paper.

In our future work, further experiments will be conducted to shorten the calculation time of getting the optimal solutions from the RaR algorithm.

Finally, as the RaR heuristic performs remarkably well on the JSSP, it may be worth considering the same idea on the other versions of the JSSP and some other NP-hard problems.

Acknowledgment: This research was supported in part by the Slovenian Research Agency.

References

[1] J. Brest and J. Žerovnik. An approximation algorithm for the asymmetric traveling salesman problem. *Ricerca Operativa*, 28:59–67, 1999.

[2] M. Debevec, M. Šimic, and N. Herakovič. Virtual factory as an advanced approach for production process optimization. *International Journal of Simulation Modelling*, 13(1):66–78, 2014.

[3] A. Elmi, M. Solimanpur, S. Topaloglu, and A. Elmi. A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. *Computers & Industrial Engineering*, 61(1):171–178, 2011.

[4] H. Eskandari, M. A. Rahaee, M. Memarpour, E. Hasannayebi, and S. A. Malek. Evaluation of different berthing scenarios in Shahid Rajaee container terminal using discrete-event simulation. *Proceedings of the Simulation Conference (WSC)*, 2013.

[5] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson (Eds.) *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[6] I. Fister Jr., X. S. Yang, I. Fister, J. Brest, and D. Fister. A Brief Review of Nature-Inspired Algorithms for Optimization. *Elektrotehniški vestnik*, 80(3):116–122, 2013.

[7] P. Hansen and N. Mladenovi. Variable Neighborhood Search Methods. In *Encyclopedia of Optimization*, pages 3975–3989, 2009.

[8] X. Hao, L. Lin, M. Gen, and K. Ohno. Effective Estimation of Distribution Algorithm for Stochastic Job Shop Scheduling Problem. *Procedia Computer Science*, 20:102–107, 2013.

[9] N. Herakovič, P. Metlikovič, and M. Debevec. Motivational lean game to support decision between push and pull production strategy. *International Journal of Simulation Modelling*, 13(4):433–446, 2014.

[10] X. W. Huang, X. Y. Zhao, and X. L. Ma. An improved genetic algorithm for jobshop scheduling problem with process sequence flexibility. *International Journal of Simulation Modelling*, 13(4):510–522, 2014.

[11] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

[12] J. Q. Li, Q. K. Pan, and M. F. Tasgetiren. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling*, 38(3):1111–1132, 2014

[13] M. K. Marichelvam, T. Prabaharan, and X. S. Yang. Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Applied Soft Computing*, 19:93–101, 2014.

[14] N. Mladenović, P. Hansen, and J. Brimberg. Sequential clustering with radius and split criteria. *Central European Journal of Operations Research*, 21(Supplement-1):95–115, 2013.

[15] B. Naderi, S. M. T. Fatemi Ghomi, and M. Aminnayeri. A high performing metaheuristic for job shop scheduling with sequence-dependent setup times. *Applied Soft Computing*, 10:703–710, 2010.

[16] N. M. Nidhiry and R. Saravanan. Scheduling optimization of a flexible manufacturing system using a modified NSGA-II algorithm. *Advances in Production Engineering & Management*, 9(3):139–151, 2014.

[17] B. Ombuki and M. Ventresca. Local search genetic algorithms for the job shop scheduling problem. *Applied Intelligence*, 21:99–109, 2004.

[18] B. Peng, Z. Lü, and T. C. E. Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53(1):154–164, 2015.

[19] I. Pesek, A. Schaerf, and J. Žerovnik. Hybrid local search techniques for the resource-constrained project scheduling problem. *Lecture Notes in Computer Science*, 4771:57–68, 2007.

[20] M. L. Pindeo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York Dordrecht, Heidelberg, London, 2012.

[21] A. Ponsich and C. A. Coello Coleo. A hybrid Differential EvolutionTabu Search algorithm for The solution of Job-Shop Scheduling Problems. *Applied Soft Computing*, 13(1):462–474, 2013.

[22] D. R. Sule. *Industrial Scheduling*. PWS Publishing Company, 1997.

[23] Tecnomatix Plant Simulation, Siemens PLM Software. `http://www.emplant.de/english/fact` [accessed on 29/02/2016].

[24] T. Yamada and R. Nakano. Job-shop scheduling. In: A. M. S. Zalzala and P. J. Fleming (Eds.) *Genetic algorithms in engineering systems*. IEE Control Engineering Series 55, pages 134–160, 1997.

[25] T. Ylipää. Correction, prevention and elimination of production disturbances. PROPER project description, Department of Product and Production Development (PPD), Chalmers University of Technology, Gothenhurg, 2002.

[26] R. Zhang, S. Song, and C. Wu. A hybrid artificial bee colony algorithm for the job shop scheduling problem. *International Journal of Production Economics*, 141(1):167–178, 2013.

[27] H. Zupan, N. Herakovič, and J. Žerovnik. A hybrid metaheuristic for job-shop scheduling with machine and sequence-dependent setup times. *Proceedings of the International Symposium on Operational Research in Slovenia (SOR)*, pages 129–134, 2015.

[28] J. Žerovnik. A Heuristics for the Probabilistic Traveling Salesman Problem. *Proceedings of the International Symposium on Operational Research in Slovenia (SOR)*, pages 165–172, 1995.

[29] J. Žerovnik. Heuristics for NP-hard optimization problems : simpler is better!? *Pre-conference proceedings of the 11th International Conference on Logistics & Sustainable Transport*, 2014.

III

APPLICATIONS

ON THE APPLICATION OF COMPLEX NETWORK ANALYSIS FOR METAHEURISTICS

Roman Šenkeřík, Michal Pluháček, Adam Viktorin, Jakub Janoštík
Department of Informatics and Artificial Intelligence, Faculty of Applied Informatics, Tomas Bata University in Zlin, Czech Republic
senkerik@fai.utb.cz

Abstract This contribution deals with the hybridisation of complex network frameworks and metaheuristic algorithms. The population is visualised as an evolving complex network that exhibits non-trivial features. It briefly investigates the time and structure development of a complex network within a run of selected metaheuristic algorithms – i.e., PSO and Differential Evolution (DE). Two different approaches for the construction of complex networks are presented herein. It also briefly discusses the possible utilisation of complex network attributes. These attributes include an adjacency graph that depicts interconnectivity, while centralities provide an overview of convergence and stagnation, and clustering encapsulates the diversity of the population, whereas other attributes show the efficiency of the network. The experiments were performed for one selected DE/PSO strategy and one simple test function.

Keywords: Complex Networks, Differential evolution, Population Dynamics, Particle swarm optimization.

1. Introduction

Currently, the utilisation of complex networks as a visualisation tool for the analysis of population dynamics for evolutionary and swarm-based algorithms is becoming an interesting open research task. The population is visualised as an evolving complex network that exhibits non-trivial features – e.g., degree distribution, clustering, centralities and in between. These features offer a clear description of the population under evaluation and can be utilised for adaptive population as well as parameter control during the metaheuristic run. The initial studies [1, 2, 13] describing the possibilities of transforming population dynamics into complex networks, were followed by the successful adaptation and

control of the metaheuristic algorithm during the run through the given complex networks frameworks [4, 9, 12].

This research represents the hybridisation of complex network frameworks using the Differential Evolution (DE) [10] and Particle Swarm Optimization (PSO) algorithms [7].

Currently, both aforementioned algorithms are known as powerful metaheuristic tools for many difficult and complex optimisation problems. A number of modern DE [8, 11] and PSO [3, 5] variants have also recently been developed.

The organisation of this paper is as follows: Firstly, the motivation and the concept of DE and PSO algorithms with a complex network are briefly described; followed by the experiments design. This is followed by graphical visualisations and the conclusions afterwards.

2. Motivation

This research is an extension and continuation of the previous successful initial experiment with the transfer of the population dynamics of the several DE variants being applied – e.g., to the flowshop scheduling problem [1] and the permutative flowshop scheduling problem [3]. This paper also extends the preliminary research [6] focused on capturing the inner dynamics of swarm algorithms in sufficient detail and in a network of appropriate size for further processing. The motivation for the research presented herein can be summarised as follows:

- To show the different approaches in building complex networks in order to capture the dynamics of evolutionary or swarm based algorithms.
- To investigate the time development of the influence of either individual selections inside a DE or communication inside a swarm transferred into the complex network.
- To briefly discuss the possible utilisation of complex network attributes – e.g., adjacency graphs, centralities, clustering, etc for adaptive population and parameter control during the metaheuristic run.

3. Complex Networks

A complex network is a graph which has unique properties - usually in the real-world graph domain. A complex network contains features which are unique to the assigned problem. These features are important markers for population used in Evolutionary/Swarm based algorithms

[2]. The following features are important for a quick analysis of the network thus created.

3.1 Degree Centrality

Degree Centrality is defined as the number of edges connected to a specific node. Degree Centrality is an important distribution hub in the network since it connects - and thereby, distributes most of the information flowing through the network. Together with the hybridisation of metaheuristic and complex network analysis, this is one of the most important features under consideration. Using Degree Centrality, one can actually analyse if stagnation or premature convergence is occurring within the population. By analysing the graphs, it can be seen that the multiple nodes are increasing (distinguished by their size), thereby emphasising their prominence in the population - and, their effect in generating better individuals.

3.2 The Clustering Coefficient

The average Clustering Coefficient for the entire network is calculated from every single local clustering coefficient for each node. The Clustering Coefficient of a node shows how concentrated the neighbourhood of that node is. Mathematically, it is defined as the ratio of the number of actual edges between neighbours to the number of potential edges between neighbours. For their utilisation in Computational Intelligence, it is also very important to analyse the distribution of a clustering coefficient within an entire network, since we can assume that it can show the population diversity, its compactness or tendency to form heterogeneous subgroups (subpopulations).

4. Metaheuristic Methods

This section contains the background of the metaheuristic algorithms PSO and DE that were used, as well as the main principles of capturing their dynamics in an evolving complex network.

4.1 Differential Evolution

DE is a population-based optimisation method that works on real-number-coded individuals [4]. DE is quite robust, fast – and effective, with global optimisation ability. There are essentially five inputs to the heuristic problem. D is the size of the problem, G_{max} is the maximum number of generations, NP is the total number of solutions, F is the

solution scaling factor and CR is the crossover factor. F and CR – taken together, form the internal tuning parameters for the heuristic.

The initialisation of the heuristic is as follows: Each solution $x_{j,i,G=0}$ is created randomly between the two bounds $x^{(lo)}$ and $x^{(hi)}$. The parameter j represents the index to the values within the solution, and parameter i indexes the solutions within the population. So, to illustrate: $x_{4,2,0}$ represents the fourth value of the second solution at the initial generation. After initialisation, the population is subjected to repeated iterations.

Within each iteration – and for a particular individual (solution), three random numbers r_1, r_2, r_3 are selected – unique to each other, and to the current indexed solution i in the population. Two solutions, $\boldsymbol{x}_{r_1,G}$ and $\boldsymbol{x}_{r_2,G}$ are selected through the r_1 and r_2 index and their values are subtracted. This value is then multiplied by F – the predefined scaling factor. This is then added to the value indexed by r_3.

However, this solution is not arbitrarily accepted. A new random number is generated – and if this random number is less than the value of CR, then the new value replaces the old value in the current solution. The fitness of the resulting solution, referred to as a "perturbed" (or trial) vector $\boldsymbol{u}_{i,G}$, is then compared with the fitness of $\boldsymbol{x}_{i,G}$. If the fitness of $\boldsymbol{u}_{i,G}$ is better than the fitness of $\boldsymbol{x}_{i,G}$, then $\boldsymbol{x}_{i,G}$ is replaced with $\boldsymbol{u}_{i,G}$; otherwise, $\boldsymbol{x}_{i,G}$ remains in the population as $\boldsymbol{x}_{i,G+1}$. Hence, the competition is only between the new *child* solution and its *parent* solution. This strategy is denoted as DE/Rand/1/bin. The trial vector for this strategy is given in (1).

$$\boldsymbol{u}_{i,G+1} = \boldsymbol{x}_{r_1,G} + F \cdot (\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G}) \tag{1}$$

4.2 The PSO Algorithm

Original PSO algorithms take their inspiration from behaviour of fish and birds [7]. The knowledge of the global best-found solution (typically denoted as *gBest*) is shared among the particles in the swarm. Furthermore, each particle has the knowledge of its own (personal) best-found solution (designated *pBest*). The last important part of the algorithm is the velocity of each particle, which is taken into account during the calculation of the particle's movement. The new position of each particle is then given by (2), where x_i^{t+1} is the new particle position; x_i^t refers to the current particle position and v_i^{t+1} is the new velocity of the particle.

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2}$$

To calculate the new velocity, the distance from *pBest* and *gBest* is taken into account along with its current velocity (3).

$$v_{ij}^{t+1} = v_{ij}^{t} + c_1 \cdot Rand \cdot (pBest_{ij} - x_{ij}^{t}) + C_2 \cdot Rand \cdot (gBest_j - x_{ij}^{t}), \quad (3)$$

where:
v_{ij}^{t+1} – New velocity of the i^{th} particle in iteration $t+1$; (component j of the dimension D).
v_{ij}^{t} – Current velocity of the i^{th} particle in iteration t; (component j of the dimension D).
$c_1, c_2 = 2$ – Acceleration constants.
pBestij – Local (personal) best solution found by the i^{th} particle; (component j of the dimension D).
gBestj – Best solution found in a population; (component j of the dimension D).
x_{ij}^{t} – Current position of the i^{th} particle; (component j of the dimension D) in iteration t.
Rand – Pseudo-random number, interval $(0, 1)$.

4.3 DE and PSO with a Complex Network Framework

In this research, the complex network approach is utilised to show the linkage between different individuals in the population. Each individual in the population can be taken as a node in the complex network graph, where its links specify the successful exchange of information in the population.

Since the internal dynamics and principles are different for evolutionary (DE) and swarm based (PSO) algorithms, two different approaches for capturing the population dynamics have been developed and tested.

In the case of the DE algorithm, an *Adjacency Graph* was used. In each generation, the node is only active for the successful transfer of information, i.e., if the individual is successful in generating a new better individual which is accepted for the next generation of the population. If the trial vector created from three randomly selected individuals (DE/Rand/1/Bin) is better than the active individual, one establishes the connections between the new created individual and the three sources; otherwise, no connections are recorded in the Adjacency Matrix.

For the PSO algorithm, the main interest is in the communications that lead to population quality improvement. Therefore, only communication leading to improvement of the particles personal best (*pBest*) was tracked. The link was created between the particle that was improved and the particle that triggered the current *gBest*'s update. This

approach creates a complex network with clusters – and can be used for particle performance evaluations. Of course, it is also possible to build an Adjacency Graph (see Section 7).

5. Experimental Design

A simple Schwefel's Test function (4) was used in this experimental research for the purpose of the generation of a complex network. Due to the limited space and focus of this paper, only one test function is used. The influences of different test functions for complex network frameworks are discussed in the Conclusion.

$$f(x) = -\sum_{i=1}^{D} x_i \sin(\sqrt{|x_i|}) \tag{4}$$

Experiments were also performed in the *C language* environment, the data from the DE algorithm was analysed and visualised using *Cytoscape* software, while data from the PSO algorithm was analysed in the *Wolfram Mathematica SW* suite.

Within the ambits of this research, only one type of experiment was performed. It utilises the maximum number of generations fixed at 100 with a population size of $NP = 50$. Two DE control parameters for mutation and crossover were set as $F = 0.5$ and $CR = 0.8$. Two acceleration constants for PSO were set as $c_1, c_2 = 2$.

Since only one run of DE or PSO algorithms were executed for this particular case-study, no statistical results related to the cost function values and no comparisons are given here, since it is not possible to compare metaheuristic algorithms only from one run.

6. Vizualisations for DE

The visualisations of complex networks are depicted in Figs 1–2 containing Adjacency Graphs for this particular case-study. The last, Fig. 3, shows the complete Complex Network Adjacency Graph for all 100 generations.

The *Degree Centrality* value is highlighted by the size of the node, and the colouring of the node is related to the *Clustering Coefficient* distribution (light coloured-lower values ranging up to the red colours – higher values). Simple analyses of the networks are given in Table 1; furthermore, it also contains the values of the total number of edges in the graph; the success rate of the evolution process in percentage, showing the ratio between the maximum possible edges in the graphs and the actual one. The theoretical maximum number of edges in the graph is given by $3 \cdot NP \cdot 10 = 1500$, i.e., the situation where every active

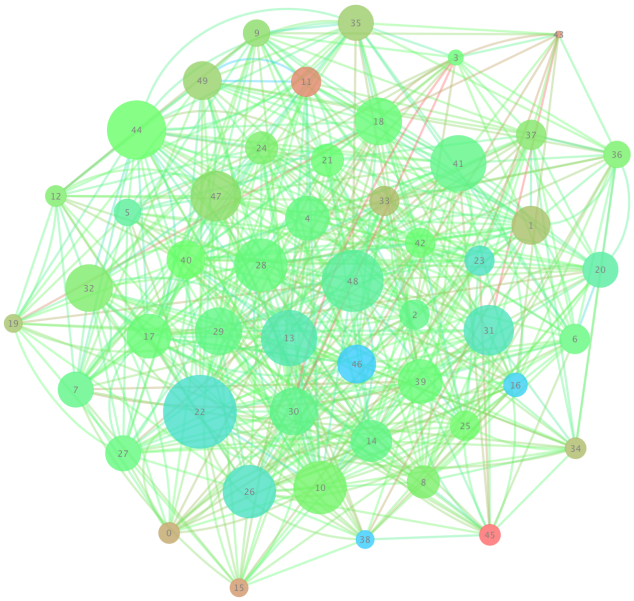


Figure 1: Complex Network Representation for DE Dynamics – Case 1: the first 10 generations.

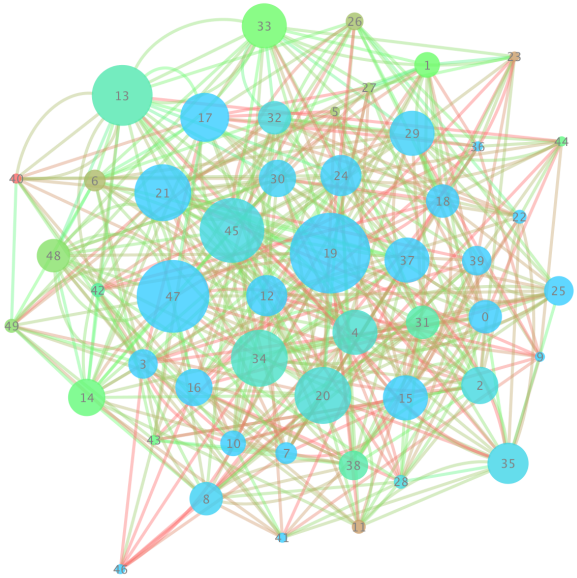


Figure 2: Complex Network Representation for DE Dynamics – Case 2: the last 10 generations.

individual in the population is replaced by a newly created one from three another individuals across the limited number of 10 generations as observed.

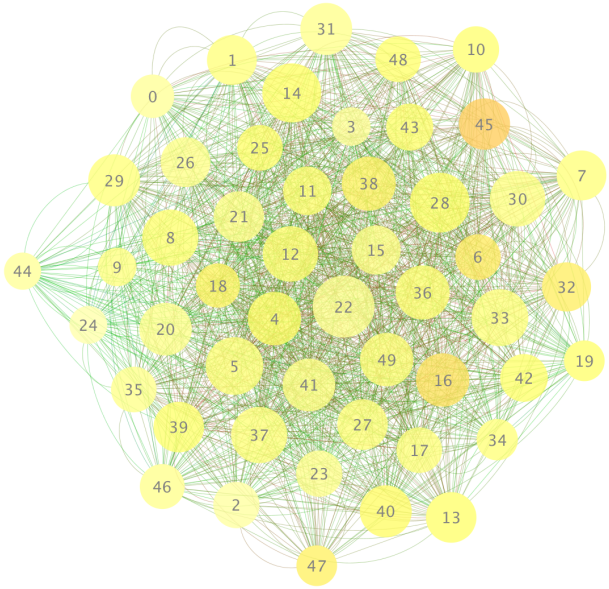


Figure 3: Complex Network Representation for DE Dynamics – Complete graph, all 100 generations.

Table 1: Shows a simple analysis of the networks for two case studies – the first and the last 10 generations.

Case.	No. of edges	Success Rate (%)	Clustering Coefficient	Network Centralization	Av. no. of Neighbours	Network Density
First 10	558	37.20	0.390	0.117	18.48	0.377
Last 10	435	29.00	0.334	0.216	14.84	0.303

7. Vizualisations for PSO

The complex network for all iterations of the PSO algorithm that was created is depicted in Fig. 4. Nodes of a similar colour represent particles with the same ID, and throughout different iterations. All links are from a particle that triggered the *gBest* update to a particle that has improved - based on that *gBest*. Due to the complexity of the Figure, it is not possible to clearly see the density of the network and links of various lengths.

A closer look at a single cluster in the network is presented in Fig. 5. The nodes' code numbers represent a particle ID and its current iteration. This way, it is possible to precisely track the development of the network and the communication that occurs within the swarm. To be more precise, from this illustrative cluster, it can be observed that a single *gBest* update led to the improvement of multiple particles in different iterations.

Alternatively, it is possible to construct an Adjacency Graph and to benefit from its statistical features – as with the DE case. The link is created between the particle that triggered the last *gBest* update and the particle that triggers a new *gBest* update. The self-loops (when a new *gBest* is found by exactly the same particle as the previous *gBest*), are omitted. The simplified example is depicted in Fig. 6. Here, for a lower level of complexity and illustration purposes, the population size was limited to 20.

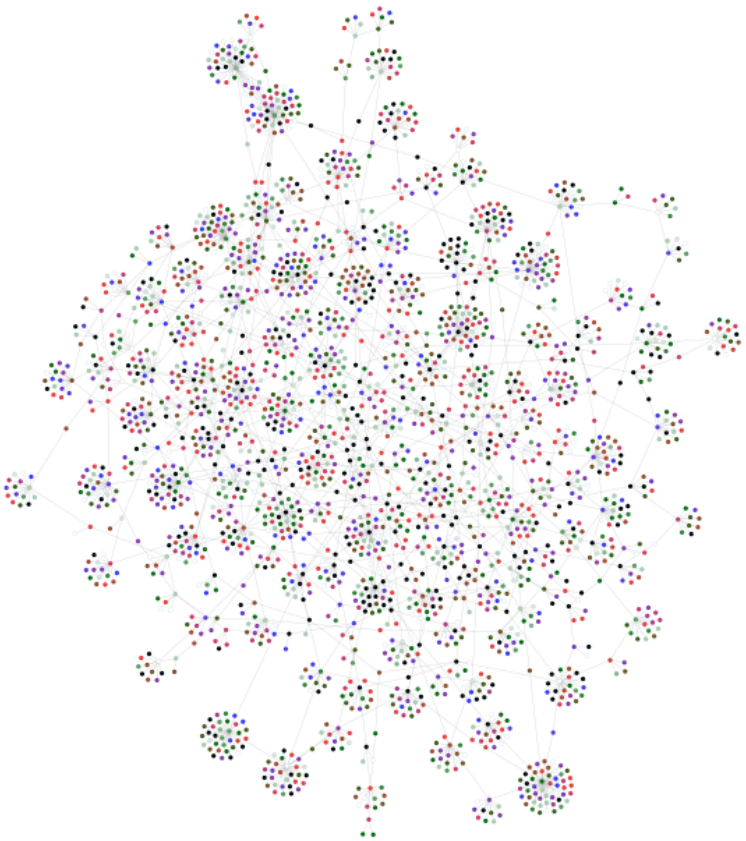


Figure 4: PSO Dynamic as a Complex Network – Complete view.

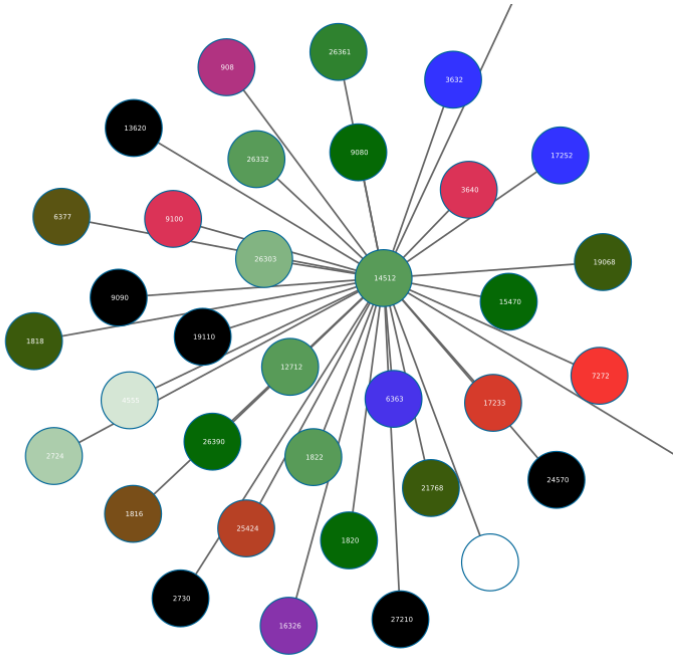


Figure 5: PSO Dynamic as a Complex Network – Close view.

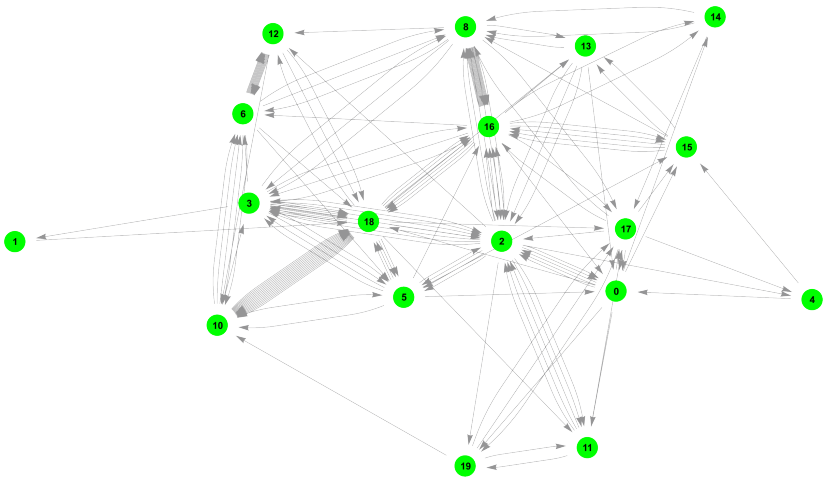


Figure 6: PSO Dynamic as an Adjacency Graph.

8. Conclusion

This work was aimed at the experimental investigation of the hybridisation of a complex network framework using DE and PSO algorithms. The population was visualised as an evolving complex network, which exhibits non-trivial features. These features provided a clear description of the population during evaluation and can be used for adaptive population and parameter control during the metaheuristic run.

The graphical and numerical data presented herein has fully manifested the influence of either time frame selection, or type of construction to the features of the complex network. These features can be used in various adaptive or learning processes. The findings can be summarised as follows:

a) Building of the Network: Since there is a direct link between parent solutions and offspring in the evolutionary algorithms, this information is used to build a complex network. In the case of swarm algorithms, the situation is a bit more difficult. It depends on the inner swarm mechanisms, but mostly, it is possible to capture the communications within the swarm during the updating of the information - based on the points of attraction. Two possible approaches are described herein, resulting in different graph visualisations and possible analyses (see Figs. 5 and 6 and Section 7).

b) Complex Network Features: A complex network created for evolutionary algorithms contains direct information about the selection of individuals and their success; therefore, many network features can be used for controlling a population during an EA run. At the beginning of the optimisation process, intensive communication occurs (Fig. 1). Later, hubs (centralities) and clusters are created (Fig. 2), and it is possible to use such information either for the injection or replacement of individuals or to modify/alternate the evolutionary strategy (see Sections 3.1 and 3.2). In the case of swarm algorithms, the communication dynamics are captured – thus the level of particle performance (usefulness) can be calculated; or alternatively, some sub-clusters and centralities of such a communication can also be identified - depending on the technique used for the transformation of swarm dynamics into the network.

c) Other Features – Randomisation, Fitness Landscape: Numerous previous experiments showed that there are no significant changes in complex network features for different test functions in the case of evolutionary algorithms. Nevertheless, the different randomization (distribution) used directly influences the network develop-

ment through the selection of individuals. Thus, through the preferences of some clusters of individuals, it is possible to temporarily simulate the different randomisation inside a metaheuristic for an entire/sub-population. Complex network construction for swarm algorithms is not distinctively sensitive to randomisation, but the capturing of communications (swarm dynamics) is sensitive to the fitness landscape. Thus, network features can be used for the raw estimation of a fitness landscape (i.e., multimodal/uni-modal, or the identification of particular benchmark function) directly.

This novel topic has brought up many new open tasks, which will be resolved in future research. Another advantage is that this complex network framework can be used almost on any metaheuristic. Moreover, especially for swarm algorithms there exist many possible ways regarding how to build a complex network.

Acknowledgment: This work was supported by the Grant Agency of the Czech Republic – GACR P103/15/06700S; and by the Internal Grant Agency of Tomas Bata University, Project No. IGA/CebiaTech/2016/007.

References

[1] D. Davendra, I. Zelinka, M. Metlicka, R. Šenkeřík, and M. Pluháček. Complex network analysis of differential evolution algorithm applied to flowshop with no-wait problem. *Proceedings of the IEEE Symposium on Differential Evolution (SDE)*, pages 1–8, 2014.

[2] D. Davendra, I. Zelinka, R. Šenkeřík, and M. Pluháček. Complex Network Analysis of Evolutionary Algorithms Applied to Combinatorial Optimisation Problem. *Proceedings of the Fifth International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA)*, pages 141–150, 2014.

[3] A. Engelbrecht. Heterogeneous particle swarm optimization. *Proceedings of the 7th International Conference on Swarm Intelligence*, pages 191–202, 2010.

[4] P. Gajdos, P. Kromer, and I. Zelinka. Network Visualization of Population Dynamics in the Differential Evolution. *Proceedings of the IEEE Symposium Series on Computational Intelligence*, pages 1522–1528, 2015.

[5] M. Imran, H. Jabeen, M. Ahmad, Q. Abbas, and W. Bangyal. Opposition based PSO and mutation operators. *Proceedings of the 2nd International Conference on Education Technology and Computer (ICETC)*, pages V4-506–508, 2010.

[6] J. Janostik, M. Pluháček, R. Šenkeřík, I. Zelinka. Particle Swarm Optimizer with Diversity Measure Based on Swarm Representation in Complex Network. *Proceedings of the Second International Afro-European Conference for Industrial Advancement (AECIA 2015)*, pages 561–569, 2016.

[7] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[8] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679–1696, 2011.

[9] M. Metlicka and D. Davendra. Ensemble centralities based adaptive Artificial Bee algorithm. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 3370–3376, 2015.

[10] K. V. Price. An Introduction to Differential Evolution. In D. Corne, M. Dorigo, and F. Glover (Eds.) *New Ideas in Optimization*, pages 79–108. McGraw-Hill 1999.

[11] A. K. Qin, V. L. Huang, and P. N. Suganthan. Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[12] L. Skanderova and T. Fabian. Differential evolution dynamics analysis by complex networks. *Soft Computing*, 1–15, 2015.

[13] I. Zelinka, D. Davendra, J. Lampinen, R. Šenkeřík, and M. Pluháček. Evolutionary algorithms dynamics and its hidden complex network structures. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 3246–3251, 2014.

A PARTICLE SWARM OPTIMIZATION HYPER-HEURISTIC FOR THE DYNAMIC VEHICLE ROUTING PROBLEM

Michał Okulewicz, Jacek Mańdziuk
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
M.Okulewicz@mini.pw.edu.pl, J.Mandziuk@mini.pw.edu.pl

Abstract This paper presents a method for choosing a Particle Swarm Optimization based optimizer for the Dynamic Vehicle Routing Problem on the basis of the initially available data of a given problem instance. The optimization algorithm is chosen on the basis of a prediction made by a linear model trained on that data and the relative results obtained by the optimization algorithms. The achieved results suggest that such a model can be used in a hyper-heuristic approach as it improved the average results, obtained on the set of benchmark instances, by choosing the appropriate algorithm in 82% of significant cases. Two leading multi-swarm Particle Swarm Optimization based algorithms for solving the Dynamic Vehicle Routing Problem are used as the basic optimization algorithms: Khouadjia's et al. Multi–Environmental Multi–Swarm Optimizer and authors' 2–Phase Multiswarm Particle Swarm Optimization.

Keywords: Dynamic vehicle routing problem, Hyper-heuristic, Particle swarm optimization.

1. Introduction

Dynamic transportation problems have been considered in the literature by Psaraftis [20, 21] since 1980. After the introduction of a set benchmarks by Kilby [14] and Montemanni [15], several meta-heuristic based algorithms have been developed in order to solve the problem, including Ant Colony Optimization (ACS) [6, 15], Genetic Algorithm (GA) [9, 7], Tabu Search (TS) [9], and Particle Swarm Optimization (PSO) [13, 17].

Although some of the works [7, 9, 12, 16] mention the features of a spatial distribution of the requests of a given DVRP instance (describing it

as spatially uniform or clustered), none of those methods directly use the information about the known requests location and volume. An initial non-parametric approach for using the information about the available requests volumes in order to generate artificial requests (to account for the unknown ones) has been presented by the authors [18].

This paper proposes a hyper-heuristic method based on Multi–Environmental Multi–Swarm Optimizer (MEMSO) [11, 12, 13] and 2–Phase Multiswarm Particle Swarm Optimization (2MPSO) [16, 17, 18] algorithms. The hyper-heuristic uses the statistical data about the initially known set of requests in the given Dynamic Vehicle Routing Problem (DVRP).

The rest of the paper is organized as follows. Section 2 defines the DVRP solved in this paper. Section 3 introduces PSO and algorithms MEMSO and 2MPSO (both based on the PSO) used for optimizing DVRP. Section 4 presents the hyper-heuristic approach for solving the DVRP. Section 5 gives experimental setup and results obtained by the method. Finally, Section 6 concludes the paper.

2. Dynamic Vehicle Routing Problem

DVRP is a dynamic version of the generalization of a Traveling Salesman Problem (TSP), called Vehicle Routing Problem (VRP). In the VRP the goal is to optimize a total route for a fleet of vehicles with a limited capacity. VRP has been introduced as a *Truck dispatching problem* in 1959 by Dantzig and Ramser [5]. After the technological advancement of the vehicle tracking devices and development of the Geographical Information Systems a notion of a DVRP has been reintroduced by Psaraftis [21]. The problem has attracted more attention after a set of static VRP benchmarks of Christofides [2], Fisher [8] and Taillard [23] has been customized for the DVRP by Kilby et al. in 1998 [14] and Montemanni et al. in 2005 [15].

In this paper a most common variant of the DVRP is solved [19], sometimes referred to as a VRP with Dynamic Requests (VRPwDR) [12]. In this variant a homogeneous fleet of vehicles (identical capacity $c \in \mathbb{R}$ and speed $sp \in \mathbb{R}$) is considered. There is also an additional constraint, that the vehicle may operate only during a *working day* defined by the opening hours of its depot. During that working day a fleet of m vehicles must serve (visit) a set of n requests. Each request is defined by a location $l_i \in \mathbb{R}^2$, an amount of cargo s_i ($0 \leq s_i \leq c$) to be delivered and an amount of time $u_i \in \mathbb{R}$ it takes to provide a service at that location (unload that cargo). The dynamic nature of that optimization problem comes from the fact that new request may arrive during the working day.

The period of time during which new requests are accepted is limited by a *cut–off time* parameter T_{co}. In all the benchmarks considered in this paper there is one depot for all the vehicles and T_{co} is equal to a half of the working day.

In the mathematical sense, the DVRP is a problem of finding a set of location permutations resulting in a shortest total path length under the time and capacity constrains for each of the permutations in which all the locations are visited exactly once. The exact mathematical formulation of the problem can be found in [7, 17, 18].

3. MEMSO and 2MPSO Algorithms

In this section two most successful approaches to solving DVRP: Khouadjia's et al. MEMSO and authors' 2MPSO are presented. Those two methods are the base algorithms for the hyper–heuristic approach proposed in this paper.

Both, the MEMSO and the 2MPSO, use PSO as their base meta-heuristic and 2-OPT [4] as a route optimization heuristic. In both methods the working day is divided into discrete number of time slices, with the instance of the DVRP problem "frozen" within each time slice. Therefore, each method solves a series of dependent static VRP instances during the optimization process. The difference between PSO applications, encoding of the problem and knowledge transfer in the MEMSO and the 2MPSO methods, together with a brief description of the PSO and 2-OPT algorithms, are presented and discussed in this section.

3.1 Particle Swarm Optimization

PSO algorithm is an iterative population based continuous optimization meta-heuristic approach utilizing the concept of Swarm Intelligence. The algorithm has been introduced by Kennedy and Eberhart in 1995 [10] and has been further developed and studied by other researchers [22, 24, 3].

During the optimization process PSO maintains a set of fitness function solutions (called particles). Each particle has its own location $x \in \mathbb{R}^n$ (an $n-$dimensional fitness function solution proposal), velocity v (a solution change vector), a set of neighbors N (particles which solutions it can observe), a memory of the best observed solution $x_N^{(BEST)}$ and a memory of the best visited solution $x_i^{(BEST)}$.

In each iteration t the location vector x and velocity vector v of ith particle are changed in the following way:

$$x_{i,t} = x_{i,t-1} + v_{i,t-1} \tag{1}$$

$$v_{i,t} = \omega v_{i,t-1} + u_1 c_1 (x_i^{(BEST)} - x_{i,t}) + u_2 c_2 (x_N^{(BEST)} - x_{i,t}) \qquad (2)$$

Where ω denotes an inertia factor, c_1 and c_2 are personal and global attraction factors, u_1 and u_2 follow the uniform $n-$dimensional distribution on $[0,1]^n$.

3.2 2–OPT Algorithm

2–OPT has been introduced as a heuristic algorithm for solving the TSP in 1958 [4]. Its most distinctive feature is the ability to remove the entanglement of routes. The algorithm operates by iterating over all the pairs of edges of a given route and checking the possibility of optimizing the length of route by swapping the ends of those edges. An example of a single step of the algorithm on a sample directed cycle is presented in Fig. 1.

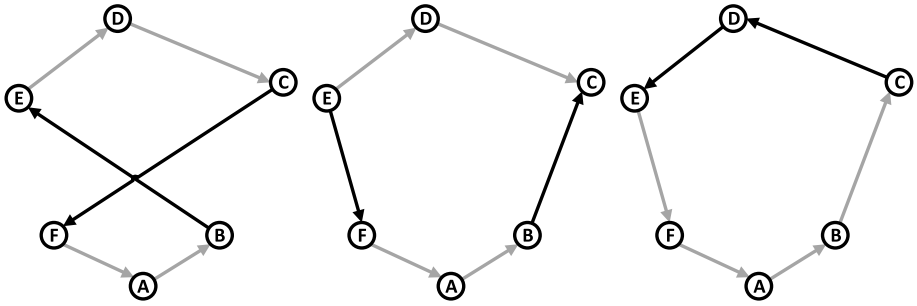


Figure 1: Depiction of a 2–OPT algorithm optimization process over a sample directed cycle. While considering edges BE and CF the algorithm observed that changing them to BC and EF results in a shorter route. After swapping the ends of the considered edges a final step of reversing the direction of the E-D-C path is performed.

2–OPT may be used directly in the VRP variants for optimizing the length of route of a single vehicle.

3.3 Multi–Environmental Multi–Swarm Optimizer

MEMSO [13] algorithm uses PSO to optimize division of the requests among the vehicles. The vehicles' routes within those divided sets are created by a greedy insertion and optimized by the 2-OPT algorithm. The fitness function value is the total length of those routes.

MEMSO uses a discrete encoding of the requests division. The solution is an integer vector representing the requests and the values in the vector are the vehicles' identifiers. Therefore, the PSO algorithm is changed in such a way that a velocity is a vector in $\{1, 2, \ldots, m\}^n$ and

addition in eq. (1) is performed in ${\mathbb{Z}_m}^n$ space (where m is a number of vehicles and n a number of requests).

The knowledge about the current solution is transferred between subsequent time slices by adapting the whole population. For each of the particles the already served and decisively assigned requests are blocked from being changed and new requests are inserted in a greedy way into the solution vectors.

3.4 2-Phase Multi-Swarm Particle Swarm Optimization

2MPSO [17] algorithm uses separate PSO instances to optimize both the division of the requests and their order (in two subsequent optimization phases, hence the name of the method). In the division optimization phase an approach similar to the MEMSO's is used in order to evaluate the total length of routes achieved from the optimized division. The routes are optimized with 2-OPT algorithm from the initially random ordering. In the route optimization phase each of the vehicles is optimized separately and the length of route of the given vehicle is used as a fitness function value.

Instead of customizing the PSO for a discrete problem, 2MPSO follows a continuous optimization approach to applying the PSO algorithm, in contrast with MEMSO. The division of requests among the vehicles is solved as a clustering task, with a number of clusters per vehicle k being the parameter of the method. Therefore, the PSO particle for the first phase optimizer is a sequence of requests clusters centers flattened into a vector in $\mathbb{R}^{2k\hat{m}}$ space (where $\hat{m}$ is the estimated number of vehicles necessary to serve the requests). Particle in the PSO instance optimizing the route of the vehicle is a sequence of requests ranks. Therefore, it is a vector in $\mathbb{R}^{n_i}$ (where n_i is the number of requests assigned to ith vehicle).

The knowledge is transferred between subsequent time slices in a form of cluster centers vector generating a division of requests and a set of requests rank vectors for the routes imposed by requests division. The transferred solution is expanded by a new random cluster center if more vehicles seem to be necessary and the initial ranks of the new requests are also initialized at random.

4. Hyper-Heuristic Approach

Although 2MPSO outperforms MEMSO by 2.22% on average on the Kilby's [14] and Montemanni's [15] sets of benchmarks (see Table 2),

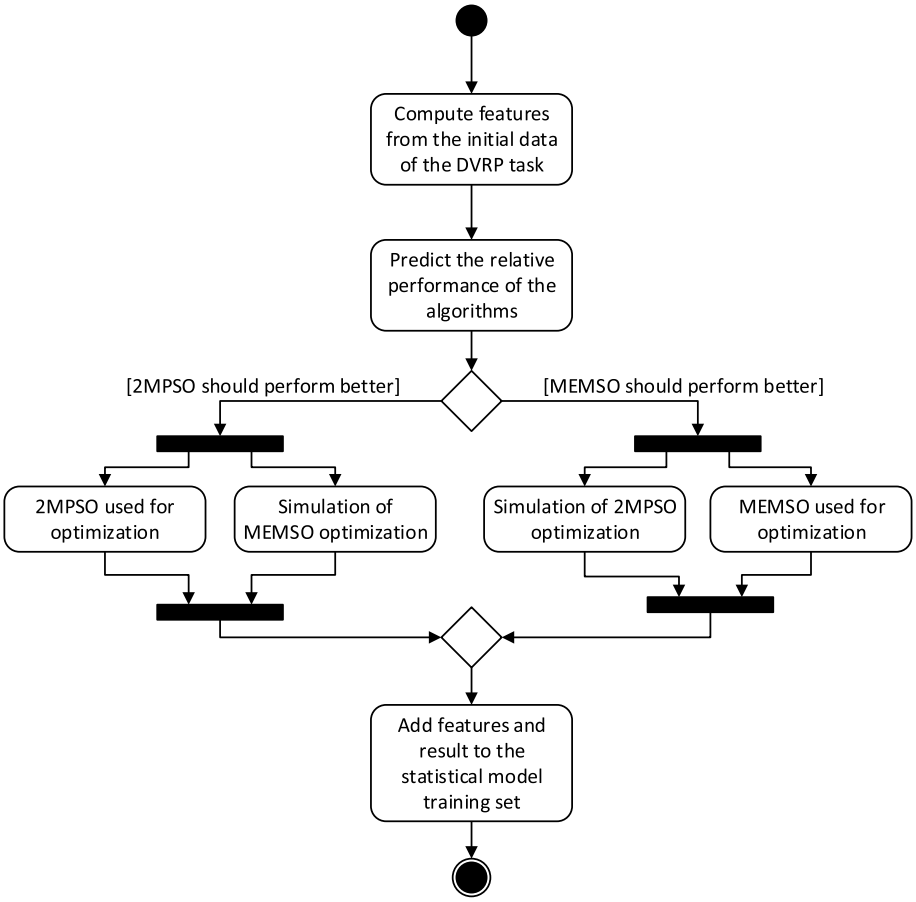


Figure 2: The activity diagram presenting a single run of a 2MPSO and MEMSO based hyper-heuristic.

it might be limited by its clustering approach (or needs an excessively large k) for some particular DVRP tasks.

For that reason, the authors propose a hyper-heuristic approach [1] in which a statistical model might be incrementally trained for choosing the algorithm, which seems to be most suitable for a given DVRP instance. A single run of such hyper-heuristic (solving a single DVRP task) is depicted in Fig. 2. Please observe in the activity diagram, that only the chosen algorithm is used to provide an actual output to some external decision support system. The other algorithm is run only to gather the data and its result is used to tune the prediction model. Please also note, that the choice of the optimization algorithm is done only at the beginning of the optimization process. The prediction model is trained

on the results gathered through a subsequent runs on different DVRP tasks.

The economic cost of running such a hyper-heuristic, in comparison with a single algorithm optimization, would be slightly more than doubled. The doubling comes from the fact, that it is necessary to make a similar amount of computations by two algorithms in order to get comparable results for the performance prediction model. The additional overhead is a result of computations needed for getting statistics from the set of requests and creating a prediction model over the already computed cases. Although the economic cost of proposed approach would definitely be larger than that of a single algorithm, the overhead for the computations necessary for getting solutions during daytime operations would be negligible. The additional operations during the daytime would consist of computing the statistics from the initial state of requests set and providing them to the prediction model. The run of the second algorithm and the training of the prediction model might be done during the nighttime.

This section presents the statistics computed from the benchmark problems and authors' approach to creating a linear model, based on those statistics, predicting the relative MEMSO and 2MPSO algorithms performance.

4.1 Benchmark Characteristics

In order to create a set of features, allowing to discriminate between different benchmark problems, the authors have computed a set of statistics for each of the benchmarks. Those statistics may be divided into 2 groups:

- requests set spatial features,
- requests set volume features.

Within those groups, the following statistics have been computed:

- Spatial features:
  - μ_x, μ_y - mean locations along coordinate system axes,
  - sd_x, sd_y - standard deviation of locations along coordinate system axes,
  - $skew_x$, $skew_y$ - standardized skewness of locations along coordinate system axes,
  - k_{gap} - gap statistic (estimated optimal number of clusters) for requests locations,

Table 1: Values of the input features computed for the initially known sets of requests from Kilby's and Montemanni's benchmarks

Name	μ_x	sd_x	$skew_x$	μ_y	sd_y	$skew_y$	μ_s	sd_s	$skew_s$	nc
c50	0.53	0.32	-0.18	0.51	0.33	0.08	0.09	0.04	0.58	2.00
c75	0.51	0.30	-0.08	0.41	0.29	0.27	0.12	0.06	0.34	4.00
c100	0.48	0.27	0.35	0.46	0.27	0.53	0.08	0.05	0.71	1.00
c100b	0.68	0.17	0.48	0.56	0.26	0.31	0.09	0.05	1.39	0.00
c120	0.38	0.32	0.84	0.67	0.24	0.20	0.06	0.03	1.46	0.67
c150	0.49	0.27	0.06	0.45	0.24	0.28	0.08	0.04	0.78	6.00
c199	0.52	0.26	-0.04	0.47	0.24	0.19	0.09	0.04	0.52	9.00
f71	0.44	0.23	-0.02	0.72	0.18	-0.17	0.04	0.05	1.87	0.00
f134	0.61	0.29	-0.47	0.42	0.22	0.70	0.07	0.11	2.35	1.50
tai75a	0.27	0.18	1.39	0.47	0.20	1.22	0.11	0.16	2.27	3.00
tai75b	0.69	0.25	-1.14	0.42	0.22	-0.69	0.11	0.16	1.24	1.00
tai75c	0.43	0.16	-0.93	0.49	0.25	-0.02	0.11	0.15	2.30	3.00
tai75d	0.43	0.30	0.55	0.49	0.29	-0.23	0.09	0.13	1.55	0.25
tai100a	0.39	0.30	0.61	0.57	0.25	-0.27	0.11	0.18	2.04	1.00
tai100b	0.46	0.29	0.46	0.46	0.22	0.56	0.12	0.16	1.64	1.00
tai100c	0.63	0.27	-0.91	0.50	0.20	-1.10	0.10	0.14	1.34	5.00
tai100d	0.44	0.23	-0.39	0.46	0.21	0.06	0.10	0.17	2.37	2.00
tai150a	0.47	0.29	0.17	0.55	0.31	-0.36	0.09	0.14	2.32	1.67
tai150b	0.61	0.23	0.13	0.52	0.24	0.81	0.09	0.14	2.30	6.00
tai150c	0.47	0.23	0.46	0.61	0.14	-0.06	0.09	0.13	1.80	6.00
tai150d	0.47	0.34	-0.05	0.67	0.23	-0.88	0.12	0.16	1.41	0.14

- Volume features:
  - μ_s - mean volume,
  - sd_s - standard deviation of volume,
  - $skew_s$ - skewness of volume,
  - m_v - minimum number of vehicles necessary to load all the requests.

In order to make the features comparable between different benchmarks, the spatial locations have been mapped to the $[0, 1] \times [0, 1]$ plain, requests volume has been divided by vehicles capacity and k_{gap} has been combined with the m_v in the following way:

$$nc = |1 - \frac{m_v}{k_{gap}}|$$

The larger values of nc suggest that the requests might not be easily divided among the vehicles.

The results of computing those features on the Kilby's and Montemanni's benchmark set for the *a priori* available requests are presented in Table 1.

4.2 Prediction Model

In order to chose a proper optimization algorithm, the authors propose a simple linear regression model.

To train such a model, for predicting the proper optimization algorithm for a given benchmark, the statistics presented in Table 1 has been selected as an input variables and a ratio between the average MEMSO and the average 2MPSO result has been chosen as an output variable. Such approach is justified by the fact that it is more important to choose a proper algorithm when the difference between different algorithms performance is significant.

Fitted linear model is used in a following way. The subset of the statistics to be computed, described in Section 4.1, is identified with the usage of a variable selection method (cf. Fig. 3). When a new DVRP instance is considered, that subset of statistics, forming the input variables for the model, is computed over the requests of that instance. If the model predicts the relative result of MEMSO to 2MPSO to be less than 1, MEMSO is chosen as the optimization algorithm, otherwise 2MPSO is chosen (see Fig. 2).

5. Results

To test the proposed approach a leave-one-out cross-validation experiment with a linear model predicting the relative performance of the MEMSO to 2MPSO has been performed. In each fold a linear model has been built on information from initial sets of requests from 20 out of 21 benchmark problems. The relative algorithm performance has been computed as a ratio of the average results obtained by the MEMSO and 2MPSO algorithms. Such approach simulated a performance of hyper-heuristic optimizing and training the model on some number of subsequently computed DVRP tasks. The average results were computed over 30 runs of MEMSO per benchmark and 20 runs of 2MPSO per benchmark. The order of the total fitness function evaluations budget for each of the algorithms run has been equal to 10^6. The details about parameter setting for MEMSO and 2MPSO experiments can be found in [13] and [17], respectively.

Additionally, in the model training phase some of the input variables have been removed from the model in a step-wise mode, with the usage of Akaike Information Criterion, in order to create a more general predictor. As an example, results of applying such procedure, to the model trained on all 21 of the benchmarks, are presented as a listing of R output in Fig. 3. It can be observed (from the corresponding p-values) that

Table 2: Results of predicting the optimization algorithm from a leave-one-out cross-validation experiment. The table presents the minimum and the average values achieved by MEMSO and 2MPSO algorithms, marking the significantly better average results with a gray background. The Hyper-heuristic column presents which algorithm has been chosen by a linear model, whether the choice has been appropriate and how much has been gained (or lost) with that choice of algorithm. For the benchmark instances with significant difference between average results of MEMSO and 2MPSO the gain has been marked with a gray background and a loss with a light-gray background.

	2MPSO		MEMSO		Hyper-heuristic		
Name	min	avg	min	avg	Chosen	T/F	Gain
c50	583.09	618.59	**577.60**	**592.95**	MEMSO	T	4.15%
c75	**904.83**	**946.85**	928.53	962.54	2MPSO	T	1.63%
c100	**926.10**	**966.27**	949.83	968.92	MEMSO	F	-0.27%
c100b	**830.58**	**875.47**	864.19	878.81	MEMSO	F	-0.38%
c120	**1061.84**	**1176.38**	1164.63	1284.62	2MPSO	T	8.43%
c150	**1132.12**	**1208.60**	1274.33	1327.24	2MPSO	T	8.94%
c199	**1371.61**	**1458.01**	1600.57	1649.17	2MPSO	T	11.59%
f71	302.50	319.01	**283.43**	**294.85**	2MPSO	F	-7.57%
f134	**11944.86**	**12416.65**	14814.10	16083.82	2MPSO	T	22.80%
tai75a	**1721.81**	1846.03	1785.11	**1837.00**	2MPSO	F	-0.49%
tai75b	1418.82	1451.92	**1398.68**	**1425.80**	MEMSO	T	1.80%
tai75c	**1456.90**	1560.68	1490.32	**1532.45**	MEMSO	T	1.81%
tai75d	1445.58	1481.25	**1342.26**	**1448.19**	MEMSO	T	2.23%
tai100a	2211.30	2327.20	**2170.54**	**2213.75**	MEMSO	T	4.87%
tai100b	**2052.54**	**2131.91**	2093.54	2190.01	2MPSO	T	2.65%
tai100c	**1465.06**	**1519.44**	1491.13	1553.55	2MPSO	T	2.20%
tai100d	**1722.16**	**1808.67**	1732.38	1895.42	2MPSO	T	4.58%
tai150a	3367.55	3537.81	**3253.77**	**3369.48**	2MPSO	F	-4.76%
tai150b	2911.22	3033.83	**2865.17**	**2959.15**	2MPSO	F	-2.46%
tai150c	2510.51	**2579.72**	**2510.13**	2644.69	2MPSO	T	2.46%
tai150d	2893.54	**2992.53**	**2872.80**	3006.88	MEMSO	F	-0.48%

both the spatial and the volume related features have been selected as informative ones.

The results from the cross-validation experiment are given in Table 2. The correct algorithm (the one with a better average result) has been chosen in 14 out of 21 cases (all of them with statistically significant difference between average algorithms results, verified by a t-test). Wrong predictions, that have been made for the other 7 cases, have resulted in a significant loss of results quality only for 3 of them (`f71`, `tai150a`, and `tai150b`). Therefore, the linear model achieved 82% accuracy for the subset of 17 benchmarks with significant differences in the algorithms average results (with 67% accuracy over the whole set of benchmarks).

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.42899    0.15676    9.116 9.64e-07 ***
sd_x         1.23119    0.26551    4.637 0.000573 ***
mu_y        -1.11031    0.16881   -6.577 2.62e-05 ***
sd_y        -1.56756    0.28836   -5.436 0.000151 ***
skew_y      -0.04576    0.02481   -1.844 0.089977 .
mu_s         2.72795    1.25435    2.175 0.050362 .
sd_s        -3.39358    0.71918   -4.719 0.000498 ***
skew_s       0.21507    0.04592    4.683 0.000529 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 .
Residual standard error:
                0.03915 on 12 degrees of freedom
Multiple R-squared: 0.86, Adjusted R-squared: 0.7764
F-statistic: 10.43 on 7 and 12 DF, p-value: 0.000286
```

Figure 3: Model trained on the full data set, with the variable selection using Akaike Information Criterion, presented as an R output.

6. Conclusion

Choosing an optimization algorithm for the DVRP on the basis of the characteristics of initial requests sets leads to the improvement of the results. Choosing between MEMSO and 2MPSO algorithms resulted in a 0.6% improvement in comparison with the 2MPSO average performance (with a maximum of 1.5% improvement if all the predictions were accurate) and 2.8% in comparison with the MEMSO performance. The best performance has been improved by 0.2% on average in comparison with 2MPSO and by 2.8% in comparison with MEMSO.

The obtained results suggest that the proper benchmark features has been selected and choosing an optimization algorithm for a given benchmark problem is possible and may lead to results improvement. Both the spatial and the volume related features have been marked as significant in predicting relative performance.

The future work should include an algorithm choice possibility during the optimization and not only at the beginning of the working day. It might be also beneficial to search for another set of characteristics allowing for a proper choice of the algorithm, in order to try eliminating the DVRP tasks for which the significantly worse algorithm has been selected.

Acknowledgment: The research was financed by the National Science Centre in Poland grant number DEC-2012/07/B/ST6/01527.
Project website: `http://www.mini.pw.edu.pl/~mandziuk/dynamic`.

References

[1] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*. International Series in Operations Research & Management Science 57, pages 457–474. Springer, 2003.

[2] N. Christofides and J. E. Beasley. The period routing problem. *Networks*, 14(2):237–256, 1984.

[3] C. W. Cleghorn and A. P. Engelbrecht. Particle swarm variants: standardized convergence analysis. *Swarm Intelligence*, 9(2-3):177–203, 2015.

[4] G. A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6:791–812, 1958.

[5] G. B. Dantzig and R. H. Ramser. The Truck Dispatching Problem. *Management Science*, 6:80–91, 1959.

[6] M. J. Elhassania, B. Jaouad, and E. A. Ahmed. A new hybrid algorithm to solve the vehicle routing problem in the dynamic environment. *International Journal of Soft Computing*, 8(5):327–334, 2013.

[7] M. J. Elhassania, B. Jaouad, and E. A. Ahmed. Solving the dynamic Vehicle Routing Problem using genetic algorithms. *Proceedings of the International Conference on Logistics and Operations Management (GOL)*, pages 62–69, 2014.

[8] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[9] F. T. Hanshar and B. M. Ombuki-Berman. Dynamic vehicle routing using genetic algorithms. *Applied Intelligence*, 27(1):89–99, 2007.

[10] J. Kennedy and R. Eberhart. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks*, vol. IV, pages 1942–1948, 1995.

[11] M. R. Khouadjia, E. Alba, L. Jourdan, and E.-G. Talbi. Multi-Swarm Optimization for Dynamic Combinatorial Problems: A Case Study on Dynamic Vehicle Routing Problem. *Lecture Notes in Computer Science*, 6234:227–238, 2010.

[12] M. R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E.-G. Talbi. A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests. *Applied Soft Computing*, 12(4):1426–1439, 2012.

[13] M. R. Khouadjia, E.-G. Talbi, L. Jourdan, B. Sarasola, and E. Alba. Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems. *Journal of Supercomputing*, 63(3):836–853, 2013.

[14] P. Kilby, P. Prosser, and P. Shaw. Dynamic VRPs: A Study of Scenarios. 1998 `http://www.cs.strath.ac.uk/~apes/apereports.html`.

[15] R. Montemanni, L. Gambardella, A. Rizzoli, and A. Donati. A new algorithm for a dynamic vehicle routing problem based on ant colony system. *Journal of Combinatorial Optimization*, 10:327–343, 2005.

[16] M. Okulewicz and J. Mańdziuk. Application of Particle Swarm Optimization Algorithm to Dynamic Vehicle Routing Problem. *Lecture Notes in Computer Science*, 7895:547–558, 2013.

[17] M. Okulewicz and J. Mańdziuk. Two-Phase Multi-Swarm PSO and the Dynamic Vehicle Routing Problem. *Proceedings of the 2nd IEEE Symposium on Computational Intelligence for Human-like Intelligence*, pages 86–93, 2014.

[18] M. Okulewicz and J. Mańdziuk. Dynamic Vehicle Routing Problem: A Monte Carlo approach. In *Information Technologies: Research and Their Interdisciplinary Applications*, pages 119–138, Pułtusk, Poland, 2015. `http://phd.ipipan.waw.pl/pliki/mat_konferencyjne/12_ITRIA_2015_01.pdf#page=120`

[19] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[20] H. N. Psaraftis. Dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[21] H. N. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61(1):143–164, 1995.

[22] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. *Proceedings of IEEE International Conference on Evolutionary Computation (CEC)*, pages 69–73, 1998.

[23] É. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[24] I. C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003.

EXTREMAL OPTIMIZATION AND NETWORK COMMUNITY STRUCTURE

Noémi Gaskó
Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
gaskonomi@cs.ubbcluj.ro

Rodica Ioana Lung
Department of Statistics, Forecasts, Mathematics
Babeş-Bolyai University, Cluj-Napoca, Romania
rodica.lung@econ.ubbcluj.ro

Mihai Alexandru Suciu
Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
mihai-suciu@cs.ubbcluj.ro

Abstract The network community structure detection problem has been recently approached with several variants of an extremal optimization algorithm. An extremal optimization algorithm is a stochastic local search method that evolves pairs of individuals that can be represented as having several components by randomly replacing components having worst fitnesses. The number of components to be replaced in one iteration influences both the exploitation and exploration capabilities of the method; an efficient method of adjusting this number during the search may significantly influence the quality of results. In this paper we explore the use of several updating mechanisms for this number. Numerical experiments are used evaluate them and also to compare results obtained with those provided by other state-of-art methods.

Keywords: Community structure detection, Extremal optimization.

1. Introduction

The network community structure detection problem has recently attracted a lot of attention from the heuristic community because both its large applicability and challenging nature. A particular challenge associ-

ated with this problem arises from the lack of a formal definition for the concept of community, and subsequently that of community structure [6]. Apart from classical definitions that attempt to characterize communities by using various network measures, a relatively recent class of approaches define the community structure as the optimum value of a certain fitness function that is supposed to illustrate the modularity of the network. Alas, it is also accepted that such an ideal function has not yet been proposed; existing attempts can only be validated by means of numerical experiments and, while some functions prove suitable for synthetic benchmarks, most of them fail when tested on real-world networks for which the community structure is not so well-defined. Moreover, the effectiveness of using a certain fitness function depends also on the underlying method used to compute the optimum value.

In this paper we investigate the behavior of an extremal optimization algorithm designed to optimize the modularity function [13], combined with the community fitness [8], when using four different manners of updating the number of nodes to be changed in one iteration: the classic variant of changing only one node [3, 4], the improved τEO [2], and the more recent variants in which this number decreases exponentially [18], or linearly [11].

2. Network Community Structure

The fact that the community structure detection problem can be reformulated as an optimization problem, makes it approachable by stochastic search methods, benefiting from their scalability and adaptability. Given an undirected, unweighted graph $G = (N, E)$, where N is the set of nodes, or vertices, and E is the set of edges/links, a community structure is described intuitively as a partition over the set of nodes such that nodes within each set are more connected to each other than to the other sets in the partition.

While this intuitive definition appears to be easily formalized, by considering either that a community is a group of nodes such that for each one the number of links within the community is greater than the number of links connecting it to the outside (the strong community concept [15]) or even that the total number of links connecting nodes inside the community is greater than the number of links to the outside (the weak community concept [15]), there are many counterexamples of networks with known community structure that do not satisfy either definition. In fact, there does not exist a definition that formalizes the intuitive description above and be accepted as valid for most situations.

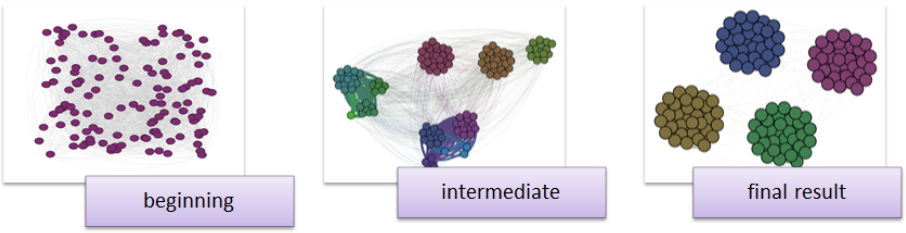


Figure 1: An example of a solution detected by an heuristic on a network with 124 nodes and 4 communities.

In spite of this, or maybe because of it, alternate methods to define the community structure have been proposed. One of the most popular one, from a computational point of view, is to use a function that has as an optimum value the real community structure of the network. Again, while such a function that may be used for all possible networks does not exist, there are some that are more effective and that became popular in approaching this problem in the last years. Examples are the modularity Q [13], the modularity density [10], the community score [14], and the community fitness [8]. These functions, and most of all the modularity and the modularity density, have been widely used and studied in conjunction with various heuristics designed for their direct or indirect optimization, i.e., directly finding their optimum and consider it as the community structure, or only including them in one or more search phases. An example of what is expected from a community structure detection algorithm is depicted in Fig. 1.

The modularity Q of a community structure is defined as

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j), \tag{1}$$

where the sum runs over all pairs of vertices i and j, A is the adjacency matrix, m the total number of links in the network, k_i the degree of node i, C_i the community of node i and $\delta(C_i, C_j)$ equals 1 if nodes i and j belong to the same community and 0 otherwise. When two community structures are compared, a higher modularity value presumably indicates a better solution.

3. Extremal Optimization

Recently, a new heuristic approach that combines the modularity and community fitness and uses extremal optimization algorithm (EO) [2]

as an underlying method has been proposed. Validated by means of numerical experiments, this approach has proven more efficient than other state-of-art methods when tested on usual benchmarks. With the purpose of improving the extremal optimization method, several other EO variants have been proposed, each one of them apparently leading to better results. This article compares these methods, in the context of the community structure detection problem, in the attempt to assess if there are significant differences among them and if so, if one of them may prove more efficient than the others.

The baseline method used here is the NoisyEO algorithm [11] which is described in Algorithms 1 and 2. A typical EO algorithm evolves a pair of individuals (s, s_{best}): s explores the search space and s_{best} preserves the best solution found by s. NoisyEO evolves a population of such pairs of individuals representing possible structures and evaluated with the modularity function. Also typical to EO is the fact that one individual is represented as a set of components with different fitnesses; during one iteration the component having the worst fitness value is randomly replaced. A more efficient variant, called $\tau-$EO, uses a probability distribution to decide which nodes are changed [2].

NoisyEO considers nodes as the components, and computes for each node a fitness function as the node's contribution to its community, i.e.:

$$f_i^{(node)}(C_1, \ldots, C_n) = f(C_i) - f(C_i \backslash \{i\}), \tag{2}$$

where C_i represents the community of player i, $s \in S$, and $C_i \backslash \{i\}$ is the same community without node i; f is the community score:

$$f(C) = \frac{k_{in}(C)}{(k_{in}(C) + k_{out}(C))^{\alpha}}, \tag{3}$$

$k_{in}(C)$ is the double of the number of internal links in community C; $k_{out}(C)$ the number of external links of C; and α - a parameter that controls the community size (in experiments presented in this paper $\alpha = 1$). Thus, a NoisyEO individual is a vector of length equal to the number of nodes, of the form $s = (C_1, \ldots C_n)$, where C_i is the community of node i.

Within NoisyEO several components are replaced simultaneously: their number starts from approximatively 10% of the number of nodes, linearly decreases until the middle of the search and after that it remains constant, equal to 1. But there are also other several ways this number can be changed, based on the intuition that larger values induce diversity and intensifies exploration of the space, while smaller values permit better exploitation. In this paper we explore the possibility of using

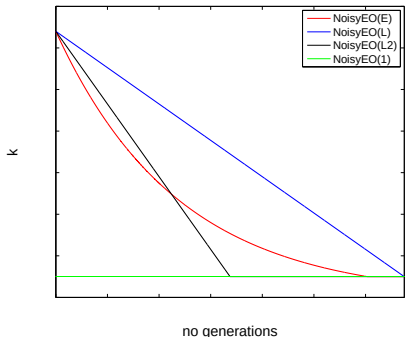


Figure 2: Different variants of updating the number of nodes κ that are changed each iteration during the search in order to maintain the equilibrium between exploration - at the beginning of the search - and exploitation - towards the end of the search.

other methods of adapting this number in an attempt to find the best version of EO suitable for the community structure detection problem. The following variants, all based on NoisyEO, are proposed:

- NoisyEO(L2) - κ decreases linearly to 1 until the middle of the search and remains 1 to the end [11];
- NoisyEO(L) - κ decreases linearly to 1;
- NoisyEO(E) - κ decreases exponentially from approx 10% of the number of nodes, to 1 at the end of the search:

$$\kappa_{NrGgen} = \max\left\{1, \left[\frac{1}{10} \cdot N \cdot (N-2)^{-\frac{NrGen}{MaxGen}}\right]\right\}, \qquad (4)$$

  where $[\cdot]$ represents the integer part, N the number of network nodes, and $MaxGen$ the maximum number of generations/iterations allowed.

- NoisyEO(1) $\kappa = 1$, constant;
- $\tau-$EO, that uses the framework of NoisyEO with a $\tau-$EO iteration, in which nodes are ranked by fitness and the probability of choosing node r is $P(r) \propto r^{-\tau}$ [5].

A graphical representation of the four different variants of setting κ is illustrated in Fig. 2.

Algorithm 1 *NoisyEO* algorithm

Parameters:

- Population size - $popsize$;
- Probability of shift - p_{shift};
- Number of generations between switching networks - G;
- Total number of shifts - $NrShifts$;
- Expected minimum and maximum number of communities.

1: Randomly initialize $popsize$ pairs of configurations (s, s_{best}).
2: $noise$=false;
3: **repeat**
4: **if** $noise$ **then**
5: Induce noise with probability $p^{(*)}_{shift}$;
6: Randomly reinitialize each s_{best} in population;
7: **else**
8: perform search on the original network;
9: **end if**
10: $noise$=not $noise$;
11: Update k depending on the tested EO variant$^{(**)}$;
12: **for** G generations **do**
13: Apply κEO (s, s_{best}) for all pairs (s, s_{best}) - Alg. 2;
14: **end for**
15: **until** $G * NrShifts >$ Maximum number of generations;
16: Return s_{best} with highest ***fitness***.

$^{(*)}$ Modify network by randomly deleting/adding nodes with probability p_{shift} which decreases linearly from an initial value to 0 during the search.
$^{(**)}$ One of the following variants are considered:

- NoisyEO(E) - κ decreases exponentially, eq. (4);
- NoisyEO(L) - κ decreases linearly to 1;
- NoisyEO(L2) - κ decreases linearly to 1 until the middle of the search and remains 1 to the end;
- NoisyEO(1) $\kappa = 1$, constant;

Algorithm 2 κEO(s, s_{best}) iteration

1: For current configuration s evaluate $u_i(s)$, the fitness function corresponding of node $i \in \{1, \ldots, n\}$.
2: find the κ worst components and replace them with a random value;
3: **if** (s is ***better***$^{(***)}$ than s_{best}) **then**
4: set $s_{best} := s$.
5: **end if**

$^{(***)}$ better modularity value (1)

4. Numerical Experiments

Numerical experiments, performed on several benchmarks, are used to compare the results offered by the five NoisyEO variants with those offered by other state-of-art algorithms.

Experimental set-up. Numerical experiments were performed on synthetic benchmarks and real-world networks with the five variants of *NoisyEO*: NoisyEO(L2), NoisyEO(L),NoisyEO(E), NoisyEO(1), $\tau-$EO. Results were compared with four state-of-art methods: *Louvain* [1], *OSLOM* [9], *Infomap* [16], and *ModOpt* [17] - run by using the source code from `sites.google.com/site/andrealancichinetti/software`, last accessed May, 2015. *Louvain* and *ModOpt* optimize the modularity, *OSLOM* uses a probability of a node to belong to a community and *Infomap* is based on a random walk.

Parameter settings. The algorithm parameters are the same for all variants of *NoisyEO*: population size 50, initial value of $p_{shift} = 1$, $G = 45$, $NrShifts = 150$; the interval for the number of communities for each network is estimated such that the correct number is included and assigned to approx. 25% of the population.

Benchmarks. Four sets of synthetic networks were generated:

- GN: 128 nodes, 4 equal sized communities, node degree 16, z_{out} indicates the number of links a node has outside its community; 30 networks for each $z_{out} \in \{1, \ldots 8\}$;
- LFR 128 nodes: average vertex degree 20, maximum vertex degree 50, community size $[10, 50]$;
- LFR 1000 nodes, S - small: average vertex degree 20, maximum vertex degree 50, community size $[10, 50]$;
- LFR 1000 nodes, B - big: average vertex degree 20, maximum vertex degree 50, community size $[20, 100]$

The LFR sets are characterized by the mixing parameter μ value - computed as the ratio between the number of links a node has outside its community and its degree. For each set and each μ value, we generated 30 networks. The most challenging sets are those where $\mu \in \{0.5, 0.6\}$ and $z_{out} = 8$, because they have a less well-defined community structure. Even among these networks (128 nodes, 1000 nodes small and big), the most difficult ones are the small ones (128 nodes), because if we increase

the network size and the number of communities, a better-defined structure is created.

The real-world networks used for experiments are: the bottle-nose *dolphin* network [12], the *football* network [7], the Zachary *karate* club network [19], and the *books* about US politics network – `www.org\net.com`, last accessed 9/3/2015.

Performance evaluation. We use the normalized mutual information (NMI) proposed in [8] to evaluate and to compare results. A NMI value of 1 indicates, that two communities are identical. We compared obtained results for each method to the real community structure of the network.

For each benchmark set, results are further compared by using the Wilcoxon sign-rank nonparametric test (for 30 independent runs for each real network and on the 30 networks for each GN and LFR sets). The Wilcoxon sign rank specifies if the difference between two sample medians may be considered significant: the null hypothesis that two samples come from the same population can be rejected with a level of significance $\alpha = 0.05$ if the computed p-value is smaller than 0.05.

Results and discussion. Results are presented in the form of box-plots of NMI values obtained for the 30 runs by each method, in Figures 3–5. Next to each box-plot, a black-white matrix corresponding to Wilcoxon h values indicates the results of the pairwise comparisons of the tested methods: a black square shows a statistical difference between the two methods.

Regarding the difference between NoisyEO variants, the Wilcoxon h matrices show very few differences between the three adaptive variants: E, L, and L2. The exponential variant, E, shows worst results for the GN $z_{out} = 8$ set, but this result is still better than all the results obtained by the other methods. Setting $k = 1$ and $\tau-$EO do not yield good results compared to the other EO variants for the synthetic benchmarks. For the real-world networks, results are much closer, but still the three variants outperform the others.

5. Conclusion

A comparative analysis of four variants of extremal optimization updating procedures for the community structure detection problem is presented. The results show that the use of an adaptive method of setting the number of nodes to be randomly reassigned each iteration is beneficial; however, differences between tested variants are not significant enough to enable us to draw a conclusion regarding the best variant for

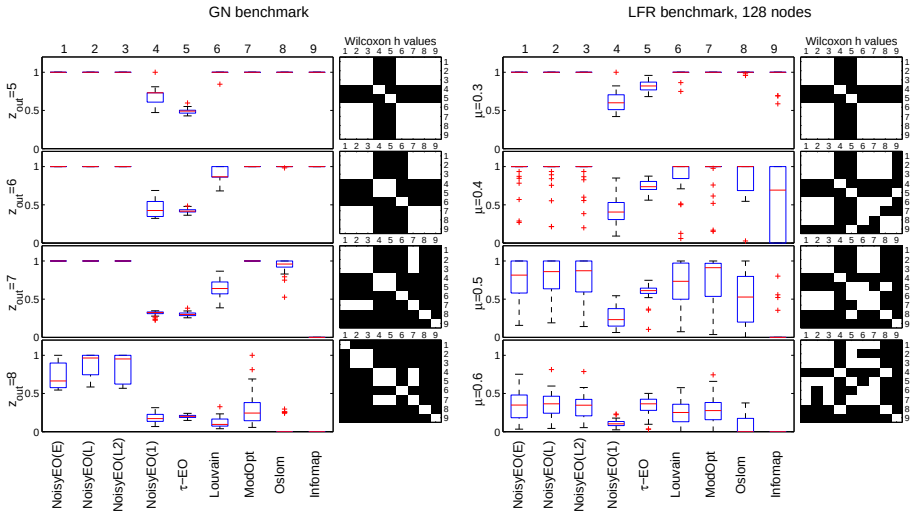


Figure 3: GN and LFR benchmarks, 128 nodes. Comparisons with other methods. Boxplots of NMI values obtained for the 30 networks in each set by each considered method. Wilcoxon h values matrices illustrate the statistical significance of the differences in results for the nine methods: a black box corresponds to $p < 0.05$ and rejection of the null hypothesis that the two samples have the same median.

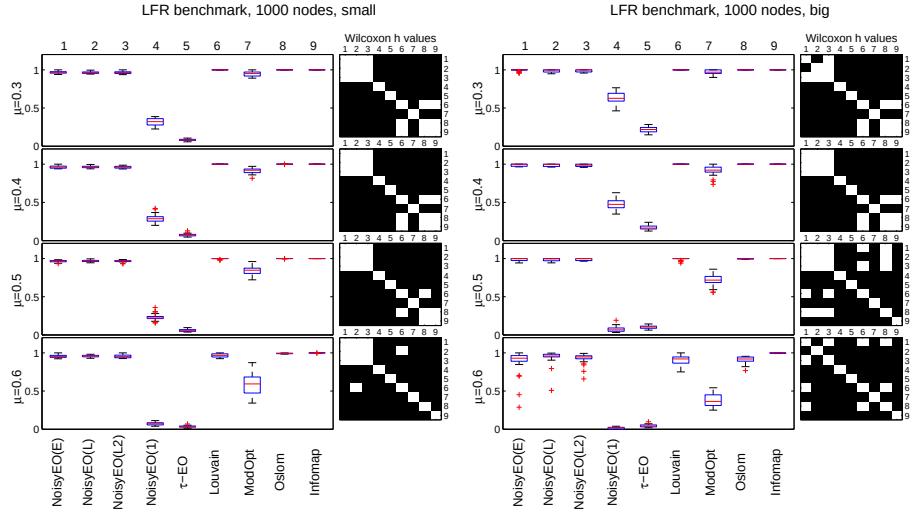


Figure 4: LFR benchmark, 1000 nodes, Small and Big. Results are represented in the same manner as in Fig. 3

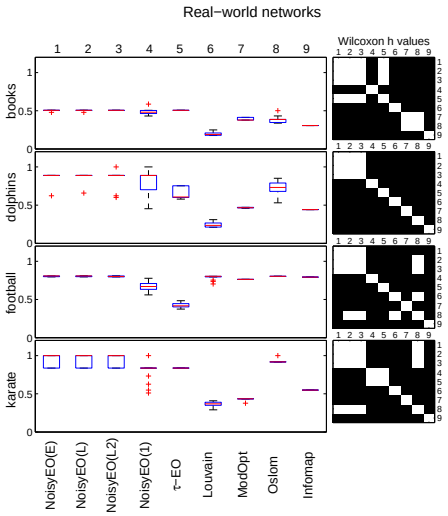


Figure 5: Real-world networks. Results are represented in the same manner as in Fig. 3.

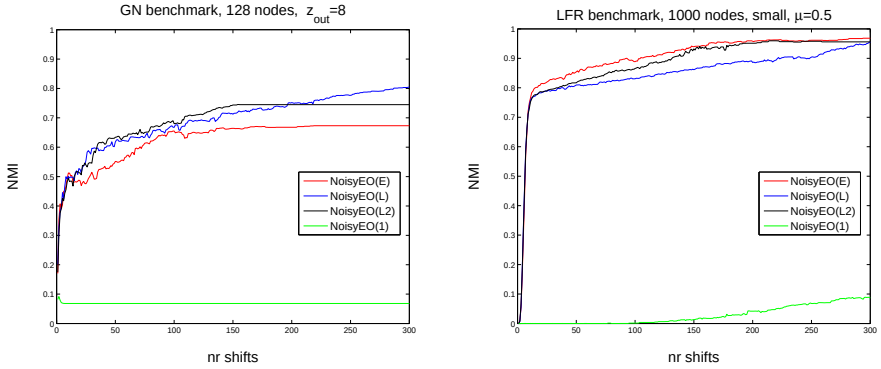


Figure 6: Evolution of NMI values in time for two of the most difficult sets: GN $z_{out} = 8$ and LFR 1000 nodes with $\mu = 0.5$. It is obvious that adapting the values of k leads to better results. The differences in means represented here have no statistical significance for the LFR benchmark.

the tested problems. Only when using an exponential rule, results are worse than the other EO variants, but even in those situations, they are very good.

Numerical results also show that extremal optimization may be very powerful in addressing the problem of community structure detection. Its main drawback, however, arises from the fact that random the computational time required by the iterative random changes makes this approach less efficient for large networks. On the other hand, this method proved very efficient for small networks with less visible community structures. Further work consists in finding the means to improve its scalability while maintaining its efficiency in dealing with ambiguous community structures.

Acknowledgment: This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS - UEFISCDI, project number PN-II-RU-TE-2014-4-2332.

References

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[2] S. Boettcher and A. Percus. Nature's way of optimizing. *Artificial Intelligence*, 119:275–286, 2000.

[3] S. Boettcher and A. G. Percus. Optimization with Extremal Dynamics. *Physical Review Letters*, 86:5211–5214, 2001.

[4] S. Boettcher and A. G. Percus. Extremal optimization: an evolutionary local-search algorithm. In *Computational Modeling and Problem Solving in the Networked World*. Operations Research/Computer Science Interfaces Series 21, pages 61–77, Kluwer Academic Publishers, 2002.

[5] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72:027104, Aug 2005.

[6] S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

[7] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[8] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.

[9] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato. Finding statistically significant communities in networks. *PloS One*, 6(4):e18961, 2011.

[10] Z. Li, S. Zhang, R.-S. Wang, X.-S. Zhang, and L. Chen. Quantitative function for community detection. *Physical Review E*, 77:036109, 2008.

[11] R. I. Lung, M. Suciu, and N. Gaskó. Noisy extremal optimization. *Soft Computing*, pages 1–18, 2015.

[12] D. Lusseau, K. Schneider, O. Boisseau, P. Haase, E. Slooten, and S. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[13] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004.

[14] C. Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. *Lecture Notes in Computer Science*, 5199:1081–1090, 2008.

[15] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, 2004.

[16] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[17] M. Sales-Pardo, R. Guimerà, A. A. Moreira, and L. A. N. Amaral. Extracting the hierarchical organization of complex systems. *Proceedings of the National Academy of Sciences*, 104(39):15224–15229, Sept. 2007.

[18] M. Suciu, R. I. Lung, and N. Gaskó. Mixing Network Extremal Optimization for Community Structure Detection. *Lecture Notes in Computer Science*, 9026:126–137, 2015.

[19] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.

THE PITFALLS OF OVERFITTING IN OPTIMIZATION OF A MANUFACTURING QUALITY CONTROL PROCEDURE

Tea Tušar
DOLPHIN Team, INRIA Lille – Nord Europe, France
Department of Intelligent Systems, Jožef Stefan Institute, Ljubljana, Slovenia
tea.tusar@ijs.si

Klemen Gantar
Faculty of Computer and Information Science, University of Ljubljana, Slovenia
kg6983@student.uni-lj.si

Bogdan Filipič
Department of Intelligent Systems, Jožef Stefan Institute, Ljubljana, Slovenia
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
bogdan.filipic@ijs.si

Abstract We are concerned with the estimation of copper-graphite joints quality in commutator manufacturing—a classification problem in which we wish to detect whether the joints are soldered well or have any of the four known defects. This quality control procedure can be automated by means of an on-line classifier that can assess the quality of commutators as they are being manufactured. A classifier suitable for this task can be constructed by combining computer vision, machine learning and evolutionary optimization techniques. While previous work has shown the validity of this approach, this paper demonstrates that the search for an accurate classifier can lead to overfitting despite cross-validation being used for assessing the classifier performance. We inspect several aspects of this phenomenon and propose to use repeated cross-validation in order to amend it.

Keywords: Computer vision, Differential evolution, Machine learning, parameter tuning, Manufacturing, Quality control.

1. Introduction

In automotive industry, only one part per million of supplied products is allowed be defective, which yields strict requirements for the involved manufacturing processes as well as their quality control procedures. We are interested in the manufacturing of graphite commutators (i.e., components of electric motors used, for example, in automotive fuel pumps) produced at an industrial production plant. More specifically, we wish to automatically assess the quality of copper-graphite joints in commutators after the soldering phase of this manufacturing process, which is one of the most critical phases of commutator production.

At present, the soldering quality control at the plant is done manually. Automated on-line quality control would bring several advantages over manual inspection. For example, it can promptly detect irregularities making error resolution faster and consequently saving a considerable amount of resources. Moreover, it does not slow down the production line and is cheaper than manual inspection. Finally, it does not suffer from fatigue and other human factors that can result in errors. This is why we aim for an automated on-line quality control procedure capable of determining whether the joints are soldered well or have any of the four known defects.

Such automation can be implemented on the production line with a classifier previously constructed on a database of commutator segment images with known defects (or absence of defects). Three previous studies [3, 4, 5] have already tackled this problem and in all cases 10-fold cross-validation (CV) was used as a measure of classifier accuracy. This work questions 10-fold CV as the measure of choice for such tasks and proposes actions to deal with the inevitable overfitting issue.

The rest of the paper is structured as follows. Section 2 presents details of the problem in question, summarizes previous work and outlines the design of the quality control procedure used in this study. Section 3 is devoted to cross-validation and the overfitting issue. Performed experiments and their results are discussed in Section 4. Finally, Section 5 summarizes the paper and gives ideas for future work.

2. Background

2.1 Soldering in Commutator Manufacturing

The soldering phase in the commutator manufacturing process consists of soldering the metalized graphite to the commutator copper base. The quality of the resulting copper-graphite joints is crucial since the reliability of end user applications directly depends on the strength of

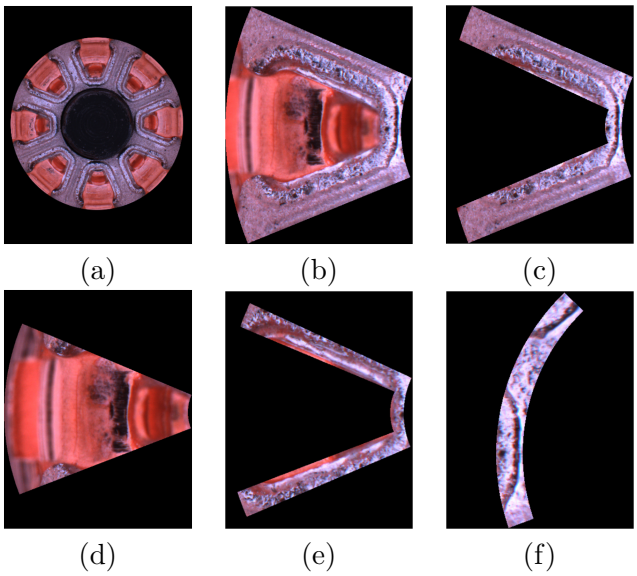


Figure 1: Images of: (a) a graphite commutator, (b) a commutator segment, (c) a ROI for metalization defect, (d) a ROI for excess of solder, (e) a ROI for deficit of solder, and (f) a ROI for disorientation.

these joints. After the soldering phase, the commutators are manually inspected for presence of any defects. Known defects comprise *metalization defect* (presence of visible defects on the metalization layer), *excess of solder* (presence of solder spots on the copper pad), *deficit of solder* (lack of solder in the graphite-copper joint) and *disorientation* (disorientation between the copper body and the graphite disc). Commutators are made up of a number of segments, depending on the model (the considered commutator model from Fig. 1 (a) consists of eight segments). If a single segment has any of the listed defects, the whole commutator is labeled as defective and removed from the production process.

Various defects occur in different regions of the commutator segment. For example, the region where the excess of solder is usually detected is different from the region where disorientation can be observed. Therefore, images of commutator segments can be divided into four regions of interest (ROIs), one for each defect (see Fig. 1).

Because five different outcomes are possible (rare cases where two or more defects appear on a single commutator segment are labeled with just one defect and are not differentiated further), we treat this as a classification problem with five classes. While the manufacturers are indeed interested in keeping statistics of the detected defects, their main concern is that no *false positives* are found. This means that cases when

a defective commutator is labeled as without defects are to be avoided as much as possible. This is, of course, very hard to achieve.

2.2 Previous Work

Three previous studies investigated different aspects of this challenging real-world problem. The initial experiment [4] explored whether computer vision, machine learning and evolutionary optimization techniques could be employed to find small and accurate classifiers for this problem. First, images of the copper-graphite joints were captured by a camera. Next, a fixed set of features were extracted from these images using digital image processing methods. This data were then used to train decision trees capable of predicting if a commutator segment has any of the four defects or is soldered well. The DEMO (Differential Evolution for Multiobjective Optimization) algorithm [12] was applied to search for small and accurate trees by navigating through the space of parameter values of the decision tree learning program. The study found this setup to be beneficial, but urged to focus future research on more sophisticated extraction of features from the images as this seemed to hinder the search for more accurate classifiers.

The second study [5] presented a different setup for the automated quality control procedure to address the issues from the first study. Instead of optimizing decision tree parameter values, differential evolution (DE) [13] was used to search for the best settings of image processing parameters such as filter thresholds. Tuning of these parameters can be a tedious task prone to bias from the engineers that usually do it by trial-and-error experimentation. Moreover, the choice of right features is crucial for obtaining a good classifier.

The single classification problem with five classes was split into four binary classification subproblems, where each subproblem was dedicated to detecting one of the four defects and used data only from the corresponding ROI. In addition, instead of classification accuracy, the measure to be optimized was set to a function penalizing the portion of false negatives 100 times harder than the portion of false positives. The study found that the new combination of computer vision, machine learning and evolutionary optimization techniques was powerful and achieved some good results. While optimization with DE always found better parameter settings for image processing methods than those defined by domain experts, some subproblems proved to be harder than others. For example, detection of commutator segments with excess of solder achieved a satisfactory accuracy, while the detection of metalization defects did not.

The third study [3] investigated the correctness of the implicit assumption from [5] that only features of the subproblem-specific ROI would influence the outcome of the classifier for that subproblem. The study found that features from other ROIs can be important as well, suggesting that it might be better not to split the classification problem into subproblems at all.

While being otherwise rather different, all three mentioned studies used 10-fold CV to estimate the performance of the employed classifiers. In this paper we wish to test if such evaluation of classifiers is appropriate when performing optimization based on this measure.

2.3 Design of the Automated Quality Control Procedure

The automated quality control procedure considered in this paper is very similar to the one presented in [5]. Again computer vision, machine learning and evolutionary optimization methods are combined in the search for the best settings for image processing parameters. In short, the procedure design consists of the following steps:

1 Determine a set of image features.

2 Use an evolutionary algorithm to search for the values of image processing parameters that result in the highest fitness. Evaluate each solution using these steps:

   (a) Based on the chosen parameter values, use the image processing methods to convert each image of a commutator segment into a vector of feature values.

   (b) Construct a classifier (in our case a decision tree) where the vectors of feature values serve as learning instances. Estimate the classifier accuracy and use this value as solution fitness.

3 Choose the best found classifier and the corresponding image processing parameter values to detect defects in images of new commutator segments as they are being manufactured.

Let us now describe the steps of processing commutator segment images, building decision trees and optimizing classifier performance in more detail.

2.3.1 Processing commutator segment images. Processing of images is the most time-consuming task of our procedure and is done in several steps. First, the image of a commutator segment needs to

be properly aligned. Next, the four ROIs shown in Fig. 1 need to be detected. This is done by applying four predefined binary masks to the image, one for each ROI. Each of the ROIs is further processed as follows. Depending on the ROI, the image in RGB format is converted into a gray-scale image by extracting a single color plane. Based on expert knowledge, red is used for all ROIs except the ROI for excess of solder, which uses the blue color plane.

The final three steps require certain parameters to be set. A 2D median filter of size 1×1, 3×3 or 5×5 is applied to reduce noise. Next, a binary threshold that can take values from $\{1, 2, \ldots, 256\}$ is used to eliminate irrelevant pixels. Finally, an additional particle filter is employed to remove all particles (connected pixels with similar properties) with a smaller number of pixels than a threshold value from $\{1, 2, \ldots, 1000\}$. Note that because of the diversity of the defects, it is reasonable to assume that these three image processing parameters should be set independently for each ROI. This means that in total 12 image processing parameters need to be set.

After these image processing steps, the chosen set of features are extracted from the image of each ROI. We use the same set of features as in [5, 3]:

- number of particles,
- cumulative size of particles in pixels,
- maximal size of particles in pixels,
- minimal size of particles in pixels,
- gross/net ratio of the largest particle,
- gross/net ratio of the cumulative size of particles.

To summarize, computer vision methods are used to convert each commutator segment image into a vector of 24 feature values.

2.3.2 Building decision trees. Commutator segment images with known classes are used to construct a database of instances, upon which a machine learning classifier can be built. We chose decision trees since they are easy to understand and implement in the on-line quality control procedure. In accordance with the guidelines from [3], we do not split the machine learning problems into subproblems, but use all instances and all ROIs to build a single classifier with five classes: no defect, metalization defect, excess of solder, deficit of solder and disorientation.

Note that the classifier predicts defects on commutator segments. For the final application, predictions for all segments of a commutator need

to be aggregated in order to produce a prediction for the commutator as a whole. While this might be straightforward to do, it is not the focus of this paper. We first wish to find good classifiers on the segment level before dealing with any meta-classifiers.

2.3.3 Optimizing classifier performance. Classifier performance can be measured in several ways, ranging from classification accuracy to the F-measure to other, even custom functions that depend on the domain (as was done for the two-class case in [5]). While we acknowledge that a similar custom function would be beneficial also for our five-class problem, where false 'no defect' classifications bear more serious consequences than other types of misclassifications, classification accuracy is chosen for now, since it is easier to interpret. Classification accuracy is estimated with 10-fold CV, which is a popular technique for predicting classifier performance on unseen instances and has been used also in the three previous studies [4, 5, 3].

In order to find the values of image processing parameters that will result in a classifier with high accuracy, an evolutionary algorithm is employed to search in the 12-dimensional space of image processing parameter values.

3. The Pitfalls of Overfitting

When building a classifier, some of the data is used for *training* the classifier, while the rest is used for *testing* its performance. Ideally, we would like both sets to be fairly large, since a lot of data is needed to train a classifier well, and a lot of data is needed to truthfully predict how it will perform on unseen instances. However, in reality, the data is often scarce and certain compromises need to be made.

One of the most popular approaches to estimate classifier performance is *k-fold cross-validation*, where the data is split into k sets of approximately equal cardinality. Next, $k-1$ of the sets are used for training the classifier, while the remaining set is used for testing its performance. This is repeated k times so that each set is utilized for testing exactly once. The average of all performance results is then used to estimate the accuracy of the classifier built on the entire data.

This and other cross-validation techniques (see [1] for a survey) were envisioned in order to avoid overfitting, i.e., constructing classifiers that describe noise in the data instead of the underlying relationships, since a classifier that overfits the training data performs poorly on unseen instances. This happens, for example, if the classifier is too complex. However, it has been long known [6] that there exists another source of overfitting that takes place despite cross-validation—if we compare a

high number of classifiers on a small set of instances, the best ones are usually those that overfit these instances.

This means, for example, that if an optimization algorithm that can produce thousands or even millions of solutions is used to find the best classifier for a problem, this best found classifier almost surely overfits the test data. Note however, that our study does not compare multiple classifiers on the same data. Our optimization problem resembles more that of feature selection where a subset of features needs to be found so that classifiers using these features will achieve good performance. The main difference to the standard feature selection is that we are performing feature selection in subgroups—for each of the 12 image processing parameters (one subgroup) we need to select exactly one among all possible values. The danger of overfitting despite using cross-validation has been noticed for feature selection problems as well [11].

In the following we present the results achieved with 10-fold CV for estimating classifier performance on our optimization problem, which show overfitting patterns, and analyze increased pruning and repeated cross-validation as possible alternatives to amend this issue.

4. Experimental Study

4.1 Experimental Setup

The experiments were performed on the commutator soldering domain from previous studies that contains 363 instances with uneven distribution of classes (see Table 1).

Table 1: The commutator soldering domain.

Class	Number of instances	Frequency [%]
No defect	212	58.4
Metalization defect	35	9.6
Excess of solder	35	9.6
Deficit of solder	49	13.5
Disorientation	32	8.8
Total	363	100.0

All computer vision methods were implemented using the Open Computing Language (OpenCL) [7], or more precisely, the OCL programming package [8], an implementation of OpenCL functions in the Open Computer Vision (OpenCV) library [9].

The decision trees were built using the J48 algorithm from the Weka machine learning environment [16], which is a Java implementation of Quinlan's C4.5 decision tree building algorithm [10]. The trees were constructed with default J48 parameter values except for the increased pruning case (more details are given in Section 4.3).

For optimization we use a self-adaptive DE algorithm called jDE [2] with a population of 80 solutions. The stopping criterion for the algorithm was set to 1000 generations. For each set of experiments nine runs have been performed and all presented results show the average values over the nine runs.

4.2 Results of Single Cross-Validation

First, we look at what happens when single 10-fold CV is used to estimate classifier accuracy (see top plot in Fig. 2). The black line shows that jDE is able to find increasingly more accurate classifiers as the evolution progresses. In order to check if these classifiers present signs of overfitting, we perform the following additional assessment. For each run and each best classifier from the population, we estimate the classifier using 10-fold CV ten more times. The span of these accuracies averaged over the nine runs is presented in red.

The increasing gap between the black line and the red area means that classifiers that are good on the default split of instances into cross-validation sets perform considerably worse when they are tested again on ten different cross-validation splits, i.e., the classifiers overfit the default cross-validation split. This happens because we are exploring a large number of classifiers and incidentally optimize them also with regard to the default cross-validation split.

Note that this kind of overfitting is different to the 'usual' one, where the classifier overfits the given instances. While we are probably experiencing both, we cannot know about the second one without testing the classifiers on a large number of unseen instances, which we unfortunately do not posses. We have experimented with reserving a small part of data for validation purposes as was done in [14], but found that this approach is not suitable for our case because of the small number of instances at our disposal. Since we have five classes with uneven distribution of instances, it proved very difficult to find representative instances for validation. Without a representative validation set the resulting estimation of overfitting can be too biased to rely on.

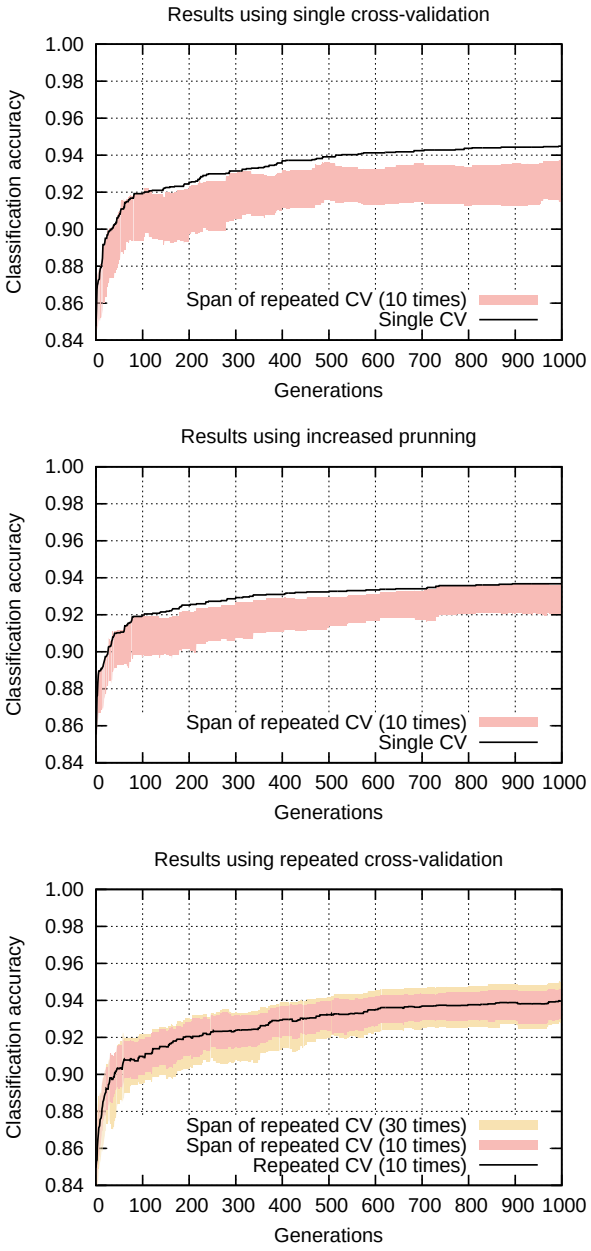


Figure 2: Experimental results of single cross-validation (top plot), increased pruning (middle plot) and repeated cross-validation (bottom plot). The black line shows the best classification accuracy found by jDE, while the red and yellow ares denote the span of accuracy values when the current-best classifier is re-estimated using additional cross-validation. All plots show average values over nine runs.

4.3 Results of Increased Pruning

Next, we investigate whether increased pruning of decision trees can help improve their generalization ability in our case. Note that the original decision trees (the J48 trees with default parameter settings) used in the previous experiments were also pruned. Here, we intensify the pruning by increasing the m parameter of the J48 algorithm that defines the minimal number of instances in any tree leaf from 2 to 5. The results of these experiments are presented in the middle plot in Fig. 2.

While the gap between the single cross-validation and the re-estimation using repeated cross-validation is smaller than in the previous experiments, the overfitting is still obvious. We can conclude that increased pruning does not alleviate much the overfitting brought by optimization.

4.4 Results of Repeated Cross-Validation

Finally, we explore the case when the fitness of the decision trees built with default parameter values is determined as the average of 10 different assessments by 10-fold CV. Again, we perform an additional assessment of the classifiers. This time we add to the repeated cross-validation 20 new estimations (for a total of 30) to see how they compare. The bottom plot in Fig. 2 shows there are no big differences when additional cross-validation results are added, indicating that repeated cross-validation is less prone to overfitting brought by optimization than single cross-validation. The average accuracy of the current-best classifiers over 10 and 30 repetitions are very similar, which suggests 10 repetitions can be chosen over 30 as they require less time.

These results seem to contradict the ones presented in [15], however this is not the case. In a series of experiments, [15] compares the estimates of classification accuracy from single 10-fold CV, 10-fold CV repeated 10 times and 10-fold CV repeated 30 times to a simulated 'true' performance of the classifier on unseen data. The results show that although the confidence interval narrows when increasing the number of cross-validation repetitions, this does not necessarily mean that the accuracy estimate will converge to the 'true' accuracy. The authors argue that the reason for this behavior is that the same data is continuously being resampled in repeated cross-validations. The experiments in [15] tackle the 'usual' overfitting problem, which is not the subject of this paper. We are concerned with the overfitting brought by optimization and find that repeated cross-validation can alleviate it.

5. Conclusion

We have presented a challenging real-world problem of estimating the quality of the commutator soldering process. We wish to find a classifier able to distinguish among joints soldered well and those that have one of the four possible defects. The problem is tackled using a combination of computer vision, machine learning and evolutionary optimization methods. In essence, we are searching for parameter settings of computer vision methods that can yield a highly accurate classifier.

Since an optimization algorithm that explores a large number of solutions to this problem is being used, we have been confronted with the problem of overfitting. We performed some experiments that have shown how overfitting can be detected and discussed on possible ways to amend it. From the results we conclude that repeated cross-validation can be used to diminish the overfitting bias brought by optimization. However, this results cannot be generalized to other machine learning problems without additional experiments that include a number of other datasets. This is a task left for future work.

The presented real-world problem is not yet solved and we can see many directions for future work. First, since the accuracies achieved are still not good enough for automotive industry standards, our main focus will be to try to improve on that (possibly by not producing even more overfitting). This can be tried, for example, by choosing other image features in addition to the six we have right now, or by trying more sophisticated classifiers than decision trees. Also, we intend to consider other measures of classifier performance beside accuracy. For example, we could use a specialized aggregation function or try to use a multiobjective approach. Finally, we will have to eventually combine the classifications of individual commutator segments into a single classification of the commutator as a whole.

Acknowledgment: This work was partially funded by the ARTEMIS Joint Undertaking and the Slovenian Ministry of Economic Development and Technology as part of the COPCAMS project (`http://copcams.eu`) under Grant Agreement no. 332913, and by the Slovenian Research Agency under research program P2-0209.

The authors wish to thank Valentin Koblar for valuable support regarding the application domain and computer vision issues, and Bernard Ženko for helpful discussions on machine learning algorithms.

References

[1] S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

[2] J. Brest, S. Greiner, B. Bosković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[3] E. Dovgan, K. Gantar, V. Koblar, and B. Filipič. Detection of irregularities on automotive semiproducts. *Proceedings of the 17th International Multiconference Information Society (IS)*, pages 22–25, 2014.

[4] V. Koblar and B. Filipič. Designing a quality-control procedure for commutator manufacturing. *Proceedings of the 16th International Multiconference Information Society (IS)*, pages 55–58, 2013.

[5] V. Koblar, E. Dovgan, and B. Filipič. Tuning of a machine-vision-based quality control procedure for product components in automotive industry. Submitted for publication, 2014.

[6] A. Y. Ng. Preventing "overfitting" of cross-validation data. *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, pages 245–253, 1997.

[7] OpenCL: The open standard for parallel programming of heterogeneous systems. `http://www.khronos.org/opencl/`. Retrieved January 25, 2016.

[8] OpenCL module within the OpenCV library. `http://docs.opencv.org/modules/ocl/doc/introduction.html`. Retrieved January 25, 2016.

[9] OpenCV: Open source computer vision. `http://opencv.org/`. Retrieved January 25, 2016.

[10] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[11] R. B. Rao, G. Fung, and R. Rosales. On the dangers of cross-validation. An experimental evaluation. *Proceedings of the SIAM Conference on Data Mining (SDM)*, pages 588–596, 2008.

[12] T. Robič and B. Filipič. DEMO: Differential evolution for multiobjective optimization. *Lecture Notes in Computer Science*, 3410:520–533, 2005.

[13] R. Storn and K. V. Price. Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[14] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Automated selection and hyper-parameter optimization of classification algorithms. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855, 2013.

[15] G. Vanwinckelen and H. Blockeel. On estimating model accuracy with repeated cross-validation. *Proceedings of the 21st Belgian-Dutch Conference on Machine Learning (BeneLearn)*, pages 39–44, 2013.

[16] Weka Machine Learning Project. `http://www.cs.waikato.ac.nz/ml/weka/index.html`. Retrieved January 25, 2016.

ROBUST MULTI-OBJECTIVE OPTIMIZATION OF WATER DISTRIBUTION NETWORKS

Taishi Ohno, Hernán Aguirre, Kiyoshi Tanaka
Faculty of Engineering, Shinshu University, Wakasato, Nagano-shi, Japan
15tm209f@shinshu-u.ac.jp, ahernan@shinshu-u.ac.jp, ktanaka@shinshu-u.ac.jp

Abstract This work studies the incorporation of robustness to changes in water demand within the evolutionary search in order to enhance the design optimization of water distribution networks. An unconstrained multi-objective formulation of the problem is used together with the AεSεH algorithm, a multi- and many-objective evolutionary algorithm known to scale up well with larger population sizes and number of objectives. An effective incorporation of robustness within the evolutionary process will open the possibility to incorporate additional important objectives of water distribution networks that can be optimized simultaneously.

Keywords: Evolutionary algorithms, Multi- and many-objective algorithm AεSεH, Multi-objective design optimization, Water distribution networks.

1. Introduction

A water distribution network refers to the drinking water supply backbone of an urban area. Its main components are reservoirs, pipes, and nodes. The diameters of the pipe are chosen from a set of commercially available sizes and there is a certain demand of water associated to each node that must be fulfilled. The design optimization of water distribution networks aims to obtain optimal combinations of pipe diameters that minimize cost while keeping enough pressure at the nodes to satisfy the water demand. In addition, an optimal design must be reliable to pipe failures and robust to changes in water demand.

The design optimization of water distribution networks has been usually formulated as a single objective cost-minimization problem with a constraint on the minimum pressure to be available at the consumer nodes in order to satisfy the demand. In addition, reliability of the op-

timal solution to failures in pipes and robustness to changes in water demand have been usually verified after optimization.

Recently, design optimization of water distribution networks using multi-objective unconstrained formulations has been explored [5], where objectives related to cost, nodal pressure, and reliability to pipe failures are optimized simultaneously. The multi-objective formulation allows to generate a number of non-dominated solutions to provide the decision maker with insights on the trade-off between the objectives. Also, it incorporates the reliability to pipe failures within the optimization process rather than as a posterior step.

A comparison between the popular NSGA-II and SMS-EMOA on benchmark water distribution problems is presented in [7, 8] following the same multi-objective formulation used in [5]. Better results are reported for SMS-EMOA than for NSGA-II. This is attributed to the better scalability of the SMS-EMOA approach for problems with three or more objectives. In the same work, an attempt to consider within the evolutionary optimization process the robustness of the network to changes in water demand is pursued. However, for the budget of fitness evaluations and population size used in [7, 8], the conclusion was that solutions optimized for a single profile of water demand are more robust to changes in demand than solutions optimized simultaneously for multiple profiles. This is rather counterintuitive and unexpected.

In this work, our aim is to clarify whether robustness to changes in water demand can be effectively incorporated within the evolutionary process in order to enhance the design optimization of water distribution networks. We follow the same multi-objective formulation used in [5, 7, 8] incorporating the AεSεH algorithm, a multi- and many-objective evolutionary algorithm known to scale up well with larger population sizes and number of objectives. An effective incorporation of robustness within the evolutionary process will also open the possibility to incorporate additional important objectives of water distribution networks that can be optimized simultaneously.

2. Method

2.1 Optimization System

A water distribution network consists of pipes, nodes (pipe junctions), pumps, valves and storage tanks or reservoirs. For each node there is associated a water demand that must be satisfied. To simulate the water distribution network we use EPANET [9]. EPANET performs extended period simulation of hydraulic and water quality behavior within pressurized pipe networks. EPANET tracks the flow of water in each pipe,

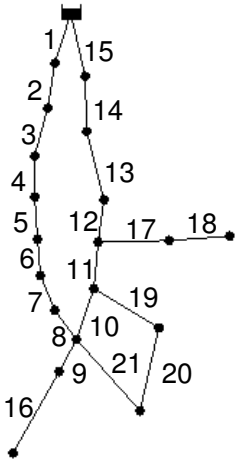


Figure 1: New York Water Distribution Network Model

Table 1: New York Water Supply Pipe Types [8]

Code	D (in)	Price ($/ft)	H-W
0	36	93.5	100
1	48	134	100
2	60	176	100
3	72	221	100
4	84	267	100
5	96	316	100
6	108	365	100
7	120	417	100
8	132	469	100
9	144	522	100
10	156	577	100
11	168	632	100
12	180	689	100
13	192	746	100
14	204	802	100

the pressure at each node, and the height of water in each tank. In can also track the concentration of a chemical species throughout the network. In addition, water age and source tracing can also be simulated.

In this work we assume a remediation problem, where there is a water distribution network in operation and all its pipes could be replaced by new ones. Thus, the coordinates where the pipes are laid down and the lengths of the pipes are already determined. Also, the available commercial diameters of the pipes and the baseline water demand for the nodes are known. We are interested in finding the diameter of the pipes that render robust configurations for the water distribution network when the nodes of the network are subjected to various demands of water.

An evolutionary algorithm is used to search for optimal combinations of pipes diameters. Thus, a solution encodes the diameters of all pipes of the network. For each solution explored by the evolutionary algorithm, we combine the information of the network with the diameters of the pipes provided by the solution to create the required input file for EPANET, run the simulator, and use its output to compute the fitness of the solution. As benchmark problem, we use the New York Water Distribution Network model, widely used in the literature and illustrated in Fig. 1.

In the following we explain the algorithm, genetic representation, fitness function, and operators used in this work.

2.2 AεSεH

In this work we use the Adaptive ε-Sampling and ε-Hood (AεSεH) [1, 2] algorithm to search optimal solutions. AεSεH is an elitist evolutionary multi- and many-objective optimizer that applies ε-dominance [6] principles both for survival selection and parent selection.

AεSεH follows the main steps of a population-based evolutionary algorithm, i.e., parent selection, offspring creation and survival selection, adjusting its operation depending on whether the population contains dominated solutions or not.

To perform survival selection, the current population and its offspring are combined and divided into non-dominated fronts using the non-dominated sorting procedure. If the number of non-dominated solutions in the first front is smaller than the population size, the sorted fronts of non-dominated solutions are copied one at a time to the next population until it is filled; if the last copied front overfills the population, the required number of solutions are chosen randomly from it to have the exact number specified by the population size. On the other hand, if the number of non-dominated solutions in the first front is larger than the population size, the first front is truncated to the size of the population using the *ε-sampling* procedure. *ε-sampling* randomly chooses solutions from the first front to include them in the surviving population, eliminating from the front those solutions that are ε-dominated by the chosen samples. As a result, solutions in the next population are spaced according to the $\mathbf{f}(\mathbf{x}) \mapsto^{\varepsilon_s} \mathbf{f}'(\mathbf{x})$ mapping function and parameter ε_s used to compute ε-dominance between solutions.

For parent selection, the algorithm first uses a procedure called *ε-hood creation* to cluster solutions in objective space and then applies *ε-hood mating* to select parents. When all solutions in the population are non-dominated, *ε-hood creation* selects *randomly* an individual from the population and applies ε-dominance with mapping function $\mathbf{f}(\mathbf{x}) \mapsto^{\varepsilon_h} \mathbf{f}'(\mathbf{x})$ and parameter ε_h. A neighborhood is formed by the selected solution and its ε_h-dominated solutions. Neighborhood creation is repeated until all solutions in the population have been assigned to a neighborhood. *ε-hood mating* sees the neighborhoods as elements of a list and visits them one at the time in a round-robin schedule. The first two parents are selected *randomly* from the first visited neighborhood in the list. The next two parents are selected randomly from the second neighborhood in the list, and so on. When the end of the list is reached, parent selection continues with the first neighborhood in the list. On the other hand, when dominated solutions are present in the population, *ε-hood creation* makes sure that the solution sampled to create the neighborhood

C_1	C_2	C_3	...	C_i	...	C_P

Figure 2: Genetic Representation

is a non-dominated solution and *ε-hood mating* uses *binary tournaments* based on dominance rank to select parents within the neighborhoods. Both epsilon parameters ε_s and ε_h used in survival selection and neighborhood creation, respectively, are dynamically adapted during the run of the algorithm.

This algorithm has been shown to work effectively on continuous and discrete multi- and many-objective optimization problems [1, 2, 3, 4]. Further details about the algorithm can be found in [1] and [2].

2.3 Genetic Representation

This paper considers n types of pipes, identified by a unique integer code $C = \{1, 2, ..., n\}$. Table 1 shows the codes of the type of pipes of the New York Network used in this work, together with their characteristics such as diameter, unit length price, and Hazen-Williams (H-W) roughness coefficient that depends on the material of the pipe. Fig. 2 illustrates the representation of the solutions used for optimization, where P is the total number of pipes of the water distribution network, C_i is the code of the i-th pipe in the network, i.e $C_i \in C$. Thus, in the New York Problem, the size of the search space is $15^{21} = 4.99 \times 10^{26}$.

2.4 Fitness Function

There are several criteria for evaluating a water distribution network in the literature. Here, we use the total cost of the pipes of the water distribution network, demand supply ratio, and system entropy [7] .

Total cost of the pipes f_1 is expressed as the sum needed for inserting new pipes in the network, priced in segments of unit length with a certain diameter.

$$f_1 = \sum_{i \in P} price\,(C_i) \cdot L_i, \tag{1}$$

where $price\,(C_i)$ is the unit length price of the new pipe i of code C_i (the prices are determined by the diameter of the pipe and are included in the table of available commercial diameters, which is supplied with the test problem), and L_i is the length of the pipe i.

The demand supply ratio is a measure of reliability of the distribution network that computes to what extent the water demand at the various

internal nodes in the network is met. To compute the demand supply ratio, first a functional relationship between pressure and water demand at a node is established. This determines the portion of the water demand at a node that could be satisfied and is expressed as follows

$$Q_j{}^{met} = \begin{cases} 0 & p_j \leq p^{zero} \\ Q_j \sqrt{\frac{p_j - p^{zero}}{p^{req} - p^{zero}}} & p^{zero} \leq p_j \leq p^{req} \\ Q_j & p_j > p^{req}, \end{cases} \tag{2}$$

where j is the node index, N is the total number of nodes, Q_j^{met} is portion of the water demand met at node j, Q_j is the nodal demand (water demand drawn from internal node j), p_j is the effective pressure remaining at internal node j, p^{zero} is the pressure corresponding to zero nodal water demand satisfaction, and p^{req} is the pressure required to satisfy the nodal water demand completely.

The average demand supply ratio f_2, normalized by the nodal demands, is expressed by

$$f_2 = \frac{\sum_{j \in N} Q_j^{met}}{\sum_{j \in N} Q_j}. \tag{3}$$

The system entropy is also a function of reliability of the network. The water demand at a given internal node is ideally met using multiple different paths to that node. The required flow should be distributed over these routes as evenly as possible. This way, should a segment of the network fail, alternative routes exist that could still supply a reasonable part of the demand. The system entropy f_3 is expressed as follows:

$$s_j = \sum_{i:dest_i = j} \frac{-|q_i|}{in_j} \ln \frac{|q_i|}{in_j}, \tag{4}$$

$$S = \sum_{j \in N} \frac{in_j}{IN} s_j + \sum_{j \in N} \frac{-in_j}{IN} \ln \frac{in_j}{IN}, \tag{5}$$

$$f_3 = exp(S), \tag{6}$$

where s_j is the entropy at node j, S is the entropy of the entire network, q_i is the water flow in pipe i, $dest_i$ is the destination of the water flow in pipe i, in_j is the water flow transported towards node j by the incoming pipes i connected to it, and IN is the sum of all incoming flows in the network. Note that system entropy f_3 is expressed as the exponential of S.

These functions are optimized simultaneously. The investment cost of installing water pipes f_1 is to be minimized, whereas the demand supply ratio f_2 and the system entropy f_3 are to be maximized.

2.5 Crossover and Mutation Operators

In this work, we use two point crossover with crossover rate P_c. Mutation is applied to all variables with mutation rate P_m. If a variable is chosen, mutation either increases or decreases with equal probability the code of the pipe by one. That is, mutation changes the current diameter of the pipe to the next higher/lower available diameter.

3. Methods of Evaluation for Evolution and Robustness

In this work we search robust optimal configurations of water distribution networks using two different methods to evaluate solutions during evolution. The first method (Method I) evolves solutions evaluating them using the baseline profile for water demand in the nodes of the network given by the benchmark problem. Thus, the fitness vector of a solutions $\mathbf{x}$ is

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x})). \tag{7}$$

The second method (Method II) establishes in advance a set E of N_E different profiles of water demand for the nodes of the network; during evolution solutions are evaluated in all N_E profiles of the set E and fitness is computed as the average of these evaluations. Thus, the fitness vector of a solutions $\mathbf{x}$ is

$$\mathbf{f}(\mathbf{x}) = \frac{1}{N_E} \sum_{i=1}^{N_E} \mathbf{f}^{(i)}(\mathbf{x}), \tag{8}$$

where $\mathbf{f}^{(i)}(\mathbf{x}) = (f_1^{(i)}(\mathbf{x}), f_2^{(i)}(\mathbf{x}), f_3^{(i)}(\mathbf{x}))$ is the fitness vector of the solution evaluated with the i-th profile of the set E.

The approximation of the Pareto optimal set ($\mathcal{A}_{POS}$) generated by an algorithm consists of the non-dominated solutions computed from all solutions explored during the search. We use another set R of N_R profiles of water demand for the nodes in order to assess the robustness of the evolved solutions and compare the two methods fairly. Thus, the approximations $\mathcal{A}_{POS}$ found by the algorithms are evaluated again with the profiles in R by

$$\mathbf{f}(\mathbf{x}) = \frac{1}{N_R} \sum_{i=1}^{N_R} \mathbf{f}^{(i)}(\mathbf{x}), \tag{9}$$

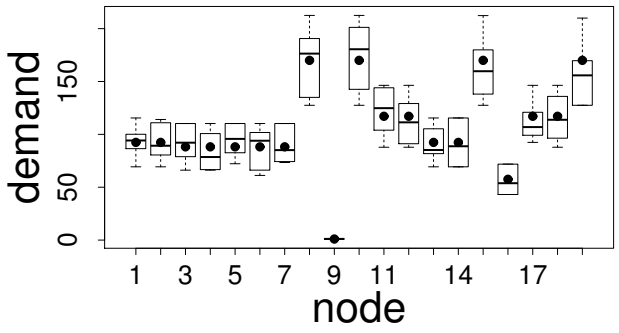


Figure 3: Profiles of water demand to measure robustness of evolved solutions

where $\mathbf{f}^{(i)}(\mathbf{x}) = (f_1^{(i)}(\mathbf{x}), f_2^{(i)}(\mathbf{x}), f_3^{(i)}(\mathbf{x}))$ is the fitness vector of the solution evaluated with the i-th profile of the set R. The set of non-dominated solutions is obtained for each method with this new fitness value and call the new approximation set as $\mathcal{A}_{POS}^R$. Both methods of evolutions are compared computing the hypervolume of their approximations $\mathcal{A}_{POS}^R$.

The profiles of water demand for sets E and R are created by randomly changing the baseline demand of the nodes by

$$\hat{d}_j = d_j(1 + 0.25 \times U(0,1)), \tag{10}$$

where, d_j is the baseline demand of node j, and $U(0,1)$ is random number sampled from normal distribution with expectation 0 and variance 1. Fig. 3 shows boxplots of the demand profiles used for the nodes of the network to evaluate robustness of the evolved solutions. The dot inside each boxplot shows the baseline demand.

4. Simulation Results

4.1 Experimental Setup

As mentioned above, in this work we use as benchmark problem the New York Model of Water Distribution that consists of a reservoir, 19 nodes, and 21 pipes. For the evolutionary algorithm we use the following parameters. Crossover rate was set to 1.0, mutation rate to $1/|P|$, population size to 300, and number of function evaluations is se to 300,000. The number of water demand profiles to evaluate solutions in Method I is 1 (the baseline profile) and the number of water demand profiles in Method II is $N_E = 10$. Thus, to keep the same number of fitness

evaluations in both methods, in Method I the number of generations is set to 1,000 whereas in Method II the number of generations is 100. The number of profiles to evaluate robustness of the approximation sets is $N_R = 200$. Unless stated otherwise, we report results of ten independent runs of the algorithms.

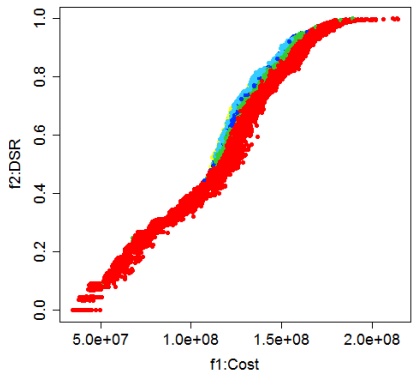
4.2 Method I: Evolution with Single Profile for Water Demand

The three-objectives fitness vector of the approximation of the Pareto optimal set $\mathcal{A}_{POS}$ obtained by Method I for one of the runs of the algorithm is projected to two-objectives planes as shown in Fig. 4. Here, solutions are colored by the value of the third not displayed objective. The colors codes are yellow, light blue, dark blue, green, and red in order of objective value from low to high. The range between the maximum and minimum objective values is divided evenly into five subranges and assigned to the colors code.

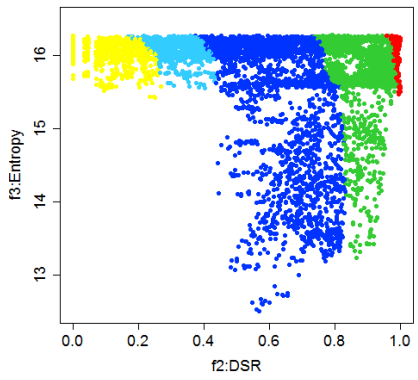
By displaying the fitness of solutions evenly divided by color we can verify the trade-offs in objective space between solutions in the Pareto optimal set. As an example, we focus on Fig. 4 (b). Recall that the demand supply ratio DSR f_2 and entropy f_3 are maximized, whereas cost f_1 is minimized. Fig. 4 (b) plots f_2 and f_3, coloring by f_1. Note that as DSR f_2 increases (DSR gets better) cost f_1 changes from yellow to light blue, $\cdots$, to red (cost gets higher). Hence, there is a clear trade-off between DSR and cost; improving DSR makes the network more expensive and vice versa. It is also interesting to note that high values of entropy can be achieved for both low and high cost networks.

4.3 Method II: Evolution with Multiple Profiles for Water Demand

Similar to Method I, Fig. 5 shows the three-objectives fitness vector of the approximation $\mathcal{A}_{POS}$ by Method II projected to two-objectives planes. Fig. 5 looks similar to Fig. 4 and the trade-off between objective functions can be verified, particularly between cost and DSR. However, note that in Fig. 5 only networks with high entropy remain. That is, solutions with low entropy have been eliminated from the Pareto optimal set by the method that evaluates solutions with multiple profiles of water demand during the evolutionary process.

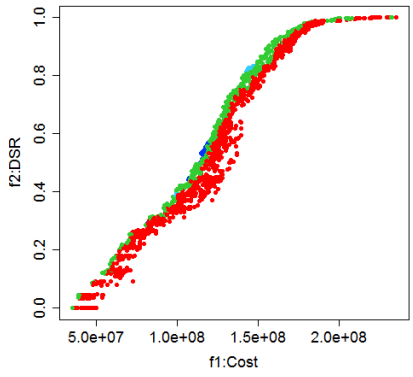


(a) Cost - DSR, colored by Entropy

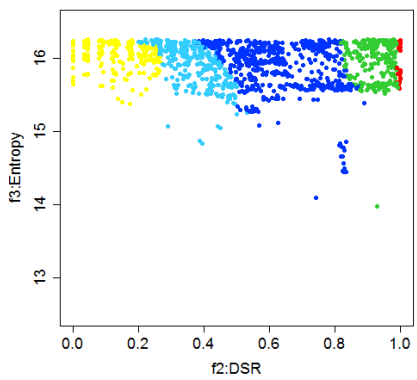


(b) DSR - Entropy, colored by Cost

Figure 4: Method I



(a) Cost - DSR, colored by Entropy



(b) DSR - Entropy, colored by Cost

Figure 5: Method II

4.4 Analysis Based on Hypervolume

In this section we compare the hypervolume HV of the approximations $\mathcal{A}^R_{POS}$ obtained by both methods. The fitness values given by Eq.(9) are used to compute the HV. Table 2 reports the average HV in ten independent runs of the algorithms. The standard deviation of HV is presented in parenthesis. The table also shows the average standard deviation of the objective values for each fitness function and the average cardinality of the approximations for both methods. Note that the HV value by Method II is higher than by Method I. This shows that overall solutions with better convergence and diversity in objective space are found by Method II, suggesting that by using Method II is posible to find designs of the water distribution network that can cope better with changes in the demand of water.

Table 2: Hypervolume and standard deviation of the approximations $\mathcal{A}^R_{POS}$

	Method I	Method II
HV	0.59022 (0.026)	0.62216 (0.028)
stdev f_1	0.21	0.21
stdev f_2	0.29	0.28
stdev f_3	0.21	0.19
Number of POS	6289.9 (336.99)	1407.1 (96.68)

Remember that the number of fitness evaluations is the same in both methods, however the number o generations is different. This indicates that robust evolution can be performed in this problem using a reduced number of generations evaluating simultaneously a number of water demand profiles rather than evolving solutions for a larger number of generations evaluating just one profile. The oposite conclusion was reached in [7, 8], where solutions by the method that evolves solutions evaluating just one demand profile showed higher robustness to changes in the demand of water. This is because [7, 8] used fewer fitness evaluations, Method II evolved for too few generations and could not converge.

Method II achieves solutions with f_3 higher than Method I, as can be seen in Fig. 5 compared to Fig. 4. Also, the standard deviation of f_3 by Method II is smaller than Method I as shown in Table 2. As mentioned above, solutions with low f_3 are eliminated by Method II. This suggests that changes in demand affect system entropy.

It is also worth noting from Table 2 that Method II finds fewer solutions than Method I. One reason for this is that Method I runs for a larger number of generations and explores many more solutions than Method II.

5. Conclusion

This work has shown that robustness to changes in water demand can be effectively incorporated within the evolutionary process in order to enhance the multi-objective design optimization of water distribution networks. We followed an unconstrained multi-objective problem formulation where three objectives were optimized simultaneously: cost of the network, nodal pressure based on demand supply ratio, and reliability to pipe failures based on a measure of entropy. Robustness to changes in the demand of water was pursued and two approaches were compared. One approach to robustness was to guide evolution based on a single

profile of water demand, whereas the other approach used multiple profiles. We showed that using the multi- and many-objective evolutionary AεSεH algorithm the approach that evolves solutions based on multiple profiles attained significantly more robust networks. This clarifies previous counterintuitive and misleading findings regarding robustness incorporated within NSGA-II and SMS-EMOA reported elsewhere and opens the possibility to incorporate additional important objectives of water distribution networks to be optimized simultaneously. In this work we used the hypervolume to compare the robustness of the optimal solutions. In the future we would like to explore other indicators to study robustness and pursue many-objective formulations for design optimization of water distribution networks with and without constraints.

References

[1] H. Aguirre, A. Oyama, and K. Tanaka. Adaptive ε-sampling and ε-hood for evolutionary many-objective optimization. *Lecture Notes in Computer Science*, 7811:322–336, 2013.

[2] H. Aguirre, Y. Yazawa, A. Oyama, and K. Tanaka. Extending AεEεH from many-objective to multi-objective optimization. *Lecture Notes in Computer Science*, 8886:239–250, 2014.

[3] H. Aguirre, S. Zapotecas, A. Liefooghe, S. Verel, and K. Tanaka. Approaches for Many-objective Optimization: Analysis and Comparison on MNK-landscapes. *Lecture Notes in Computer Science*, 9554, 2015.

[4] R. Armas, H. Aguirre, S. Zapotecas-Martínez, and K. Tanaka, Traffic Signal Optimization: Minimizing Travel Time and Fuel Consumption. *Lecture Notes in Computer Science*, 9554, 2015.

[5] K. Formiga, F. Chaudhry, P. Cheung, and L. Reis. Optimal Design of Water Distribution System by Multiobjective Evolutionary Methods. *Lecture Notes Computer Science*, 2632:677-691, 2003.

[6] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[7] E. Reehuis. Multiobjective Robust Optimization of Water Distribution Networks. Master's Thesis Leiden University, Netherlands, 2010.

[8] E. Reehuis, J. Kruisselbrink, A. Deutz, T. Back, and M. Emmerich. Multiobjective Optimization of Water Distribution Networks Using SMS-EMOA. *Proceedings of the International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems*, pages 269–279, 2011.

[9] L. Rossman. EPANET 2 Users Manual. Technical Report EPA/600/R-00/057, Water Supply and Water Resources Division, National Risk Management Research Laboratory, U.S. EPA, Cincinnati, OH, USA, 2000.

MODELING AND OPTIMIZATION OF A ROBUST GAS SENSOR

Margarita A. Rebolledo C., Sebastian Krey, Thomas Bartz-Beielstein, Oliver Flasch, Andreas Fischbach, Jörg Stork
SPOTSeven Lab, TH Köln, Gummersbach, Germany
margarita.rebolledo@th-koeln.de, sebastian.krey@th-koeln.de, thomas.bartz-beielstein@th-koeln.de, oliver.flasch@th-koeln.de, Andreas.fischbach@th-koeln.de, joerg.stork@th-koeln.de

Abstract In this paper we present a comparison of different data driven modeling methods. The first instance of a data driven linear Bayesian model is compared with several linear regression models, a Kriging model and a genetic programming (GP) model. The models are build on industrial data for the development of a robust gas sensor. The data contain limited amount of samples and a high variance. The mean square error of the models implemented in a test dataset is used as the comparison strategy. The results indicate that standard linear regression approaches as well as Kriging and GP show good results, whereas the Bayesian approach, despite the fact that it requires additional resources, does not lead to improved results.

Keywords: Bayesian modeling, BMA, Design of experiments, Genetic programming, Kriging, Lasso, Linear regression.

1. Introduction

Theoretically, there are many advantages for the implementation of Bayesian analysis [9]. The use of Bayesian models might represent a good alternative for industrial applications as they produce more informative results. The generation of a data-driven model to optimize the development of a carbon-monoxide sensor provides an opportunity to test these assertions on limited and sparse data. As a first approach, Bayesian robust linear regression is implemented and compared to standard regression methods and a genetic programming approach. Our goal is to learn the difference in performance from the tested methods when applied to this kind of data and to set future considerations for working with Bayesian models in a more demanding fashion.

In recent years the need to reduce air pollution levels has gained more importance in the automotive industry. The efficiency increase of the motor combustion process plays an important role for the reduction of pollution levels. This efficiency can be indirectly measured by monitoring the concentrations of carbon monoxide and other harmful gases released into the atmosphere. This paper focuses on the modeling and optimization of a carbon monoxide in-situ sensor. The sensor should be able to discern the carbon-monoxide concentration apart from the other exhaust gases. This is a difficult goal, because the sensor is exposed to and influenced by the other gases. Thus, the sensor output is not expected to be a direct result of the concentration of the gas of interest. Instead it will be the result of an underneath process influenced by all the other gases. Similar or relatable problems have been addressed in the literature. In these instances the sensor had different manufacture methods, submitted to different gas mixtures or operative conditions. The modelling was addressed using either numerical methods based on the sensor composition [1] or data driven methods like neuronal networks and partial least squares [2, 6].

At the end of our analysis we hope to obtain models from different methods with an improved sensitivity to carbon-monoxide concentrations. The models will be compared in order to check the performances differences and possible improvement opportunities.

This paper is structured as follows: Section 2 describes the research configuration, i.e., data and experimental designs. Key features of the algorithms are introduced in Section 3. Section 4 presents results from the experiments. Finally, a discussion of the results in given in Section 5.

2. Problem

2.1 Data Description

The data was collected following a *response surface design of experiments* (RS-DoE). This design constraints itself to the maximum and minimum expected concentration values of each gas under normal working conditions. Given the cost and time consumption required for the experiments, only a limited amount of samples could be measured. The minimum number of samples required to have a good system description and the real limit of possible realizable samples in the industrial testing station was balanced. Finally, a sample size of 80 was chosen. A summary of the data is shown in Table 1. This application example is anonymized due to confidentiality reasons. The data were standardized, meaning that every sample had its mean subtracted and was then divided by the standard deviation. The different gases are denominated

as the variables $X1$ to $X7$. The values of interest correspond to the columns denominated $Y1$ and $Y2$, which are the sensor measurements. All the models will use this dataset as the training set.

Table 1: Overview of the standardized dataset used to generate the models of the sensors. Here every input of the model is denoted by an X and every sensor output is denoted by an Y.

	X1	**X2**	**X3**	**X4**	**X5**	**X6**	**X7**	**Y1**	**Y2**
Minimum	-1.13	-1.21	-1.16	-1.13	-1.15	-1.17	-1.00	-1.94	-2.06
1st Quartal	-1.13	-1.21	-1.16	-1.13	-1.15	-1.17	-0.82	-0.63	-0.58
Median	0.09	0.03	0.12	0.08	0.08	0.05	-0.39	0.06	0.09
Mean	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3rd Quartal	1.30	1.26	1.40	1.29	1.28	1.28	0.59	0.66	0.67
Maximum	1.30	1.26	1.40	1.29	1.31	1.28	3.79	2.32	2.28

A general idea of the system behavior can be obtained by examining the correlation between the system output and inputs as shown in Table 2. Some assumptions can be made about the influence each variable has on the sensor output: not all parameters seem to have the same influence on the sensor output. Also, the sensors do not behave identically. Figure 1 shows the effect the two most strongly correlated parameters, X1 and X4, respectively, have on the sensors signal.

A second dataset, denominated *test set*, which follow the characteristic of the previously described training set was made available to validate the results of the obtained models.

2.2 Experimental Design Considerations

The data described in Section 2.1 was retrieved during tests based on an experimental design suitable for fitting models using the *response surface methodology* (RSM) [10]. First experimental results taken from screening design experiments indicated that a first order polynomial model is not sufficient, due to cross-sensitivity of the sensors. Therefore it was decided to use a RSM with two-factor interactions quadratic effects. In this case central composite designs would have been a logical

Table 2: Correlation between the system output and inputs for the training dataset

	X1	**X2**	**X3**	**X4**	**X5**	**X6**	**X7**
Y1	0.34	-0.19	-0.27	0.73	0.01	-0.00	-0.21
Y2	0.31	-0.16	-0.18	0.78	0.00	-0.03	-0.22

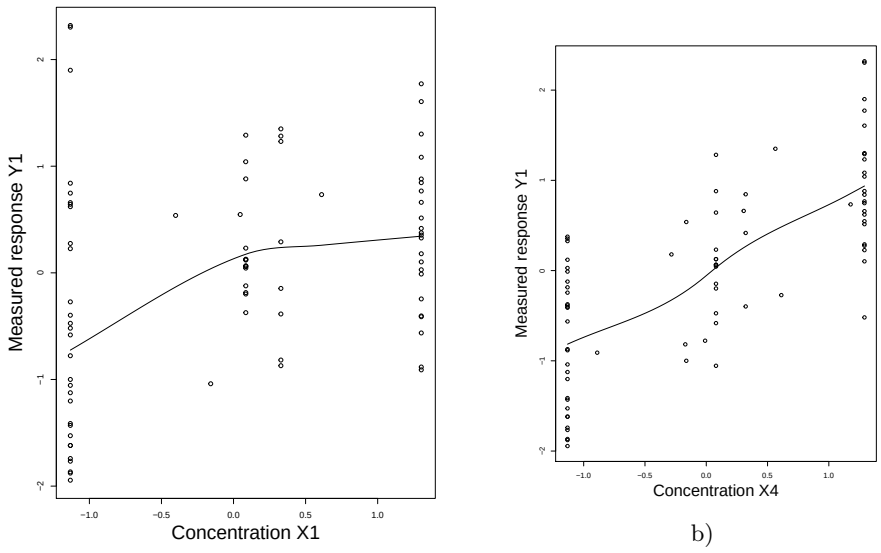


Figure 1: Scatter plots showing the general behavior of Y1 with respect to: a) the influence of X1 and b) the influence of X4.

choice. They are a combination of a box design, typically a full factorial or fractional factorial design and additional star or center points.

A *full factorial design* (FFD) with three levels for each of the six factors to estimate main effects and all quadratic terms would lead to $3^7 = 2187$ experiment runs at minimum. Choosing to divide factors in two levels, a two level FFD would still need at least $2^7 = 128$ runs, without any repetitions or center points. A valid system description would need even more runs.

To reduce the number of runs and be capable of fitting second order polynomial models a Box-Behnken design comes into consideration. But as a linear constraint on the input variables is limiting the sum of their values, almost all standard designs methods does not meet the requirements. Therefore, a more flexible design is needed. Using the statistical software JMP the RS-DoE was generated following the given constraints and applying the I-optimality criterion [7, 13]. The design was run and optimized a total of 80 times with 80 data points in order to obtain the best possible inference accuracy.

3. Algorithms

3.1 OLS

A linear model estimated by ordinary least squares is the natural first modeling attempt for data generated by an experimental design. Our baseline model for the comparison of the different modeling methods is the linear main effects model

$$f^1 : \hat{y} = \beta_0 + \sum_{i=1}^{7} \beta_i x_i.$$

In this work we used a RSM design, so beside the main effects the parameters of all two-way interactions and quadratic terms of the input variables can be estimated. This results in the full linear model

$$f^7 : \hat{y} = \beta_0 + \sum_{i=1}^{7} \beta_i x_i \sum_{i=1}^{7} \sum_{j=i}^{7} \beta_{ij} x_i x_j.$$

Based on the full linear model f^7 we applied variable selection based on an analysis of variance to get a more sparse model, which can be better interpreted. With a F-Test p-value of $\alpha = 0.01$ as the decision boundary for the inclusion into the final model, we obtained the model

$$f^2 : \hat{y} = \beta_0 + \sum_{i=1}^{4} \beta_i x_i + \beta_{14} x_1 x_4 + \beta_{34} x_3 x_4.$$

The *mean squared error* (MSE) as defined in Eq. 4 on page 276, was used for our comparisons. While the full linear model f^7 has a lower training error, i.e., MSE of 0.11 for sensor $Y1$ and 0.10 for sensor $Y2$, compared to the baseline model (MSE of 0.24 for sensor $Y1$ and 0.23 for sensor $Y2$), the prediction performance on the test dataset is very weak (MSE of 7.76 and 9.08). This is a strong indicator of overfitting.

The model f^2 has a MSE of 0.79 for sensor $Y1$ and 0.80 for sensor $Y2$, which is comparable (for sensor $Y1$) and little higher (for sensor $Y2$) than the baseline model. The residual standard error for the training set is lower (0.43 compared to 0.52 for sensor $Y1$ and 0.41 compared to 0.51 for sensor $Y2$) resulting in narrower confidence intervals for the parameters. The adjusted coefficient of determination (adjusted R^2) is 0.81 compared to 0.73 for sensor $Y1$ and 0.83 compared to 0.74 for sensor $Y2$. This means the inclusion of the two two-way interactions X1:X4 and X3:X4 has a large contribution for the explanation of the variance in the dataset, while the input variables X5, X6 and X7 have very little or no contribution and can be left out of the model.

3.2 Lasso

The *Least Absolute Shrinkage and Selection Operator* (Lasso) implements a selection method for linear models [15]. It selects solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent. The Lasso trains a linear model with a L_1 prior as regularizer.

Give a set of input measurements $X = \{x_i\}_{i=1}^n$ and an outcome measurement y, the lasso fits a linear model

$$\hat{y} = \beta_0 + \sum_{i=1}^{p} \beta_i x_i.$$

Let $\alpha \geq 0$ be a constant. The Lasso uses the following optimization criterion:

$$\min_{\beta} \frac{1}{2n} ||X\beta - y||_2^2 \text{ under the constraint } ||\beta||_1 \leq \alpha, \tag{1}$$

where $||\cdot||_1$ and $||\cdot||_2$ denote the L_1- and L_2-norm, respectively. The positive constant α is a tuning parameter. For large α values, the constraint $||\beta||_1 \leq \alpha$ in Eq. 1 has no effect and the usual linear least squares regression is performed. For smaller values of α, the solutions are shrunken versions of the least squares estimates. Decreasing the values of α forces the coefficients β_i's to become zero, i.e., choosing α results in selecting the number of predictors to use in a regression model. The Lasso can recover the exact set of non-zero weights (under certain conditions). Coordinate descent is used to fit the coefficients.

3.3 Kriging

Kriging or Gaussian process regression is a method of interpolation [14]. The n observations in an arbitrary data set, $Y = \{y_i\}_{i=1}^n$ can be associated as a single point sampled from some multivariate (n-variate) Gaussian distribution. The observations and the Gaussian process are related to each other by the covariance or kernel function $k(x_i, x_j)$. Kernel functions compute the distance between two samples in an arbitrary metric and apply a radial function to this distance. The squared exponential kernel, also known as the Gaussian *radial basis function* (RBF) kernel, is used in our study. This kernel is given by

$$k_1(x_i, x_j) = \sigma^2 \exp(-\theta \|x_i - x_j\|_2^2) \text{ with } \theta = \frac{1}{2l^2}. \tag{2}$$

The RBF kernel can be interpreted as a similarity measure, because values of this kernel decrease with distance. They range between zero

(in the limit) and one. The length parameter l in Eq. 2 determines the effect of other observations during interpolation at new x values. The RBF kernel was selected in our study, because Gaussian processes with this kernel generate smooth functions. Since noisy data were analyzed in our study, the white noise kernel $k_2(x_i, x_j) = \sigma^2\delta(x_i, x_j)$, where $\delta(x_i, x_j)$ denotes the Kronecker delta function, was added to the RBF kernel. Hence, we used the kernel function $k = k_1 + k_2$.

3.4 Robust Bayesian Modeling

Bayesian modeling is the mathematical relocation of credibility of parameters values for a model according to what can be inferred from the data. From previous knowledge of the combustion process it is expected that not only the main predictors but also the interactions between predictors have an effect on the sensor reading. As a general rule, if the data contains K variables then the expected number of possible models will be 2^K. The total number of variables in the dataset with all the interactions included accounted to 22, that is 4.19×10^3 possible model combinations to describe the sensor reading. To reduce the dimensionality of the problem *Bayesian model averaging* (BMA) is implemented. This provides a way to account for the uncertainty in model selection and provide in average a better predictive ability [5]. BMA is implemented in the statistical programing language R using the Bayesian Model Sampling (BMS) package [16]. The results show that the predictors X1, X3, X4, X1:X4, X3:X4 and X2 seem to be the most important for a good model.

As the first taken approach the reduced model containing only 6 out of the 22 variables is defined using a linear relationship. The model was implemented using *Just Another Gibbs Sampler* (JAGS), which is a program for Bayesian modeling using *Markov Chain Monte Carlo* (MCMC) [11] and rjags [12] as a link between R and JAGS.

The sensor responses $Y1$ and $Y2$ are modeled following a non standardized Student's t-distribution. This distribution was selected assuming that the variance present in the sensor output, illustrated in Fig. 1, served as an indicator of variance in the model response. The mean of the distribution is defined by the canonical linear formula of the linear regression. The spread of the data was set to have a wide range of probable values defined by an uniform distribution. The normality factor, expressed as an exponential distribution, have preference for values close to one. Given the limited prior information available for the experiment, weakly informative priors are assigned to the parameters. The coefficients priors, β_i, are defined to have a normal distribution cen-

tered around zero and a large variance. The t-distribution normality factor υ favors values smaller than 30 and σ allows for a wide enough distribution. The prior distributions were chosen as follows (Eq. 3):

$$\alpha \sim N(0,4), \quad \beta_i \sim N(0,4), \quad \upsilon \sim Exp(30), \quad \sigma \sim U(-1^{-4},10) \quad (3)$$

The MCMC simulation are executed on the defined model to sample the posterior distribution of the parameters of interest, α, β_i, σ, and υ. The chains were specified to run 500 adaptive iterations, followed by 1,500 burn-in iterations. Afterwards, 15,000 samples were taken from the posterior distribution with a thinning factor of 20 steps. Investigating their trace plots and diagnostic statistics of the resulting MCMC object reveals that the chains have converged. A value for the Gelman-Rubin diagnostic statistic [4] of under 1.1 suggest a good convergence. The effective sampling size (ESS) backups this assumption. The visual and numeric diagnostics allows us to think that the resulting MCMC sampling is representative and accurate of the posterior distribution of the different parameters. The posterior distributions obtained from the MCMC sampling for each parameter coefficient can be seen in Table 3 together with the *high density intervals* (HDI) of 95%. The MSE for the fitted models is 0.16 and 0.15 for the sensor $Y1$ and $Y2$, respectively.

Table 3: Posterior mean for the coefficients β_i for $i = 1, ..7$ and the parameters σ and υ for the models of Y1 and Y2. The lower HDI (L-HDI) and upper HDI (U-HDI) limits are indicated for each entry.

					Y1				
	B0	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	σ	υ
Mean	-0.01	0.32	-0.14	-0.28	0.72	-0.23	-0.14	0.41	37.67
L-HDI	-0.09	0.23	-0.24	-0.38	0.63	-0.33	-0.42	0.34	5.13
U-HDI	0.09	0.42	-0.04	-0.18	0.82	-0.14	-0.05	0.49	117 .66
					Y2				
	B0	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	σ	υ
Mean	-0.00	0.29	-0.10	-0.20	0.78	-0.26	-0.13	0.39	34.2
L-HDI	-0.09	0.20	-0.19	-0.29	0.69	-0.35	-0.22	0.31	4.44
U-HDI	0.09	0.39	-0.10	-0.11	0.88	-0.17	-0.04	0.47	113.50

3.5 Genetic Programming

Genetic programming is an evolutionary algorithm that searches the set of symbolic expressions defined by a set of basis expressions (building

blocks) for expressions that minimize one or multiple loss (fitness) functions. Symbolic regression is the application of genetic programming to regression. In this work, the *Generational Multi-Objective Genetic Programming* (GMOGP) symbolic regression algorithm is used. See [3] for a detailed description of this algorithm.

In this specific case, the set of building blocks $B := \mathbb{R} \cup V \cup F$ consists of the set of real-valued constants $\mathbb{R}$, the set of independent variables $V := \{x_1, x_2, \ldots, x_7\}$ and the set of real-valued functions $F := \{+, -, \times, \div, \sqrt{}, \log, \exp, \sin, \cos\}$.

The GMOGP algorithm performs the following four steps:

1) The algorithm proceeds by initializing a population of $\mu := 100$ random symbolic expressions with maximum node count of 128 based on the building blocks in B.

2) Next, $\lambda := 50$ expressions are created by random recombination and mutation of randomly selected population expressions, together with $\nu := 2$ randomly initialized expressions.

3) The complexity (expression visitation length) and goodness-of-fit (scaled-mean-squared error on training data) for each expression is calculated. The expressions are then sorted into successive Pareto fronts according to both criteria. Solutions on the same front are sorted by crowding distance.

4) If a predefined termination criterion is met (in this case an iteration limit of $300,000$) the algorithm terminates and returns the first Pareto front. Otherwise, the the best $\mu := 100$ expressions are kept and the algorithms enters the next iteration (generation) at step 2).

As symbolic regression is a randomized algorithm, the implementation uses multiple parallel runs to reduce result variance. The first Pareto front based on all parallel runs is returned as the final result. In this case, five parallel runs of $300,000$ generations each were used, each requiring two minutes of single-core compute time on a 1.8 GHz Intel Core i7 processor, or ten minutes compute time in total. The used GP implementation selects model constants by sampling from a uniform random distribution and optimizes existing constants via mutation by adding samples from a normal random distribution. Crossover operations may lead to the duplication of constants, as seen in GP model 2 in Table 4. To facilitate model comparisons, only the model with best goodness-of-fit on training data is selected from the result Pareto front and reported as the final result. Note, these results numbers are based on the commercial parallel GMOGP-FCA implementation *sourcewerk RSR*.

Table 4: Models used in this study. Model formulas and the corresponding MSE values are shown. The first model, f^1, is included as a baseline. It implements a first order linear model with all effects, but no interactions or higher order terms. Best values are shown in boldface.

j	Name	Model sensor Y1	MSE_1
1	LM (base)	$f_1^1 = 0 + 0.35x_1 - 0.17x_2 - 0.26x_3 + 0.74x_4$ $+0.01x_5 - 0.03x_6 + 0.03x_7$	0.76
2	OLS	$f_1^2 = -0.01 + 0.33x_1 - 0.14x_2 - 0.29x_3$ $+0.73x_4 - 0.23x_1x_4 - 0.15x_3x_4$	0.79
3	Lasso	$f_1^3 = 0.25x_1 - 0.05x_2 - 0.17x_3 + 0.63x_4$	**0.56**
4	Kriging	$f_1^4 : \theta =$ $(0.57, 2.59, 6.08, 3.45, 1.53, 18.53, 18.46)$	0.57
5	Bayes	$f_1^5 = dt(0.32x_1 - 0.14x_2 - 0.28x_3 + 0.72x_4$ $-0.23x_1x_4 - 0.14x_3x_4, 0.41, 37)$	0.79
6	GP	$f_1^6 = -0.02 + 0.31x_1 + (0.64 - 0.12x_1)x_4$	0.58
j	Name	Model sensor Y2	MSE_2
1	LM (base)	$f_2^1 = 0 + 0.31x_1 - 0.07x_2 - 0.19x_3 + 0.78x_4$ $+0.01x_5 - 0.04x_6 + 0.04x_7$	0.67
2	OLS	$f_2^2 = -0.01 + 0.30x_1 - 0.10x_2 - 0.20x_3$ $+0.78x_4 - 0.26x_1x_4 - 0.14x_3x_4$	0.80
3	Lasso	$f_2^3 = 0.22x_1 - 0.08x_3 + 0.68x_4 - 0.02x_7$	0.49
4	Kriging	$f_2^4 : \theta =$ $(0.43, 2.41, 17.53, 4.21, 1.30, 19.96, 19.15)$	0.49
5	Bayes	$f_2^5 = dt(0.29x_1 - 0.1x_2 - 0.2x_3 + 0.78x_4$ $-0, 26x_1x_4 - 0.13x_3x_4, 0.39, 34.2)$	0.79
6	GP	$f_2^6 = 0.52 - 0.21x_3 + \frac{0.21(-1.43+x_4)}{1.43+x_1} + 0.42x_4$ $-0.21\cos(x_3 - x_4) + 0.21\cos(x_7)$	**0.27**

4. Results

The comparison is based on the MSE. We consider six different models ($j = 1, \ldots, 6$) and two different data sets ($k = 1, 2$) from Table 4.

Let f_k^j denote the function, which models the relationship between y_{train} and x_{train} ($j = 1, \ldots, 4,\ k = 1, 2$). Let $\hat{y} = f_k^j(x_{\text{test}})$ denote a vector of n predictions on the test data x_{test}, and y_{test} denote the vector of observed (true) values, then the MSE can be estimated by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y_{test}^{(i)})^2 \text{ with } i = 1, \ldots, n. \tag{4}$$

5. Conclusion

Interestingly, the lightweight and simplistic Lasso approach and the heavy weighted, sophisticated genetic programming approach obtained the best results. Lasso performed best on $Y1$, whereas genetic programming was able to find the best MSE on $Y2$. However, given the complexity of the genetic programming model, it is hard to see the real effect each variable has on the sensor output. Lasso on the other side, gives a clear and simple overview of the variables effects while keeping the prediction error low. The Kriging model demonstrated a relatively good performance but, the interpretation of the results is less intuitive compared to the Lasso results.

In this particular application the models need to be adaptable to allow its use on other similar sensors. A model of high complexity together with a difficult interpretability constitutes an extra effort. Bayesian and linear regression approaches generated similar formulas. This was expected as the definition of the Bayesian model was done following the canonical linear formula. Both basic linear models (LM and OLS) and Bayesian model present easily interpretable results but with poor MSE values. This implies that, although new information is available when implemented the Bayesian modeling method, there is no significant difference when compared to the standard linear regression. We expect however to obtain further improvements of the model prediction accuracy by means of more complex and specific model definition for the Bayesian case.

In addition to the six algorithms discussed in this study, further algorithms were tested. For example, a standard random forest algorithm was able to obtain results that are comparable to the Kriging results (MSE's: 0.63 and 0.41 on $Y1$ and $Y2$, respectively). However, due to space limitations, these extended results will be analyzed in a forthcoming publication. Results, presented in this study, are limited to models that are of great practical relevance, i.e., can be immediately interpreted by and discussed with technicians and engineers.

At the end and for this specific application, the Lasso model was our preferred method. A simple and clear formula provides valuable starting points for the discussion with the engineers in charge of the project and makes for a straightforward implementation.

Acknowledgement: This report project was promoted by the Federal Ministry of Economy and Energy under the project funding reference number KF3145101WM3 and KF3145103WM4 . Responsibility for the content of this publication lies with the author.

References

[1] A. Bejaoui, J. Guerin, and K. Aguir. Modeling of a p-type resistive gas sensor in the presence of a reducing gas. *Sensors and Actuators B: Chemical*, 181:340–347, 2013.

[2] T. Eklöv, P. Martensson, and I. Lundström. Selection of variables for interpreting multivariate gas sensor data. *Analitycal Chimica Acta*, 381(2-3):221–232, 1999.

[3] O. Flasch. A Modular Genetic Programming System. Dissertation, TU Dortmund, 2015.

[4] A. Gelman and D. B. Rubin. Inference fron iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–511, 1992.

[5] J. A. Hoeting, D. Madigan, A.E. Raftery, and C.T. Volisnky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–417, 1999.

[6] A. Jafarian, D. Baleanu, H. Darwish, M. Senel, and S. Okur. Applications of Artificial Neural Network Technique to Polypyrrole Gas Sensor Data for Environmental Analysis. *Computational and Theoretical Nanoscience*, 12:1–7, 2015.

[7] JMP. *Design of Experiments Guide*. 2009.

[8] J. K. Kruschke. *Doing Bayesian Data Analysis: a tutorial with R, JAGS and Stan*. Academic Press, 2nd edition, 2014.

[9] J. K. Kruschke, H. Aguinis, and H. Joo. The time has come: Bayesian methods for data analysis in the organizational sciences. *Organizational Research Methods*, 15(4):722–752, 2012.

[10] D. C. Montgomery. *Design and Analysis of Experiments*. Wiley, 5th ed., 2001.

[11] M. Plummer. *Jags: A program for analysis of Bayesian graphical models using gibbs sampling*, 2003.

[12] M. Plummer. *rjags: Bayesian Graphical Models using MCMC*, 2015. R package version 4-4.

[13] F. Pukelsheim. *Optimal Design of Experiments*. Wiley, 1993.

[14] J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.

[15] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B (Methodolgical*, 58(1):267–288, 1996.

[16] S. Zeugner. *Bayesian Model Averaging with BMS*, 2011. R package version 0.3.4.