

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

Martin Middendorf Christian Blum (Eds.)

Evolutionary Computation in Combinatorial Optimization

13th European Conference, EvoCOP 2013

Vienna, Austria, April 3-5, 2013

Proceedings



Springer

Volume Editors

Martin Middendorf
University of Leipzig
Department of Computer Science
Johannisgasse 26, 04103 Leipzig, Germany
E-mail: middendorf@informatik.uni-leipzig.de

Christian Blum
University of the Basque Country
IKERBASQUE, Basque Foundation for Science
Department of Computer Science and Artificial Intelligence
Paseo Manuel Lardizabal 1, 20018 Donostia, Spain
E-mail: christian.c.blum@gmail.com

Front cover EvoStar 2013 logo by Kevin Sim, Edinburgh Napier University

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-37197-4 e-ISBN 978-3-642-37198-1
DOI 10.1007/978-3-642-37198-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013933104

CR Subject Classification (1998): G.1.6, F.2, G.2, F.1, I.2.8

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

© Springer-Verlag Berlin Heidelberg 2013

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

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

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

Printed on acid-free paper

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

Preface

Algorithms from the field of metaheuristics have been shown to be provenly effective for an ever-broadening range of difficult combinatorial optimization problems arising in a wide variety of industrial, economic, and scientific domains. Prominent examples of metaheuristics include, but are not limited to, ant colony optimization, evolutionary algorithms, greedy randomized adaptive search procedures, iterated local search, simulated annealing, tabu search, and variable neighborhood search. Applications of metaheuristics can be found in a wide variety of fields such as scheduling, timetabling, network design, transportation and distribution, vehicle routing, the travelling salesman problem, packing and cutting, satisfiability, and general mixed integer programming.

The series of EvoCOP workshops/conferences was initiated in 2001 and has been held annually since then. In fact, EvoCOP is the first event specifically dedicated to the application of evolutionary computation and related methods to combinatorial optimization problems. Originally held as a workshop, EvoCOP became a conference in 2004. Past events gave researchers an excellent opportunity to present their latest research and to discuss current developments and applications. Following the general trend of hybrid metaheuristics and diminishing boundaries between different classes of metaheuristics, EvoCOP has broadened its scope in recent years and has solicited papers on any kind of metaheuristic for combinatorial optimization.

This volume contains the proceedings of EvoCOP 2013, the 13th European Conference on Evolutionary Computation in Combinatorial Optimization. It was held in Vienna, Austria, during April 3–5, 2013, jointly with EuroGP 2013, the 16th European Conference on Genetic Programming, EvoBIO 2013, the 11th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, EvoMUSART 2013, the Second International Conference and 11th European Event on Evolutionary and Biologically Inspired Music, Sound, Art and Design, and EvoApplications 2013 (formerly EvoWorkshops), which consisted of 12 individual tracks ranging from complex systems over games to risk management. From 2007, all these events are grouped under the collective name EvoStar, and constitute Europe's premier co-located event on evolutionary computation and metaheuristics.

Accepted papers of previous EvoCOP editions were published by Springer in the series *Lecture Notes in Computer Science* (LNCS – Volumes 2037, 2279, 2611, 3004, 3448, 3906, 4446, 4972, 5482, 6022, 6622, 7245). Statistics for each conference are as follows:

EvoCOP	Submitted	Accepted	Acceptance Ratio
2001	31	23	74.2%
2002	32	18	56.3%
2003	39	19	48.7%
2004	86	23	26.7%
2005	66	24	36.4%
2006	77	24	31.2%
2007	81	21	25.9%
2008	69	24	34.8%
2009	53	21	39.6%
2010	69	24	34.8%
2011	42	22	52.4%
2012	48	22	45.8%
2013	50	23	46.0%

The rigorous, double-blind reviewing process of EvoCOP 2013 resulted in the selection of 23 out of 50 submitted papers; the acceptance rate was 46.0%. It is worth pointing out that the number of submissions was higher than at last year's event. Given the current times of crisis, this is a remarkable achievement. Each paper was reviewed by a sufficient number of members of the international Program Committee. In fact, their work is essential for the continuing success of EvoCOP. Moreover, acceptance/rejection decisions were not only based on the received referee reports but also on a personal evaluation of the Program Chairs. At this point we would like to thank all authors for submitting their work to this EvoCOP edition.

There are various persons and institutions that have contributed to the success of the conference and to whom we would like to express our appreciation. First of all, we thank the local organizers of EvoStar 2013, Günther R. Raidl, Bin Hu, and Doris Dicklberger, from the Vienna University of Technology. They did an extraordinary job. Furthermore, we would like to thank Marc Schoenauer from INRIA (France) for his continuing support concerning the MyReview conference management system. We also thank Kevin Sim from Edinburgh Napier University and A. Şima Etaner-Uyar from the Istanbul Technical University for the excellent website and publicity material. Thanks are also due to Jennifer Willies and the Institute for Informatics and Digital Innovation at Napier University in Edinburgh, Scotland, for administrative support and event coordination. Finally, we gratefully acknowledge the Vienna University of Technology for its support of EvoStar.

Last, but not least, we would like to thank Carlos Cotta, Peter Cowling, Jens Gottlieb, Jin-Kao Hao, Jano van Hemert, Peter Merz, and Günther R. Raidl for their hard work and dedication at past editions of EvoCOP, which contributed to making this conference one of the reference events in evolutionary computation and metaheuristics.

Organization

EvoCOP 2013 was organized jointly with EuroGP 2013, EvoBIO 2013, EvoMUSART 2013, and EvoApplications 2013.

Organizing Committee

Program Committee Chairs

Martin Middendorf	University of Leipzig, Leipzig, Germany
Christian Blum	IKERBASQUE, Basque Foundation for Science, and University of the Basque Country, San Sebastian, Spain

Local Organization

Günther R. Raidl	Vienna University of Technology, Austria
Bin Hu	Vienna University of Technology, Austria
Doris Dicklberger	Vienna University of Technology, Austria

Publicity Chair

A. Şima Etaner-Uyar	Istanbul Technical University, Turkey
---------------------	---------------------------------------

EvoCOP Steering Committee

Carlos Cotta	Universidad de Málaga, Spain
Peter Cowling	University of York, UK
Jens Gottlieb	SAP AG, Germany
Jin-Kao Hao	University of Angers, France
Jano van Hemert	University of Edinburgh, UK
Peter Merz	University of Applied Sciences and Arts, Hannover, Germany
Günther Raidl	Vienna University of Technology, Austria

Program Committee

Adnan Acan	Eastern Mediterranean University, Gazimagusa, Turkey
Hernán Aguirre	Shinshu University, Nagano, Japan
Enrique Alba	Universidad de Málaga, Spain
Mehmet Emin Aydin	University of Bedfordshire, UK

VIII Organization

Ruibin Bai	University of Nottingham, UK
Thomas Bartz-Beielstein	Cologne University of Applied Sciences, Germany
Maria Blesa	Universitat Politècnica de Catalunya, Spain
Christian Blum	IKERBASQUE and University of the Basque Country, Spain
Rafael Caballero	University of Málaga, Spain
Alexandre Caminada	UTBM, France
Pedro Castillo	Universidad de Granada, Spain
José Francisco Chicano Garcia	Universidad de Málaga, Spain
Carlos Coello Coello	CINVESTAV-IPN, Mexico
Peter Cowling	University of York, UK
Keshav Dahal	University of Bradford, UK
Karl Doerner	Universität Wien, Austria
Benjamin Doerr	Max-Planck-Institut für Informatik, Germany
Anton V. Eremeev	Omsk Branch of Sobolev Institute of Mathematics, Russia
Antonio J. Fernández	Universidad de Málaga, Spain
Francisco Fernández de Vega	University of Extremadura, Spain
Bernd Freisleben	University of Marburg, Germany
Philippe Galinier	Ecole Polytechnique de Montreal, Canada
Jens Gottlieb	SAP, Germany
Walter Gutjahr	University of Vienna, Austria
Jin-Kao Hao	University of Angers, France
Geir Hasle	SINTEF Applied Mathematics, Norway
István Juhos	University of Szeged, Hungary
Graham Kendall	University of Nottingham, UK
Joshua Knowles	University of Manchester, UK
Mario Köppen	Kyushu Institute of Technology, Japan
Jozef Kratica	University of Belgrade, Serbia
Rhyd Lewis	Cardiff University, UK
Arnaud Liefoghe	Université des Sciences et Technologies de Lille, France
Arne Løkketangen	Molde College, Norway
José Antonio Lozano	University of the Basque Country, Spain
Zhipeng Lu	HUST, China
Penousal Machado	University of Coimbra, Portugal
Dirk C. Mattfeld	University of Braunschweig, Germany
Barry McCollum	Queen's University Belfast, UK
Juan Julián Merelo	University of Granada, Spain
Peter Merz	University of Applied Sciences and Arts, Hannover, Germany
Martin Middendorf	Universität Leipzig, Germany
Julian Molina	University of Málaga, Spain

Pablo Moscato	The University of Newcastle, Australia
Christine L. Mumford	Cardiff University, UK
Nysret Musliu	Vienna University of Technology, Austria
Yuichi Nagata	Tokyo Institute of Technology, Japan
Giuseppe Nicosia	University of Catania, Italy
Mario Pavone	University of Catania, Italy
Francisco J. B. Pereira	Universidade de Coimbra, Portugal
Daniel Cosmin Porumbel	University of Artois, France
Jakob Puchinger	Arsenal Research, Vienna, Austria
Günther Raidl	Vienna University of Technology, Austria
Marcus Randall	Bond University, Queensland, Australia
Marc Reimann	Warwick Business School, UK
Andrea Roli	Università degli Studi di Bologna, Italy
Eduardo Rodriguez-Tello	CINVESTAV-Tamaulipas, Mexico
Patrick Siarry	Université Paris-Est Créteil Val-de-Marne, France
Jim Smith	University of the West of England, UK
Giovanni Squillero	Politecnico di Torino, Italy
Thomas Stützel	Université Libre de Bruxelles, Belgium
El-ghazali Talbi	Université des Sciences et Technologies de Lille, France
Kay Chen Tan	National University of Singapore, Singapore
Jorge Tavares	MIT, USA
Jano van Hemert	University of Edinburgh, UK
Sebastien Verel	Université de Nice Sophia Antipolis, France
Takeshi Yamada	NTT Communication Science Laboratories, Japan
Shengxiang Yang	De Montfort University, UK

Additional Referees

Pablo García-Sánchez	Antonio M. Mora
Jean-Philippe Hamiez	Mariela Nogueira
Raúl Lara-Cabrera	Martin Zaefferer

Table of Contents

A Hyper-heuristic with a Round Robin Neighbourhood Selection	1
<i>Ahmed Kheiri and Ender Özcan</i>	
A Multiobjective Approach Based on the Law of Gravity and Mass Interactions for Optimizing Networks	13
<i>Álvaro Rubio-Largo and Miguel A. Vega-Rodríguez</i>	
A Multi-objective Feature Selection Approach Based on Binary PSO and Rough Set Theory	25
<i>Liam Cervante, Bing Xue, Lin Shang, and Mengjie Zhang</i>	
A New Crossover for Solving Constraint Satisfaction Problems	37
<i>Reza Abbasian and Malek Mouhoub</i>	
A Population-Based Strategic Oscillation Algorithm for Linear Ordering Problem with Cumulative Costs	49
<i>Wei Xiao, Wenqing Chu, Zhipeng Lü, Tao Ye, Guang Liu, and Shanshan Cui</i>	
A Study of Adaptive Perturbation Strategy for Iterated Local Search . . .	61
<i>Una Benlic and Jin-Kao Hao</i>	
Adaptive MOEA/D for QoS-Based Web Service Composition	73
<i>Mihai Suciú, Denis Pallez, Marcel Cremene, and Dumitru Dumitrescu</i>	
An Analysis of Local Search for the Bi-objective Bidimensional Knapsack Problem	85
<i>Leonardo C.T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle</i>	
An Artificial Immune System Based Approach for Solving the Nurse Re-rostering Problem	97
<i>Broos Maenhout and Mario Vanhoucke</i>	
Automatic Algorithm Selection for the Quadratic Assignment Problem Using Fitness Landscape Analysis	109
<i>Erik Pitzer, Andreas Beham, and Michael Affenzeller</i>	
Balancing Bicycle Sharing Systems: A Variable Neighborhood Search Approach	121
<i>Marian Rainer-Harbach, Petrina Papazek, Bin Hu, and Günther R. Raidl</i>	

Combinatorial Neighborhood Topology Particle Swarm Optimization Algorithm for the Vehicle Routing Problem	133
<i>Yannis Marinakis and Magdalene Marinaki</i>	
Dynamic Evolutionary Membrane Algorithm in Dynamic Environments	145
<i>Chuang Liu and Min Han</i>	
From Sequential to Parallel Local Search for SAT	157
<i>Alejandro Arbelaez and Philippe Codognet</i>	
Generalizing Hyper-heuristics via Apprenticeship Learning	169
<i>Shahriar Asta, Ender Özcan, Andrew J. Parkes, and A. Şima Etaner-Uyar</i>	
High-Order Sequence Entropies for Measuring Population Diversity in the Traveling Salesman Problem	179
<i>Yuichi Nagata and Isao Ono</i>	
Investigating Monte-Carlo Methods on the Weak Schur Problem	191
<i>Shalom Eliahou, Cyril Fonlupt, Jean Fromentin, Virginie Marion-Poty, Denis Robilliard, and Fabien Teytaud</i>	
Multi-objective AI Planning: Comparing Aggregation and Pareto Approaches	202
<i>Mostepha R. Khouadjia, Marc Schoenauer, Vincent Vidal, Johann Dréo, and Pierre Savéant</i>	
Predicting Genetic Algorithm Performance on the Vehicle Routing Problem Using Information Theoretic Landscape Measures	214
<i>Mario Ventresca, Beatrice Ombuki-Berman, and Andrew Runka</i>	
Single Line Train Scheduling with ACO	226
<i>Marc Reimann and Jose Eugenio Leal</i>	
Solving Clique Covering in Very Large Sparse Random Graphs by a Technique Based on k-Fixed Coloring Tabu Search	238
<i>David Chalupa</i>	
Solving the Virtual Network Mapping Problem with Construction Heuristics, Local Search and Variable Neighborhood Descent	250
<i>Johannes Inführ and Günther R. Raidl</i>	
The Generate-and-Solve Framework Revisited: Generating by Simulated Annealing	262
<i>Rommel D. Saraiva, Napoleão V. Nepomuceno, and Plácido R. Pinheiro</i>	
Author Index	275

A Hyper-heuristic with a Round Robin Neighbourhood Selection

Ahmed Kheiri and Ender Özcan

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{axk,exo}@cs.nott.ac.uk

Abstract. An iterative selection hyper-heuristic passes a solution through a heuristic selection process to decide on a heuristic to apply from a fixed set of low level heuristics and then a move acceptance process to accept or reject the newly created solution at each step. In this study, we introduce Robinhood hyper-heuristic whose heuristic selection component allocates equal share from the overall execution time for each low level heuristic, while the move acceptance component enables partial restarts when the search process stagnates. The proposed hyper-heuristic is implemented as an extension to a public software used for benchmarking of hyper-heuristics, namely HyFlex. The empirical results indicate that Robinhood hyper-heuristic is a simple, yet powerful and general multistage algorithm performing better than most of the previously proposed selection hyper-heuristics across six different Hyflex problem domains.

1 Introduction

A *hyper-heuristic* is a heuristic that performs a search over a space of heuristics, as opposed to space of solutions directly. Although the term hyper-heuristic was coined recently, the idea of combining the strengths of different heuristics (neighbourhood operators) dates back to the 1960s [10]. An aim of the hyper-heuristic research is to raise the level of generality by providing a high level strategy that is applicable across different problem domains rather than a single one. There are two main types of hyper-heuristics in the academic literature: methodologies used for *generation* or *selection* of heuristics [5,6,23].

A selection hyper-heuristic methodology combines heuristic selection and move acceptance processes under a single point search framework [1,2,8,20,21]. A candidate solution is improved iteratively by selecting, applying a heuristic (neighbourhood operator) from a set of low level heuristics, then passing the new solution through move acceptance to replace the solution in hand at each step. [5,7] are the recent surveys on hyper-heuristics. In this study, a new simple selection hyper-heuristic is introduced and tested across a variety of problem domains using Hyflex (Hyper-heuristics Flexible framework) [18], a software tool for hyper-heuristic development and research.

1.1 Hyflex

Hyflex provides an interface for the implementation of not only hyper-heuristics but also other (meta)heuristics and problem domains. Any problem domain developed for Hyflex is required to define a set of low level heuristics (neighbourhood operators) which should be classified as *mutational* (MU), *hill climbing* (HC), *ruin and re-create* (RC) or *crossover* (XO). A mutational heuristic makes a random perturbation producing a new solution and this process does not necessarily generate an improvement over the input solution. Local search or hill climbing is often an iterative procedure searching different neighbourhoods starting from a given solution. A ruin and re-create operator produces a partial solution from a given complete solution and then rebuilds a new complete solution. Crossover is a well known operator in evolutionary computation, which takes two solutions and produces a new solution. In general, crossover yields two new solutions and the best new solution is returned in Hyflex.

HyFlex provides utilities to control the behaviour of some low level heuristics to a limited extent. It is possible to increase or decrease the intensity of some mutational and ruin and re-create operations by adjusting a control parameter from 0.0 to 1.0. Changing the value of the intensity parameter could mean changing the range of new values that a variable can take in relation to its current range of values or changing the number of solution variables that will be processed by a heuristic. There is also another similar control parameter for some local search operators for changing the depth of search which relates to the number of hill climbing steps.

HyFlex currently provides an implementation of six minimisation problem domains: Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Permutation Flow Shop (PFS), Personnel Scheduling (PS), Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) each with different instances and a set of low-level heuristics. The nature of each low level heuristic for each Hyflex problem domain is summarised in Table 1. Currently, there are 12 different instances for the first four problem domains and 10 for the last two problem domains. What is left is to design and implement a high-level strategy (hyper-heuristic) that intelligently selects and applies suitable low-level heuristics from the set provided to each instant from the given domain to get the minimum objective function value in ten minutes. Hyflex was used for Cross-domain Heuristic Search Challenge (CHESC)¹ in 2011. The hyper-heuristics entered into this competition and made it to the finals serve as a benchmark for newly developed hyper-heuristics targeting generality.

2 Related Work

There is a growing number of work on selection hyper-heuristics which have been designed and tested using hyflex. Before CHESC 2011 (Cross-Domain Heuristic

¹ <http://www.asap.cs.nott.ac.uk/chesc2011/>

Table 1. The nature of the low level heuristics used in each problem domain. The bold entries for each problem domain mark the last low level heuristic of each type.

Heuristic IDs	LLH0	LLH1	LLH2	LLH3	LLH4	LLH5	LLH6	LLH7
MAX-SAT	MU ₀	MU ₁	MU ₂	MU ₃	MU ₄	MU₅	RC₀	HC ₀
Bin Packing	MU ₀	RC ₀	RC₁	MU ₁	HC ₀	MU₂	HC₁	XO₀
PS	HC ₀	HC ₁	HC ₂	HC ₃	HC₄	RC ₀	RC ₁	RC₂
PFS	MU ₀	MU ₁	MU ₂	MU ₃	MU₄	RC ₀	RC₁	HC ₀
TSP	MU ₀	MU ₁	MU ₂	MU ₃	MU₄	RC₀	HC ₀	HC ₁
VRP	MU ₀	MU ₁	RC ₀	RC₁	HC ₀	XO ₀	XO₁	MU₂
Heuristic IDs	LLH8	LLH9	LLH10	LLH11	LLH12	LLH13	LLH14	
MAX-SAT	HC₁	XO ₀	XO₁					
PS	XO ₀	XO ₁	XO₂	MU ₀				
PFS	HC ₁	HC ₂	HC₃	XO ₀	XO ₁	XO ₂	XO₃	
TSP	HC₂	XO ₀	XO ₁	XO ₂	XO₃			
VRP	HC ₁	HC₂						

Search Challenge), a mock competition was organised with hyflex and the performance of several well known previously proposed hyper-heuristics were compared across a subset of CHESC problem domains. Burke et al. [3] reported that the best performing hyper-heuristic was an iterated local search approach which applied a randomly selected mutational and ruin and re-create heuristic and then the hill climbers in a predefined sequence. This framework is based on the most successful hyper-heuristic framework reported to perform the best in [21]. Özcan and Kheiri [22] provide a greedy heuristic selection strategy named *dominance-based heuristic selection* which aims to determine low level heuristics with good performance based on the trade-off between the change (improvement) in the solution quality and the number of steps taken. The approach attempts to reduce the number of low level heuristics. It performs well with respect to the mock competition hyper-heuristics on four problem domains of HyFlex. Nguyen et al. [17] tested an evolutionary approach to generate hyper-heuristics across three HyFlex problem domains and according to the experimental results, they obtained only one improving solution over the top two hyper-heuristics from the mock competition.

In the mock competition, each algorithm was run once for each instance, while 31 runs were performed in CHESC and the median results were compared to determine the best performing hyper-heuristic among the CHESC participants generalising well across all six (four public and two hidden) HyFlex problem domains given 10 minutes of execution time per instance. The algorithm description of the approaches developed by CHESC competitors are provided in (<http://www.asap.cs.nott.ac.uk/external/chesc2011/results.html>). The winner of the competition, denoted as *AdapHH* was a hyper-heuristic which combines a learning adaptive heuristic selection method with an adaptive iteration limited list-based threshold move accepting method ([15]). This hyper-heuristic does not make use of the type information provided for the low level heuristics.

A hyper-heuristic based on Variable Neighborhood Search (*VNS-TW*) ranked the second at the competition [11]. This approach applies shaking heuristics followed by hill-climbers to solve the problems. The third ranking algorithm (*ML*) was based on a self-adaptive meta-heuristic using multiple agents. The fourth approach (*PHUNTER*) was inspired by the pearl hunting utilising two phases *diversification* and *intensification*. The fifth hyper-heuristic (*EPH*) was based on evolutionary programming which evolves population of solutions and heuristics. The other hyper-heuristics were inspired from well known population based and single point-based search metaheuristics: Iterated Local Search driven by Evolutionary Algorithms (*ISEA*) [14], Hybrid Adaptive Evolutionary Algorithm (*HAEA*), Genetic Hive Hyper-heuristic (*GenHive*), Ant Colony Optimization based hyper-heuristic (*ACO*), Simulated Annealing Hyper-Heuristic with Reinforcement Learning and Tabu-Search (*KSATS-HH*), Reinforcement Learning (*AVEG-Nep*) [9] and Generic Iterative Simulated-Annealing Search (*GISS*). More on these approaches can be found at the competition webpage.

After the CHeSC 2011 competition, a number of researchers attempted to improve previously proposed hyper-heuristics. Kalender et al. [13] proposed a hyper-heuristic which combines a learning greedy gradient approach for heuristic selection and simulated annealing move acceptance. The results show that this approach performs slightly better than a Choice Function hyper-heuristic whose performance is improved by Drake et al. [12] substantially with a minor adjustment. Although, these approaches improved the performance of the traditional Choice Function and Greedy hyper-heuristics on HyFlex problem domains, their performances still on average as compared to the hyper-heuristics of CHeSC competitors. In [4,19], the authors proposed an adaptive neighbourhood iterated local search algorithm based on Hyflex and its variant.

The proposed hyper-heuristic in this study is also implemented as an extension to HyFlex. Its performance is compared to the mock competition hyper-heuristics as well as hyper-heuristics provided by the CHeSC competitors.

3 Methodology

This study introduces a multi-stage selection hyper-heuristic framework based on a **round-robin neighbourhood** selection mechanism (Algorithm 1). This framework gives a chance for each low level heuristic in a selected subset of low level heuristics to execute for a certain duration at a stage. A low level heuristic is chosen in a round robin fashion. Depending on the strategy whole set of low level heuristics can be used and the order of low level heuristics can be fixed or varied. Any move acceptance method could be used within this framework. We describe an easy-to-implement yet powerful selection hyper-heuristic based on this framework which will be referred to as *Robinhood hyper-heuristic* in this section. The Robinhood hyper-heuristic is implemented for Hyflex accommodating performance testing across domain implementations and comparison to the top hyper-heuristics competed in CHeSC.

The Robinhood hyper-heuristic is composed of components inspired from previously proposed approaches. The heuristic selection methods presented by

Algorithm 1. Robinhood hyper-heuristic framework

```

1: procedure ROBINHOOD
2:   initialise();
3:   while (terminationCriteriaNotSatisfied1()) do    ▷ e.g., terminate when the
   given overall execution time is exceeded
4:     update1(); ▷ set/update relevant parameter/variable values before entering
   into a stage or no-op
5:     for ( $i = \text{nextLowLevelHeuristicID}()$ ) do    ▷ entry of the stage
6:       while ( terminationCriteriaNotSatisfied2() ) do ▷ e.g., terminate when
   the given time for a heuristic is exceeded
7:          $S' = \text{applyLLH}(i, S)$ ; ▷  $S$  and  $S'$  are the current and new solutions,
   respectively
8:         moveAcceptance( $S, S'$ );
9:       end while
10:      update2();    ▷ set/update relevant parameter/variable values after
   employing a low level heuristic or no-op
11:     end for
12:     update3();    ▷ set/update relevant parameter/variable values after a stage
   or no-op
13:   end while
14: end procedure

```

Cowling et al. [8] includes *Random Permutation* and *Random Permutation Gradient*. This method applies a low level heuristic one at a time sequentially in a randomly generated permutation order. Random Permutation Gradient operates in the same with a minor change that is as long as the chosen heuristic makes an improvement in the current solution the same heuristic is employed. Given a time limit of t (Algorithm 1, line 3), and n low level heuristics, the Robinhood hyper-heuristic fixes the number of stages as k and applies all low level heuristics (line 5) to the current solution in a given order for $t/(n.k)$ time unit at a stage (line 6). Hyflex does not provide any annotation for the low level heuristics in a given domain, indicating whether they operate on a given solution with a stochastic or deterministic approach. Although this could be detected with a level of certainty using some initial tests over the set of heuristics, we assumed that all operators are stochastic and so each heuristic is given an equal amount of time to process a given solution at each stage.

We classified all ruin and re-create low level heuristics provided in a given problem domain as mutational heuristics, since Hyflex does not provide any indication whether a ruin and re-create heuristic is a mutational or local search operator. The proposed hyper-heuristic aims to use all low level heuristics assuming that the domain implementers chose reasonable heuristics which will not be misleading for the search process. Consequently, we randomly order the low level heuristics within each group of heuristics: mutational, crossover and hill climbing. Inspired by memetic algorithms [16,24], in which solutions are

improved through successive application of mutation, crossover and hill climbing, the Robinhood hyper-heuristic uses the same ordering of groups and randomly fixing the ordering of heuristics within each group at a stage. There is also a strong empirical evidence in the literature that this ordering is a good choice even for selection hyper-heuristics as reported in [3,21]. Our hyper-heuristic uses the same ordering in the subsequent stage if there is an improvement in the solution quality at a given stage. Otherwise, without changing the group ordering, another random ordering of low level heuristics within each group is generated for the subsequent stage.

In this study, we discretised the choices for the control parameters provided in Hyflex into five different levels of intensity and depth of search: $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. A low level heuristic with the chosen parameter setting is treated as a different heuristic, hence producing five heuristics instead of one. The crossover operator is not commonly used by single point-based search techniques. In order to be able to apply a crossover heuristic, an extra solution (argument) is required. In our approach, the current solution is always used as one of the solutions passed to the crossover operator. To decide on the second solution, we used a circular queue containing M best solutions so far. Again, the round robin strategy is employed. A pointer is used to indicate which solution will be used from this queue during crossover. After a crossover operation, the pointer advances to the next item in the list for the next crossover.

A modified version of the adaptive acceptance method in [3] is introduced for the move acceptance. This acceptance method accepts all improvements as usual, but the deteriorations are accepted with respect to an adaptively changing rate, denoted as *acceptanceRate*. Assuming a minimisation problem, let $f(x)$ denote the quality of a given solution x , then if $f(S')$ is less than $f(S)$, then S' is accepted, otherwise S' is accepted with a uniform probability of *acceptanceRate* (Algorithm 1, line 8). Initially, only strictly improving moves are accepted. However, if the solution does not improve for one stage, only the moves generating improving or equal quality new solutions are accepted. If the solution does not improve for another following stage, then threshold move acceptance is activated based on *acceptanceRate*. A reinforcement learning mechanism is used for adapting the value of *acceptanceRate*. If the solution gets stuck at a local optimum for a given stage, then *acceptanceRate* is increased by a δ value for the next stage, making it more likely that a worsening solution is accepted. Conversely, if the solution in hand worsens in a given stage, then the *acceptanceRate* is reduced by δ , making it less likely for a worsening solution to be accepted. the value of δ is fixed arbitrarily as 0.01 during the experiments. The *acceptanceRate* value updates are intended to help the search navigate out of local optima, and focus the search if it is progressing well.

4 Empirical Results

In all the cases, a run terminates after $t = 600$ seconds or equivalent to 10 minutes as the competition requires. The equivalent value is obtained using the

benchmarking tool provided at the competition website. Initial experiments are performed for parameter tuning of number of stages, k . Testing different values of $k = \{1, 2, 10, 20, 30, \dots, 100, 200, 300, 1000\}$ revealed that the best k value is 200 in the current. Due to the memory limitation, the size of the stored solutions M has fixed arbitrarily as 50. The Formula1 scoring system is used for comparing the performance of hyper-heuristics. The top hyper-heuristic receives 10 points, the second one gets 8 and then 6, 5, 4, 3, 2, 1, respectively. The remaining approaches get zero point. These points are accumulated as a score for a hyper-heuristic over all instances.

4.1 Performance Comparison to the Mock Competition Hyper-heuristics

The performance of the Robinhood hyper-heuristic (RHH) is compared to the performances of eight different previous hyper-heuristics (HH1-HH8) across four problem domains, each with 10 different instances, as provided for the mock competition at: www.asap.cs.nott.ac.uk/external/chesc2011/defaulthh.html The problem domains used in the mock competition are Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Personnel Scheduling (PS) and Permutation Flow Shop (PFS). A single run is performed using each problem instance in the mock competition.

The Robinhood selection hyper-heuristic outperforms the mock competition hyper-heuristics with a Formula 1 score of 264.25 in the overall (Figure 1). It obtains the best results in 7 out of 10 instances with 2 draws in the MAX-SAT and with no draws in the 1D Bin Packing. In the personnel scheduling problem, RHH produces the best results in 2 instances. RHH delivers a relatively poor performance in the permutation flow shop problem domain as compared to its performance on the other domains. RHH is the winner in the MAX-SAT and 1D Bin Packing problem domains and loses to the other hyper-heuristics in the rest of the problem domains.

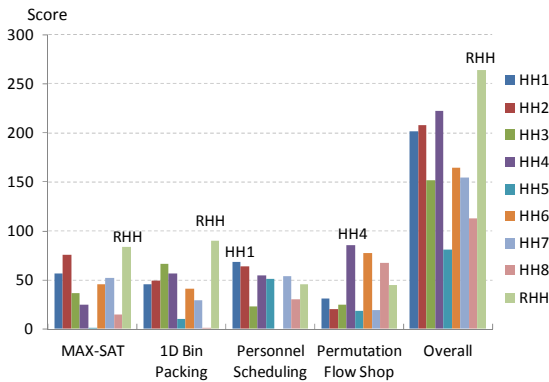


Fig. 1. Comparisons of the different hyper-heuristics over each domain based on Formula1 scores

4.2 Analysis of RHH and Its Performance Comparison to the CHESC Competitors

The Robinhood selection hyper-heuristic is run for 31 times across six problem domains including Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Personnel Scheduling (PS), Permutation Flow Shop (PFS), Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). We have initially analysed whether the proposed heuristic selection method makes effective use of the low level heuristics. Figure 2 shows the *percentage utilisation* of the low level heuristics considering improving moves only with respect to the number of times a heuristic gets selected while solving an arbitrary instance from each problem domain as an example. A similar phenomena is observed for the other instances. Not all the low level heuristics are useful in improving a solution. For example, in 1D Bin Packing, LLH1, LLH5 and LLH7 generates no improvements. Most of the improving moves are due to mutational heuristics rather than hill climbers across all problem domains. Ruin and recreate heuristics are more successful for creating improving moves in the Permutation Flow Shop domain, while a hill climbing heuristic creates the most improvements in 1D bin packing problem domain. Although a heuristic that does not generate any improvement could still be useful when used in combination with another heuristic, so there is a lot of research scope for methodologies attempting to reduce the number of low level heuristics before and during a run.

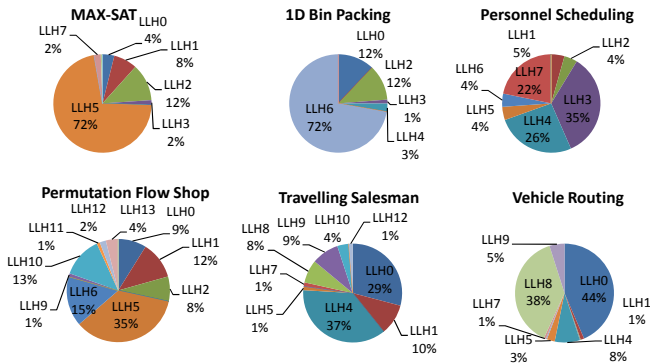


Fig. 2. Percentage utilisation of low level heuristics obtained from a sample run while solving an arbitrary instance from each problem domain

We have investigated the behavior of RHH based on the proposed acceptance method. In most of the cases, RHH rapidly improves the quality of the solution in hand. After a while, the improvement process slows down as the approach reaches a local optimum. Still, it seems that the proposed move acceptance works well as a part of the proposed hyper-heuristic, allowing further improvement in time even if takes a while to obtain an improved solution. The proposed move acceptance allows partial restarts and the extension of these restarts change

if there is no improvement and in general there is some improvement. This behaviour is illustrated in Figure 3 for an arbitrarily selected instance from each problem domain. Similar phenomena are observed in the other instances. RHH seems to require partial restarts while solving problems from the MAX-SAT, Permutation Flow Shop and Personnel Scheduling problem domains more than the others. For the Vehicle Routing and somewhat 1D Bin Packing problem domains, RHH generates continuous improvement via the heuristics.

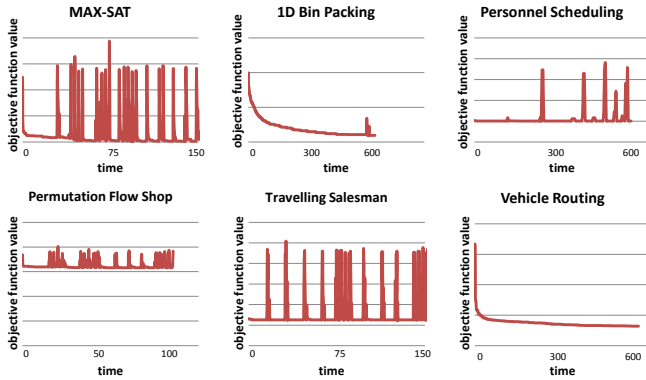


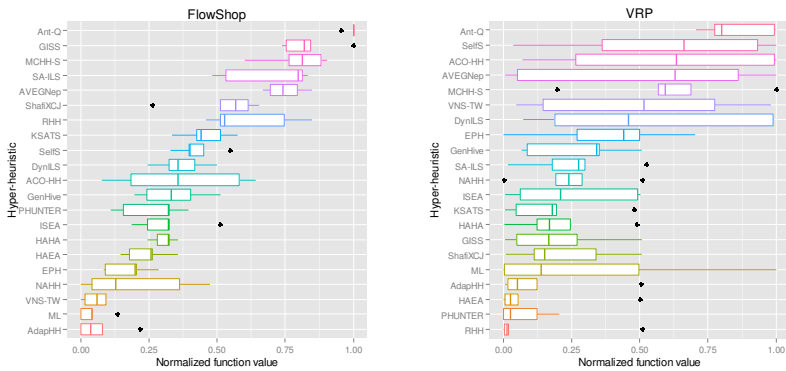
Fig. 3. Plots of the objective value versus time from a sample run while solving an arbitrary instance from each problem domain

The performance of the Robinhood selection hyper-heuristic is compared to the performances of CHeSC hyper-heuristics as provided at the competition website. The comparison is based on the Formula1 scoring system using the median of 31 runs for each instance. The points are accumulated as a score for each hyper-heuristic over five instances from six problem domains including Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Personnel Scheduling (PS), Permutation Flow Shop (PFS), Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). Table 2 summarises the results. The table shows that RHH is ranking the fourth when compared to the algorithms implemented by the CHeSC competitors with a total score of 93.70.

The normalized function values based on the median of 31 runs for each instance can also be used to evaluate the performance of the different hyper-heuristics [9]. The objective values are calculated and rescaled in $[0,1]$ as a score to rank the different approaches for each problem domain. Figure 4 illustrates the normalized function for the problem domains in which Robinhood hyper-heuristic performs the best and worst. The results are still consistent with the previous findings. The Robinhood hyper-heuristic is the top in the VRP problem domain. In the MAX-SAT and 1D Bin Packing problem domains, the RHH produces good quality of solutions comparing to other approaches. In the other domains, RHH produces a relatively poor performance. It delivers the worst performance, particularly in the permutation flow shop problem domain.

Table 2. Formula 1 scores of the top ten hyper-heuristics among CHeSC finalists and Robinhood hyper-heuristic (RHH) across six problem domains

HH	SAT	BP	PS	PFS	TSP	VRP	TOT
AdapHH	33.10	45.00	8.00	37.00	40.25	11.00	174.35
VNS-TW	33.60	2.00	37.50	34.00	16.25	4.00	127.35
ML,	11.00	8.00	29.50	39.00	13.00	21.00	121.50
RHH	22.70	26.00	16.00	0.00	3.00	26.00	93.70
PHUNTER	8.00	3.00	11.50	9.00	26.25	29.00	86.75
EPH	0.00	7.00	9.50	21.00	36.25	12.00	85.75
HAHA	31.10	0.00	23.50	3.50	0.00	13.00	71.10
NAHH	11.50	19.00	1.00	22.00	12.00	5.00	70.50
ISEA	3.50	28.00	14.50	3.50	11.00	4.00	64.50
KSATS-HH	21.70	7.00	7.50	0.00	0.00	19.00	55.20

**Fig. 4.** Normalized function values

5 Conclusion and Future Work

Hyper-heuristics have been shown to be effective solution methods across many problem domains. In this study, an easy-to-implement selection hyper-heuristic combining a round-robin strategy-based neighbourhood selection and an adaptive move acceptance methods is introduced. The Robinhood hyper-heuristic allocates equal share from the overall time for each low level heuristic ordering them randomly within their categories of mutation, crossover and local search. In this manner, memetic algorithm is imitated under a single point-based search framework with multiple operators. The Robinhood hyper-heuristic operates in stages and prior to each stage, relevant decisions are made for the ordering of heuristics within groups and parameters of the system components, such as move acceptance. The experimental results show that proposed hyper-heuristic is a simple yet very powerful and general strategy outperforming many previously

proposed selection hyper-heuristics across six different domains. As for the future work, we plan to work on more learning components within this framework to further improve its performance without introducing additional parameters and making the hyper-heuristic more complicated. The Robinhood hyper-heuristic has only three parameters: δ , M and k . All these values are currently tuned after some experimentation, but of course, the question is whether it is possible to adapt them during the search process, in particular the duration allocated for each stage and get improved performance. We have observed that some of the heuristics are almost useless at different stages of the search process, then by reducing the number of heuristics involved in the search process at a stage would be a good idea.

References

1. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 457–474. Kluwer (2003)
2. Burke, E., Kendall, G., Misir, M., Özcan, E.: Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 1–18 (2010)
3. Burke, E.K., Curtois, T., Hyde, M.R., Kendall, G., Ochoa, G., Petrovic, S., Rodríguez, J.A.V., Gendreau, M.: Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In: *IEEE Congress on Evolutionary Computation*, pp. 1–8 (2010)
4. Burke, E.K., Gendreau, M., Ochoa, G., Walker, J.D.: Adaptive iterated local search for cross-domain optimisation. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011*, pp. 1987–1994. ACM, New York (2011)
5. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: *Hyper-heuristics: A survey of the state of the art*. Technical report (2013)
6. Burke, E., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.: A classification of hyper-heuristics approaches. In: Gendreau, M., Potvin, J.-Y. (eds.) *Handbook of Metaheuristics*, vol. 146, pp. 449–468. Springer (2010)
7. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent Developments. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) *Adaptive and Multilevel Metaheuristics*. SCI, vol. 136, pp. 3–29. Springer, Heidelberg (2008)
8. Cowling, P., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
9. Di Gaspero, L., Urli, T.: Evaluation of a Family of Reinforcement Learning Cross-Domain Optimization Heuristics. In: Hamadi, Y., Schoenauer, M. (eds.) *LION 2012*. LNCS, vol. 7219, pp. 384–389. Springer, Heidelberg (2012)
10. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J.F., Thompson, G.L. (eds.) *Industrial Scheduling*, pp. 225–251. Prentice-Hall, Inc., New Jersey (1963)
11. Hsiao, P.C., Chiang, T.C., Fu, L.C.: A vns-based hyper-heuristic with adaptive computational budget of local search. In: *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8 (June 2012)

12. Drake, J.H., Özcan, E., Burke, E.K.: An Improved Choice Function Heuristic Selection for Cross Domain Heuristic Search. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part II. LNCS, vol. 7492, pp. 307–316. Springer, Heidelberg (2012)
13. Kalender, M., Kheiri, A., Özcan, E., Burke, E.K.: A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem. In: 2012 12th UK Workshop on Computational Intelligence (UKCI), pp. 1–8 (September 2012)
14. Kubalík, J.: Hyper-Heuristic Based on Iterated Local Search Driven by Evolutionary Algorithm. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 148–159. Springer, Heidelberg (2012)
15. Misir, M., Verbeek, K., De Causmaecker, P., Vanden Berghe, G.: A new hyper-heuristic implementation in HyFlex: a study on generality. In: Fowler, J., Kendall, G., McCollum, B. (eds.) Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Application, pp. 374–393 (August 2011)
16. Moscato, P., Norman, M.G.: A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: Proceedings of the International Conference on Parallel Computing and Transputer Applications, pp. 177–186. IOS Press (1992)
17. Nguyen, S., Zhang, M., Johnston, M.: A genetic programming based hyper-heuristic approach for combinatorial optimisation. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1299–1306. ACM, New York (2011)
18. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 136–147. Springer, Heidelberg (2012)
19. Ochoa, G., Walker, J., Hyde, M., Curtois, T.: Adaptive Evolutionary Algorithms and Extensions to the HyFlex Hyper-heuristic Framework. In: Coello Coello, C.A., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part II. LNCS, vol. 7492, pp. 418–427. Springer, Heidelberg (2012)
20. Özcan, E., Bilgin, B., Korkmaz, E.E.: Hill Climbers and Mutational Heuristics in Hyperheuristics. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 202–211. Springer, Heidelberg (2006)
21. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis* 12(1), 3–23 (2008)
22. Özcan, E., Kheiri, A.: A hyper-heuristic based on random gradient, greedy and dominance. In: Gelenbe, E., Lent, R., Sakellari, G. (eds.) *Computer and Information Sciences II*, pp. 557–563. Springer London (2012)
23. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 2011–2018 (2011)
24. Özcan, E., Parkes, A.J., Alkan, A.: The interleaved constructive memetic algorithm and its application to timetabling. *Comput. Oper. Res.* 39(10), 2310–2322 (2012)

A Multiobjective Approach Based on the Law of Gravity and Mass Interactions for Optimizing Networks

Álvaro Rubio-Largo and Miguel A. Vega-Rodríguez

Department of Technologies of Computers and Communications,
University of Extremadura, Polytechnic School, Cáceres, 10003 Spain
{ar1,mavega}@unex.es

Abstract. In this work, we tackle a real-world telecommunication problem by using Evolutionary Computation and Multiobjective Optimization jointly. This problem is known in the literature as the Traffic Grooming problem and consists on multiplexing or grooming a set of low-speed traffic requests (Mbps) onto high-speed channels (Gbps) over an optical network with wavelength division multiplexing facility. We propose a multiobjective version of an algorithm based on the laws of motions and mass interactions (Gravitational Search Algorithm, GSA) for solving this NP-hard optimization problem. After carrying out several comparisons with other approaches published in the literature for this optical problem, we can conclude that the multiobjective GSA (MO-GSA) is able to obtain very promising results.

Keywords: Multiobjective optimization, Gravitational Search Algorithm, Traffic Grooming, WDM optical networks.

1 Introduction

Optical networks have attracted more attention in the last years. This is due to the fact that our current data networks do not provide enough bandwidth for allocating the enormous number of current data requests. However, the bandwidth in optical networks is in the range of Tbps, which is sufficient for solving this bandwidth problem.

In optical networks the most common technology to make the most of the bandwidth is the Wavelength Division Multiplexing (WDM). This technology allows us to divide each fiber (Tbps) into several wavelengths of light (λ) or channels (Gbps). A lightpath is defined as the physical path from a source node to a destination node over a specific wavelength of light, see Figure 1.

Unfortunately, the requirements of the majority of data requests are in the range of Mbps (low-speed requests), which is translated into a waste of bandwidth per channel. The use of access station at each optical node allows us to groom several low-speed requests onto high-speed lightpaths in order to minimize the waste of bandwidth [12]. This problem of grooming requests onto high-speed channels is known in the literature as the Traffic Grooming problem [13].

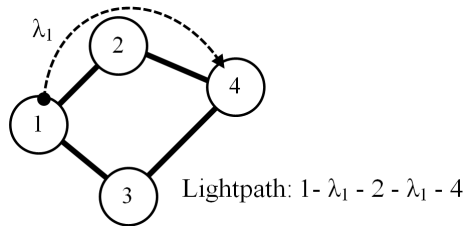


Fig. 1. Lightpath concept

Many authors have dealt with this telecommunication problem in the last years. A formal description of the Traffic Grooming by using integer linear programming is presented in [13]. Furthermore, they proposed two efficient heuristics that have resulted to be reference methods for testing new approaches, they are: *Maximizing Single-hop Traffic* (MST) and *Maximizing Resource Utilization* (MRU). A INTeGrated Grooming PROCedure (INGPROC) based on an auxiliary graph model is reported in [11]. This problem has been also tackled by using the Clique Partitioning concept in [2]. Other authors have solved this problem by using multiobjective optimization. A well-known Multiobjective Evolutionary Algorithm (MOEA) is presented in [7], the Strength Pareto Evolutionary Algorithm (SPEA). Recently, in [9], the authors propose two innovative MOEAs based on the Differential Evolution (DE) and Variable Neighbourhood Search (VNS). Furthermore, they present a multiobjective comparison among these two MOEAs and two standard MOEAs in the multiobjective domain: the Fast Non-Dominated Sorting Genetic Algorithm (NSGA-II) and the Strength Pareto Evolutionary Algorithm 2 (SPEA2).

The main contribution of this work is the proposal of an innovative MOEA based on the Gravitational Search Algorithm (GSA) [8] for solving the Traffic Grooming problem in WDM optical networks. The GSA is a Swarm Intelligence evolutionary algorithm based on the laws of gravity and mass interactions. However, we present a multiobjective version of the GSA (MO-GSA). We demonstrate the effectiveness of the MO-GSA by comparing it with other heuristics and metaheuristics published in the literature.

The rest of this paper is organized as follows. A description of the Traffic Grooming problem is presented in Section 2. Section 3 is devoted to describe the Multiobjective Gravitational Search Algorithm. In Section 4, we present a comparison between the MO-GSA and several methods published in the literature. Finally, we summarize the conclusions of the work and discuss possible lines of future work in Section 5.

2 Traffic Grooming Problem

In this paper, we define an optical network topology as a directed graph $G=(N, E)$, where N is the set of optical nodes and E is the set of physical links connecting nodes.

The following assumptions have been taken into account in this paper:

- The set of low-speed data requests is known in advance and cannot be split into lower speed requests and routed separately. Furthermore, the granularity of each low-speed request is OC- x , $x \in \{1, 3, 12, \text{ and } 48\}$, where $x \times 51.84\text{Mb/s}$.
- *Multi-hop grooming facility* [13]. Several concatenated lightpaths can be used with the aim of accommodating a low-speed request.
- *Wavelength continuity constraint* [5]. The same wavelength of light (λ) must be used across all physical fibers traversed in a lightpath. Thus, none of the nodes in N supports *wavelength conversion*.
- For all links in E , the number of wavelengths (W) is the same, as well as the capacity (C). Furthermore, we suppose that in every link $(m,n) \in E$ where $m, n \in N$, the propagation delay is equal to one ($d_{mn}=1$).
- For all nodes in N , the number of outgoing links is equal to the number of incoming links. Furthermore, at each node the number of transceivers (T_i/R_i or T) will be greater than or equal to one.

The more relevant parameters in the Traffic Grooming problem are listed below:

- A : traffic demand matrix

$$A^x = [A_{sd}^x; s, d \in N]_{|N| \times |N|},$$

where A_{sd}^x is the number of OC- x connection requests between node pair (s, d) .

- S_{sd}^x , $x \in \{1, 3, 12, \text{ and } 48\}, \forall s, d \in N$: number of OC- x streams requested from node s to node d that are successfully routed.
- P_{mn} , $\forall m, n \in N$: number of fibers interconnecting nodes m and n .
- V_{ij}^w , $\forall w \in \{1 \dots W\}, \forall i, j \in N$: number of lightpaths from node i to node j on wavelength w (Virtual Topology).
- $P_{mn}^{i,j,w}$, $\forall w \in \{1 \dots W\}, \forall i, j, m, n \in N$: number of lightpaths between node i to node j routed through fiber link (m, n) on wavelength w (Physical Topology route).

Thus, given an optical network topology, a set of connection requests with different bandwidth granularity, a fixed number of available wavelengths per fiber, a capacity of each wavelength, and a fixed number of transmitters and receivers at each node, the Traffic Grooming problem may be stated as a Multiobjective Optimization Problem (MOOP) [3], in which our objective functions are:

- *Traffic Throughput* (y_1): Maximize the total successfully routed low-speed traffic demands on virtual topology, see equation 1.
- *Number of Transceivers or Lightpaths* (y_2): Minimize the total number of transceivers used or the number of lightpaths established, see equation 2.
- *Average Propagation Delay* (APD, y_3). Minimize the average hop count of lightpaths established, due to we assume $d_{mn} = 1$ in all physical fiber links (m, n) , see equation 3.

$$y_1 = \text{Max} \sum_{s=1}^{|N|} \sum_{d=1}^{|N|} \sum_{t=1}^{A_{sd}^x} (x \times S_{sd}^{x,t}) \quad (1)$$

$$x \in \{1, 3, 12, \text{ and } 48\}$$

$$y_2 = \text{Min} \sum_{i=1}^{|N|} \sum_{j=1}^{|N|} \sum_{w=1}^W V_{ij}^w \quad (2)$$

$$y_3 = \text{Min} \frac{\sum_{i=1}^{|N|} \sum_{j=1}^{|N|} \sum_{m=1}^{|N|} \sum_{n=1}^{|N|} (d_{mn} \times \sum_{w=1}^W P_{mn}^{ij,w})}{\sum_{i=1}^{|N|} \sum_{j=1}^{|N|} \sum_{w=1}^W V_{ij}^w} \quad (3)$$

For a complete formulation of the Traffic Grooming problem, including all parameters, variables, constraints, objective functions, and an illustrative example, please see [9].

3 Multiobjective Gravitational Search Algorithm

The Gravitational Search Algorithm (GSA) is a population based algorithm created by Rashedi et al. [8] in 2009. This new optimization algorithm is based on the law of gravity and mass interactions. The searcher agents (individuals) are a collection of masses which interact with each other based on the Newtonian gravity laws of motion. In this work, the individuals have been designed as in [9], for further information, please refer to [9].

Since the Traffic Grooming is a MOOP, we have adapted the standard GSA to the multiobjective context (MO-GSA) as follows:

1. We set the output set of non-dominated solutions (PF) as empty.
2. We generate NA searcher agents randomly, x_1, x_2, \dots, x_{NA} .
3. The input parameters of the MO-GSA are initialized: K_{best} , α , and G_0 . The K_{best} parameter is used to control the exploration and exploitation of the algorithm, in this work, it is initialized as NA . On the other hand, G_0 is the initial value of the Gravitational constant (G) and α is a parameter for decreasing the value of G throughout the execution of the algorithm. Note that G will control the search accuracy of the MO-GSA.
4. We compute, for each searcher agent x_i in $\{x_1, x_2, \dots, x_{NA}\}$, its value of $MOFitness$ according to equation 4. In equation 4, $Rank$ and $CrowdingDistance$ are well-known concepts in the multiobjective field; so, for further information about them, please refer to [4].

$$MOFitness(x_i) = \left(2^{Rank(x_i)} + \frac{1}{1 + CrowdingDistance(x_i)} \right)^{-1} \quad (4)$$

5. We update the value of the Gravitational constant (G), see equation 5. Note that, in equation 5, t_r is the elapsed time and T_r is the total runtime established.

$$G = G_0 e^{-\alpha \frac{t_r}{T_r}} \quad (5)$$

6. We sort the agents $\{x_1, x_2, \dots, x_{NA}\}$ by *MOFitness* and we look for the best (x_{best}) and worst (x_{worst}) searcher agent. To decide which is the best or the worst, we use the value of *MOFitness*. Since the value of *MOFitness* needs to be maximized, we select the individual with higher and lower value of *MOFitness*; respectively.
7. We compute the mass for each agent x_i in $\{x_1, x_2, \dots, x_{NA}\}$, see equations 6 and 7.

$$Q(x_i) = \frac{MOFitness(x_i) - MOFitness(x_{worst})}{MOFitness(x_{best}) - MOFitness(x_{worst})} \quad (6)$$

$$Mass(x_i) = \frac{Q(x_i)}{\sum_{j=1}^{NA} Q(x_j)} \quad (7)$$

8. For each agent x_i in $\{x_1, x_2, \dots, x_{NA}\}$, we calculate the force exerted by each agent x_j in $x_1 \dots x_{K_{best}}$, see equation 8 and 9.

$$F(x_i, x_j) = G \cdot \frac{Mass(x_i) \times Mass(x_j)}{r_{ij}} \cdot (x_j - x_i) \quad (8)$$

$$Force(x_i) = \sum_{j=1}^{K_{best}} rand[0, 1] \times F(x_i, x_j) \quad (9)$$

Note that, $x_1 \dots x_{K_{best}}$ is the set of the first K_{best} agents with best value of *MOFitness*. In equation 8, r_{ij} refers to the Euclidean distance between the agents x_i and x_j .

9. We calculate the acceleration of each searcher agent x_i in $\{x_1, x_2, \dots, x_{NA}\}$ according to equation 10:

$$Acceleration(x_i) = \frac{Force(x_i)}{Mass(x_i)} \quad (10)$$

10. We update each agent x_i in $\{x_1, x_2, \dots, x_{NA}\}$, see equations 11 and 12.

$$Velocity(x_i) = rand[0, 1] \times PreviousVelocity(x_i) + Acceleration(x_i) \quad (11)$$

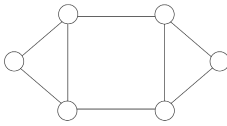
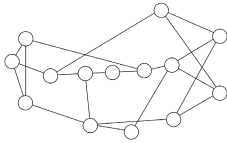
$$x_i = x_i + Velocity(x_i) \quad (12)$$

11. We update the set of non-dominated solutions (*PF*) with each agent x_i in $\{x_1, x_2, \dots, x_{NA}\}$. We remove from *PF* all searcher agents dominated by x_i . Then, we add x_i to *PF* only if no other searcher agent dominates it.

12. We decrease linearly K_{best} to 1.
13. If the stopping criterion is satisfied, then stop and output PF . Otherwise, go to Step 4.

As we observe, the input parameters of the MO-GSA are determinant to achieve a good balance between exploration and exploitation. After a parameter tuning, for this particular problem, the best configuration found for the MO-GSA is: $NA=100$ searcher agents, $G_0=1000$, $\alpha=2$, and K_{best} is initialized as NA (total number of agents).

Table 1. Specifications for the topologies 6-node and NSF

6-node Network		
	N : 6 E : 16 C: OC-48 Traffic: 988 OC-1 units T={3, 4, 5, 7} W={3} T={3, 4, 5} W={4} Norm. Points: Max=(988,T*6,6) Min=(1,1,1)	
	NSF Network	
		N : 14 E : 42 C: OC-192 Traffic: 5724 OC-1 units T={3, 4, 5} W={3} T={4, 5, 6} W={4} Norm. Points: Max=(5724,T*14,9) Min=(1,1,1)

4 Experimental Results

A comparison between the MO-GSA and several approaches published in the literature is shown in this section.

We start comparing the MO-GSA with the MOEAs proposed in [9]: Differential Evolution with Pareto Tournaments (DEPT), Multiobjective Variable Neighbourhood Search (MO-VNS), Fast Non-Dominated Sorting Genetic Algorithm (NSGA-II), and Strength Pareto Evolutionary Algorithm 2 (SPEA2). The comparison among these five MOEAs is conducted by using two well-known multiobjective metrics, the Hypervolume quality indicator (HV) [15] and the Set Coverage (SC) indicator [14]. Furthermore, we have used the 6-node network and the National Science Foundation (NSF) network. The specifications for both topologies, as well as the scenarios tested are shown in Table 1. The sets of low-speed requests are the same as in [9] for both topologies, with a total amount of 988 and 5724 OC-1 units for the topologies 6-node and NSF, respectively. The configuration of the algorithms DEPT, MO-VNS, NSGA-II, and SPEA2 is presented in [9].

Table 2. Average value of HV in 30 independent runs. The notation used for pointing the statistically non-significant differences between pairs of algorithms is the following:: (*) DEPT and MO-VNS, (**) DEPT and MO-GSA, (†) MO-VNS and MO-GSA, (‡) NSGA-II and SPEA2.

6-node Network						
	DEPT	MO-VNS	NSGA-II	SPEA2	MO-GSA	
T=3 W=3	37.95%	38.16%	36.16%	35.73%	37.76%	**
T=4 W=3	48.89%	48.50%	46.64%	45.88%	48.63%	†
T=5 W=3	56.95%	55.52%	54.41%	53.07%	57.11%	**
T=7 W=3	64.81%	63.36%	62.40%	60.68%	66.50%	
T=3 W=4	38.27%	38.20%	36.20%	35.69%	37.83%	*
T=4 W=4	49.02%	48.58%	46.38%	45.95%	48.64%	†
T=5 W=4	58.22%	56.56%	55.82%	54.95%	57.85%	
Average	50.59%	49.84%	48.29%	47.42%	50.62%	
NSF Network						
	DEPT	MO-VNS	NSGA-II	SPEA2	MO-GSA	
T=3 W=3	29.10%	19.11%	28.61%	28.36%	32.01%	
T=4 W=3	39.32%	32.65%	38.55%	37.88%	42.48%	
T=5 W=3	48.78%	40.68%	46.70%	46.32%	51.40%	
T=4 W=4	39.91%	26.04%	38.35%	36.73%	42.24%	
T=5 W=4	48.03%	34.65%	46.92%	46.22%	51.00%	
T=6 W=4	54.72%	42.57%	53.34%	53.15%	57.61%	‡
Average	43.31%	32.62%	42.08%	41.44%	46.12%	

All MOEAs were run using g++ (GCC) 4.4.5 on a 2.3GHz Intel PC with 1GB RAM. Furthermore, the number of independent runs of the MOEAs is 30 for each data set, where the stopping criterion is based on the runtime and depend on the topology: 30s (6-node) and 360s (NSF).

Firstly, we compare the algorithms DEPT, MO-VNS, NSGA-II, SPEA2, and MO-GSA by using the HV indicator. This indicator is not free from arbitrary scaling of objective; thus, we have to normalize the set of solutions achieved by each approach. In this work, to normalize the solutions we have used different maximum and minimum normalization points, depending on the scenario. In Table 1, we report the maximum and minimum points to normalize each scenario. For example, in data set NSF T=4 W=3, the maximum and minimum points are: (5724, 56, 9) and (1, 1, 1); respectively. Note that the normalization points are different than in [9].

In Table 2, we present the average value of HV obtained by each MOEA in the different scenarios. On the one hand, we may observe that, in the small 6-node network topology, the DEPT algorithm obtains higher values than the MO-GSA in five out of seven data sets. However, we can notice that the results obtained by the MO-GSA are close to the results of the DEPT algorithm; in fact, the MO-GSA presents a higher final average HV for this network topology.

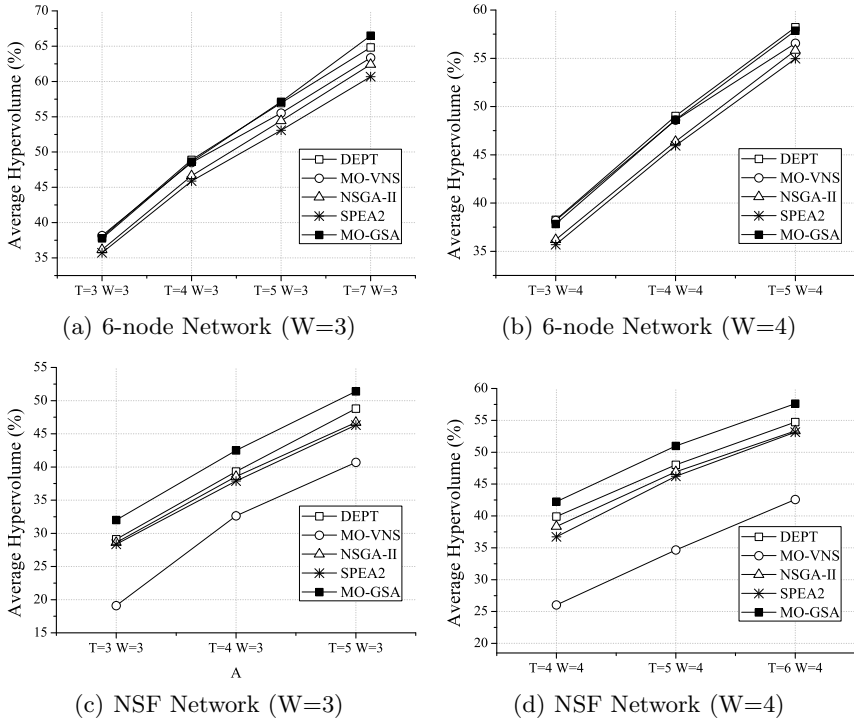


Fig. 2. Illustrative Hypervolume comparison among the MOEAs

On the other hand, we may observe that, for the large NSF network topology, the values of HV achieved by the MO-GSA are higher than those obtained by the DEPT, MO-VNS, NSGA-II, and SPEA2 in all scenarios. The final average shows that there exists a difference around 3% of HV between the MO-GSA and the second best MOEA (DEPT algorithm).

Furthermore, with the aim of making the comparison with a certain level of confidence, we have carried out the same statistical analysis as in [9]. In the statistical test, we consider a significance level of 5% (p-value under 0.05); therefore, the differences are unlikely to have occurred by chance with a probability of 95%. In Table 2, we have pointed those data sets in which the differences between two algorithms are not statistically significant.

In Figure 2 we plot the average value of HV obtained by each MOEA in order to clarify the comparison of HV among the five MOEAs.

In the second place, we compare the algorithms by using the SC indicator. This indicator is devoted to measure the fraction of non-dominated solutions obtained by an algorithm B which are covered by the non-dominated solutions obtained by an algorithm A. In Table 3, we present a comparison by pair of MOEAs using this multiobjective indicator. We may observe that the DEPT algorithm dominates the solutions of the MO-GSA in the small network topology; however,

Table 3. Comparison among the MOEAs by using the SC indicator ($A \succ B$)

6-node Network																
A	B	T=3	W=3	T=4	W=3	T=5	W=3	T=7	W=3	T=3	W=4	T=4	W=4	T=5	W=4	\overline{SC}
DEPT	MO-VNS	41.82%		60.34%		75.00%		82.35%		58.33%		66.20%		76.62%		65.81%
	NSGA-II	94.34%		98.36%		100.00%		90.77%		100.00%		91.38%		96.25%		95.87%
	SPEA2	97.73%		98.36%		98.31%		98.11%		100.00%		95.00%		93.94%		97.35%
	MO-GSA	46.97%		53.25%		76.67%		75.95%		58.57%		64.44%		53.85%		61.38%
MO-VNS	DEPT	48.10%		35.48%		21.59%		19.79%		37.66%		36.36%		22.81%		31.69%
	NSGA-II	73.58%		75.41%		59.70%		67.69%		76.60%		68.97%		57.50%		68.49%
	SPEA2	65.91%		72.13%		62.71%		60.38%		80.36%		71.67%		66.67%		68.55%
	MO-GSA	46.97%		42.86%		37.78%		44.30%		38.57%		45.56%		34.07%		41.44%
NSGA-II	DEPT	8.86%		2.15%		1.14%		5.21%		2.60%		9.09%		1.75%		4.40%
	MO-VNS	16.36%		8.62%		20.00%		22.06%		11.67%		8.45%		14.29%		14.49%
	SPEA2	47.73%		60.66%		64.41%		54.72%		62.50%		60.00%		50.00%		57.14%
	MO-GSA	9.09%		1.30%		4.44%		11.39%		5.71%		7.78%		3.30%		6.15%
SPEA2	DEPT	1.27%		3.23%		2.27%		5.21%		2.60%		5.05%		5.26%		3.55%
	MO-VNS	10.91%		6.90%		18.33%		14.71%		10.00%		7.04%		12.99%		11.55%
	NSGA-II	32.08%		16.39%		25.37%		26.15%		27.66%		25.86%		32.50%		26.57%
	MO-GSA	6.06%		1.30%		4.44%		11.39%		2.86%		6.67%		4.40%		5.30%
MO-GSA	DEPT	59.49%		54.84%		22.73%		18.75%		28.57%		29.29%		43.86%		36.79%
	MO-VNS	43.64%		53.45%		40.00%		38.24%		51.67%		35.21%		53.25%		45.06%
	NSGA-II	94.34%		98.36%		95.52%		89.23%		95.74%		87.93%		97.50%		94.09%
	SPEA2	90.91%		96.72%		94.92%		88.68%		96.43%		93.33%		98.48%		94.21%

NSF Network														
A	B	T=3	W=3	T=4	W=3	T=5	W=3	T=4	W=4	T=5	W=4	T=6	W=4	\overline{SC}
DEPT	MO-VNS	40.74%		50.56%		40.09%		67.15%		42.77%		62.05%		50.56%
	NSGA-II	61.11%		64.32%		81.67%		68.67%		74.11%		77.14%		71.17%
	SPEA2	46.46%		41.41%		59.60%		50.51%		44.44%		49.49%		48.65%
	MO-GSA	14.84%		18.67%		15.43%		25.38%		14.95%		21.61%		18.48%
MO-VNS	DEPT	31.49%		36.29%		37.07%		14.35%		35.27%		19.49%		28.99%
	NSGA-II	41.67%		57.79%		46.67%		39.76%		42.13%		42.86%		45.15%
	SPEA2	15.15%		32.32%		24.24%		11.11%		19.19%		16.16%		19.70%
	MO-GSA	19.53%		21.52%		18.88%		15.90%		20.92%		17.73%		19.08%
NSGA-II	DEPT	12.71%		12.10%		6.23%		12.66%		8.36%		5.88%		9.66%
	MO-VNS	8.33%		4.49%		15.09%		7.30%		15.66%		4.82%		9.28%
	SPEA2	37.37%		35.35%		26.26%		31.31%		24.24%		26.26%		30.13%
	MO-GSA	1.17%		0.00%		1.33%		1.53%		0.54%		1.39%		0.99%
SPEA2	DEPT	8.84%		11.29%		7.48%		17.30%		13.82%		10.66%		11.56%
	MO-VNS	11.11%		2.81%		6.60%		8.03%		9.64%		1.81%		6.67%
	NSGA-II	23.61%		26.13%		25.00%		22.89%		30.46%		28.57%		26.11%
	MO-GSA	0.00%		0.63%		2.13%		0.31%		0.82%		1.66%		0.92%
MO-GSA	DEPT	69.61%		62.10%		78.19%		64.14%		76.73%		71.69%		70.41%
	MO-VNS	49.07%		58.43%		59.91%		56.20%		59.64%		51.81%		55.84%
	NSGA-II	98.61%		99.00%		97.22%		96.99%		97.97%		96.57%		97.73%
	SPEA2	95.96%		95.96%		92.93%		96.97%		93.94%		92.93%		94.78%

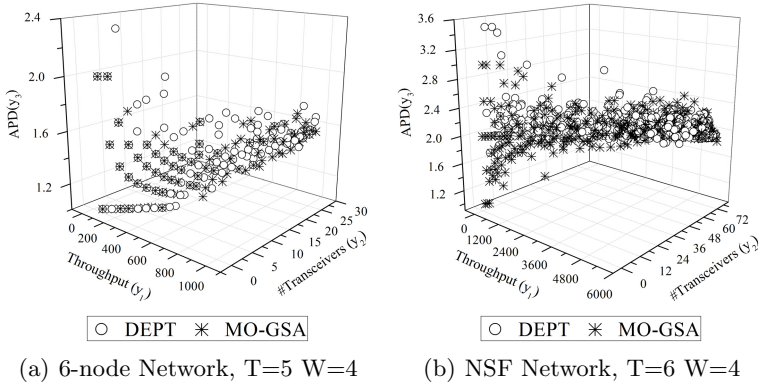


Fig. 3. Pareto fronts obtained by the algorithms DEPT and MO-GSA

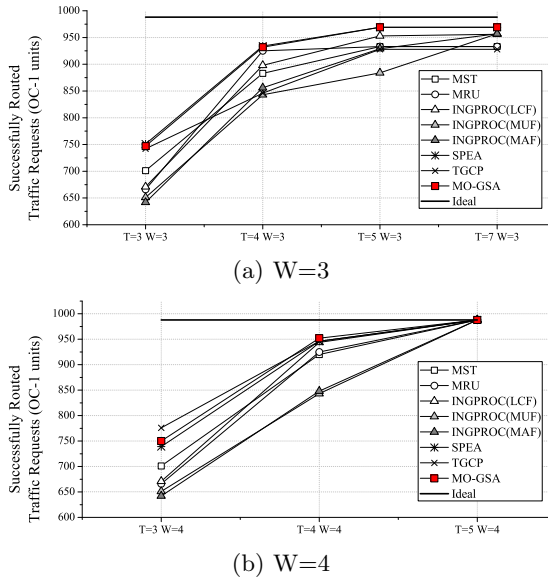


Fig. 4. Maximum throughput comparison (y_1) among MO-GSA, MST, MRU, INGPROC (LCF, MUF, and MAF), SPEA, and TGCP in the 6-node Network

we can see that the MO-VNS only dominates 41.44% of the solutions obtained by the MO-GSA, and the well-known NSGA-II and SPEA2 less than 7%. If we focus on the large network topology (NSF), we can see that the MO-GSA clearly dominates the Pareto fronts achieved by the approaches published in [9].

Finally, in Figure 3 we present the Pareto front obtained by the two best algorithms (DEPT and MO-GSA) in the scenarios 6-node $T=5$ $W=4$ and NSF $T=6$ $W=4$.

On the other hand, we compare the MO-GSA with several methods proposed in the literature which deal with the Traffic Grooming by only optimizing the total successfully routed low-speed traffic demands (y_1) over the 6-node network. Therefore, for each scenario, we compare the maximum values of y_1 obtained by the MO-GSA and by the following methods: Maximizing Single-Hop Traffic (MST) [13], Maximizing Resource Utilization (MRU) [13], INtegrated Grooming PROCedure (INGPROC) [11] with several traffic policies (*Least Cost First* (LCF), *Maximum Utilization First* (MUF), and *Maximum Amount First* (MAF)), Strength Pareto Evolutionary Algorithm (SPEA) [7], and the Traffic Grooming based on Clique Partitioning (TGCP) [2].

In Figure 4 we present an illustrative comparison among the approaches in different scenarios ($W = 3$ and $W = 4$). As we may observe, the MO-GSA is able to obtain a maximum throughput higher than or equal to other approaches published in the literature which are only focused on optimizing this objective function (y_1). Note that, the MO-GSA not only optimizes this objective function, but also minimizes the number of transceivers used and the average propagation delay at the same time.

5 Conclusions and Future Works

In this work we propose the use of a Multiobjective Evolutionary Algorithm based on the laws of motions and mass interactions for optimizing a real-world telecommunication problem: the Traffic Grooming problem. The selected algorithm is a multiobjective version of the Gravitational Search Algorithm, we refer to it as MO-GSA.

After performing a comparative study between the MO-GSA and several approaches published in the literature, we can conclude that this algorithm is an effective MOEA for solving the Traffic Grooming problem. As we have observed, it seems that it works better than the other algorithms when the network complexity increases because it generates new individuals taking not only parts from its parent, but also from the rest of the population, increasing the richness of search as a result. Furthermore, a maximum throughput comparison between the MO-GSA and other seven methods published in the literature has been carried out with successful results for the MO-GSA.

From the positive results obtained in this paper, it seems reasonable to think that the multiobjective proposals of this manuscript could be applied not only in other Telecommunication problems that involve routing [6], but also in diverse Engineering multiobjective problems [10].

As future work, we intend to apply the MO-GSA to the Traffic Grooming problem by using the large network Nippon Telegraph and Telephone (NTT) and make more comparisons with the MOEAs published in [9]. Furthermore, we would attempt to use another metaheuristic, such as Greedy Randomized Adaptive Search Procedure (GRASP), which seems to be more suitable for graph search [1].

Acknowledgements. This work was partially funded by the Spanish Ministry of Economy and Competitiveness and the ERDF (European Regional Development Fund), under the contract TIN2012-30685 (BIO project). Álvaro Rubio-Largo is supported by the research grant PRE09010 from Gobierno de Extremadura (Consejería de Economía, Comercio e Innovación) and the European Social Fund (ESF).

References

1. Arroyo, J., Vieira, P., Vianna, D.: A grasp algorithm for the multi-criteria minimum spanning tree problem. *Annals of Operations Research* 159, 125–133 (2008)
2. De, T., Pal, A., Sengupta, I.: Traffic Grooming, Routing, and Wavelength Assignment in an Optical WDM Mesh Networks Based on Clique Partitioning. *Photonic Network Communications* 20, 101–112 (2010)
3. Deb, K.: *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York (2001)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 182–197 (2000)
5. Gagnaire, M., Koubaa, M., Puech, N.: Network Dimensioning under Scheduled and Random Lightpath Demands in All-Optical WDM Networks. *IEEE Journal on Selected Areas in Communications* 25(S-9), 58–67 (2007)
6. Gong, Y.J., Zhang, J., Liu, O., Huang, R.Z., Chung, H.H., Shi, Y.H.: Optimizing the Vehicle Routing Problem With Time Windows: A Discrete Particle Swarm Optimization Approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 42(2), 254–267 (2012)
7. Prathombutr, P., Stach, J., Park, E.K.: An Algorithm for Traffic Grooming in WDM Optical Mesh Networks with Multiple Objectives. *Telecommunication Systems* 28, 369–386 (2005)
8. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: A Gravitational Search Algorithm. *Information Sciences* 179(13), 2232–2248 (2009), Special Section on High Order Fuzzy Sets
9. Rubio-Largo, A., Vega-Rodríguez, M.A., Gomez-Pulido, J.A., Sanchez-Perez, J.M.: Multiobjective Metaheuristics for Traffic Grooming in Optical Networks. *IEEE Transactions on Evolutionary Computation* (available online since June 2012), 1–17 (2012)
10. Xue, F., Sanderson, A., Graves, R.: Multiobjective Evolutionary Decision Support for Design Supplier Manufacturing Planning. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 39(2), 309–320 (2009)
11. Zhu, H., Zang, H., Zhu, K., Mukherjee, B.: A Novel Generic Graph Model for Traffic Grooming in Heterogeneous WDM Mesh Networks. *IEEE/ACM Transaction on Networking* 11, 285–299 (2003)
12. Zhu, K., Mukherjee, B.: A Review of Traffic Grooming in WDM Optical Networks: Architectures and Challenges. *Optical Networks Magazine* 4(2), 55–64 (2003)
13. Zhu, K., Mukherjee, B.: Traffic Grooming in an Optical WDM Mesh Network. *IEEE Journal on Selected Areas in Communications* 20(1), 122–133 (2002)
14. Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8, 173–195 (2000)
15. Zitzler, E., Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation* 3(4), 257–271 (1999)

A Multi-objective Feature Selection Approach Based on Binary PSO and Rough Set Theory

Liam Cervante¹, Bing Xue¹, Lin Shang², and Mengjie Zhang¹

¹ School of Engineering and Computer Science, Victoria University of Wellington,
PO Box 600, Wellington 6140, New Zealand

{Bing.Xue,Liam.Cervante,Mengjie.Zhang}@ecs.vuw.ac.nz

² State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing
210046, China
shanglin@nju.edu.cn

Abstract. Feature selection has two main objectives of maximising the classification performance and minimising the number of features. However, most existing feature selection algorithms are single objective wrapper approaches. In this work, we propose a multi-objective filter feature selection algorithm based on binary particle swarm optimisation (PSO) and probabilistic rough set theory. The proposed algorithm is compared with other five feature selection methods, including three PSO based single objective methods and two traditional methods. Three classification algorithms (naïve bayes, decision trees and k-nearest neighbours) are used to test the generality of the proposed filter algorithm. Experiments have been conducted on six datasets of varying difficulty. Experimental results show that the proposed algorithm can automatically evolve a set of non-dominated feature subsets. In almost all cases, the proposed algorithm outperforms the other five algorithms in terms of both the number of features and the classification performance (evaluated by all the three classification algorithms). This paper presents the first study on using PSO and rough set theory for multi-objective feature selection.

Keywords: Particle Swarm Optimisation, Feature Selection, Rough Set Theory, Multi-objective Optimisation.

1 Introduction

Classification tasks are to classify a given instance in the dataset to one of the known classes according to the information described by features. However, some of them are irrelevant or redundant features, which may even increase the classification error rate. Feature selection is to select a subset of relevant features to achieve similar or even better classification performance [6]. By reducing or eliminating the irrelevant and redundant features, feature selection can reduce the dimensionality of the data, simplify the learnt classifier, reduce the training time, and/or increase the classification accuracy [4,17].

Based on the evaluation criteria, feature selection methods are generally classified into two categories: wrapper and filter approaches [6]. Wrapper approaches

include a learning/classification algorithm in the evaluation procedure while filter approaches do not. Therefore, wrappers usually achieve better results than filter approaches, but they are computationally expensive. Filter approaches are more general and computationally cheaper than wrapper approaches, but an appropriate evaluation criterion is needed in filter approaches [4,6].

A challenge in feature selection is that the size of the search space is 2^n , where n is the total number of features. Most of the existing feature selection algorithms suffer from the problems of stagnation in local optima and high computational cost [4,17], especially for wrapper approaches. Evolutionary computation (EC) techniques are well-known for their global search ability. Particle swarm optimisation (PSO) [14] is a relatively recent EC technique, which is computationally less expensive than other EC algorithms. Therefore, PSO has recently gained more attention for solving feature selection problems [17,11].

Feature selection is a multi-objective problem, which aims to maximise the classification performance and minimise the number of features selected. However, most of the existing EC based feature selection algorithms are wrapper based single objective approaches. The use of wrapper algorithms is limited in real-world applications because of their high computational cost. Meanwhile, from a theoretical point of view, Yao and Zhao [21] have shown that probabilistic rough set can be a good measure in feature selection. Therefore, it is thought to develop a filter based multi-objective feature selection approach using PSO and probabilistic rough set theory.

1.1 Goals

The overall goal of this paper is to develop a filter based multi-objective feature selection approach to obtaining a set of non-dominated solutions, which include a smaller number of features and achieve similar or even better classification performance than using all features. To achieve this goal, we propose a multi-objective feature selection algorithm based on PSO and probabilistic rough set theory. Specifically, we will investigate

- whether using PSO and *probabilistic* rough set theory can reduce the number of features and maintain or even increase the classification performance, and can outperform the algorithm using PSO and *standard* rough set theory,
- whether considering *the number of features* in the fitness function can further reduce the number of features and maintain the classification performance,
- whether the proposed *multi-objective* algorithm can obtain a set of non-dominated feature subsets, and can outperform two traditional methods and the above three single objective methods, and
- whether the proposed algorithm is general to different learning algorithms.

2 Background

2.1 Binary Particle Swarm Optimisation

In PSO [14], a particle represents a candidate solution. Particles move in the D -dimensional search space to search for the best solutions. Particle i has a

position denoted by $x_i = (x_{i1}, x_{i2}, \dots, x_{iD})$ and a velocity denoted by $v_i = (v_{i1}, v_{i2}, \dots, v_{iD})$. During the search process, the best position visited so far by the particle is its personal best ($pbest$) and the best position obtained by the population thus far is called global best ($gbest$). Particles share information through $pbest$ and $gbest$ to update their positions and velocities to search for the optimal solutions. In binary PSO (BPSO) [9], x_i , $pbest$ and $gbest$ are restricted to 1 or 0. The position and velocity updating equations can be seen as follows:

$$v_{id}^{t+1} = w * v_{id}^t + c_1 * r_1 * (p_{id} - x_{id}^t) + c_2 * r_2 * (p_{gd} - x_{id}^t) \quad (1)$$

$$x_{id} = \begin{cases} 1, & \text{if } rand() < \frac{1}{1+e^{-v_{id}}} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where t represents the t th iteration in the evolutionary process. $d \in D$ represents the d th dimension in the search space. w is the inertia weight. c_1 and c_2 are acceleration constants. r_1 and r_2 are random constants uniformly distributed in $[0, 1]$. p_{id} and p_{gd} denote the values of $pbest$ and $gbest$ in the d th dimension. $rand()$ is a random number selected from a uniform distribution in $[0,1]$.

2.2 Rough Set Theory

Rough set theory [13] is an intelligent mathematical tool to handle uncertainty, imprecision and vagueness. One of the strengths of rough set theory is that it does not need any prior knowledge about data.

In rough set theory, knowledge and information is represented as an information system, which can be denoted as $I = (U, A)$, where U is a finite non-empty set of objects and A is the attributes/features that describe each object. For any $P \subseteq A$ and $X \subseteq U$, there is an equivalence relation defined as $IND(P) = \{(x, y) \in U * U | \forall a \in P, a(x) = a(y)\}$. If two objects in U satisfy $IND(P)$, they are indiscernible with regards to P . The equivalence relation $IND(P)$ induces a partition of U denoted by U/P , which induces the equivalence classes. The equivalence class of U/P containing x is given by $[x]_P = [x]_A = \{y \in U | (x, y) \in IND(P)\}$. The equivalence classes are the basic blocks to construct rough set approximations. For $X \subset U$, a lower approximation $\underline{P}X$ and an upper approximation $\overline{P}X$ of X with respect to $IND(P)$ are defined as follows [13]:

$$\underline{P}X = \{x \in U | [x]_P \subseteq X\} \quad \overline{P}X = \{x \in U | [x]_P \cap X \neq \emptyset\} \quad (3)$$

$\underline{P}X$ contains all the objects, which are surely belong to the target set X . $\overline{P}X$ contains the objects, which are surely or probably belong to the target set X .

An ordered pair $(\overline{P}X, \underline{P}X)$ is called a rough set. The concept of the reduct is fundamental in rough sets theory. A reduct is the essential part of $I = (U, A)$, which can achieve similar approximation power of classification as all the original features A . There could be many different reducts and feature selection using rough set theory is to remove redundant and irrelevant features to search for the smallest reduct (feature subset).

$\underline{P}X$ and $\overline{P}X$ in standard rough set theory were defined as two extreme cases in terms of the relationship of an equivalence class and the target set [13]. The degree of their overlap is not taken into account, which will unnecessarily limit its applications. Therefore, researchers investigate probabilistic rough set theory to relax the definitions of the lower and upper approximation [21]. The lower approximation is re-defined as Equation 4, where $\mu_P[x]$ shown is defined as a way to measure the fitness of a given instance $x \in X$.

$$\underline{apr}_P X = \{x | \mu_P[x] \geq \alpha\} \quad (4)$$

where

$$\mu_P[x] = \frac{|[x]_P \cap X|}{|[x]_P|} \quad (5)$$

α can be adjusted to restrict or relax the lower approximation. If a large number of instances X are in the target set but a small number are not in a given equivalence class, it can include them in the lower approximation. $\underline{apr}_P X = \underline{P}X$ when $\alpha = 1$.

From theoretical point of view, Yao and Zhao have claimed that probabilistic rough set can be a good way for feature selection problems [21]. However, it has not been proved by any experiment.

2.3 Related Work on Feature Selection

Traditional Feature Selection Approaches. Hall [7] proposes a filter feature selection method (Cfs) based on the correlation between features and class labels. FOCUS algorithm [1], a filter algorithm, exhaustively examines all possible feature subsets, then selects the smallest feature subset. However, the FOCUS algorithm is computationally inefficient because of the exhaustive search. Two commonly used wrapper methods are greedy search based sequential forward selection (SFS) [19] and sequential backward selection (SBS) [10]. In SFS (SBS), once a feature is selected (eliminated) it cannot be eliminated (selected) later, which causes the problem of so-called nesting effect. The “plus- l -take away- r ” method proposed by Stearns [16] could overcome this limitation by performing l times forward selection followed by r times backward elimination. However, the determination of the optimal values of (l, r) is a difficult problem.

EC Algorithms for Features Selection. EC techniques have been applied to address feature selection problems. Based on genetic algorithms (GAs), Chakraborty [3] proposes a feature selection algorithm using a fuzzy sets based fitness function. Kourosch and Zhang [12] propose a genetic programming based filter method as a multi-objective approach for feature selection in binary classification problems. Based on ant colony optimisation and fuzzy-rough theory, Jensen [8] proposes a filter feature selection method for web content classification and complex systems monitoring.

Unler and Murat [17] propose a PSO based feature selection algorithm with an adaptive selection strategy. Mohammed et al. [11] propose a hybrid method that

incorporates PSO with an AdaBoost framework to search for the best feature subset and determine the decision thresholds of AdaBoost simultaneously. Wang et al. [18] propose a filter feature selection algorithm based on an improved binary PSO and rough set. However, the feature subset is only tested on one learning algorithm, which can not show the advantage that filter algorithms are more general.

Most of the existing feature selection algorithms are single objective, wrapper approaches, which are computationally more expensive and less general than filter approaches. Meanwhile, the performance of the probabilistic rough set theory for feature selection has not been investigated in multi-objective feature selection. Therefore, the development of using PSO and probabilistic rough set for multi-objective feature selection is still an open issue.

3 Proposed Multi-objective Method

In this section, three feature selection algorithms [2] based on PSO and probabilistic rough set theory is firstly described, which are used as the baseline to test the performance of the proposed algorithm. Then we propose a multi-objective algorithm (MOPSOPRS) based on PSO and probabilistic rough set theory.

3.1 PSORS, PSOPRS and PSOPRSN

When using rough set theory for feature selection, a dataset can be regarded as an information system $I = (U, A)$, where all features can be considered as A in the rough set theory. Based on the equivalence described by A , U can be partitioned to $U_1, U_2, U_3, \dots, U_n$, where n is the number of classes in the dataset. After feature selection, the achieved feature subset can be considered as $P \subseteq A$. Therefore, the fitness of P can be evaluated by how well P represents each target set in U , i.e., a class in the dataset.

PSORS. In standard rough set theory, for $U_1 \subseteq U$ and $P \subseteq A$, $\underline{P}U_1 = \{x \in U \mid [x]_P \subseteq U_1\}$ is the lower approximation of P according to U_1 if $[x]_P$ only contains instances in U_1 . $\underline{P}U_1$ measures the number of instances that have been completely separated from instances of other classes. Therefore, how well P describes each target in U can be calculated by Equation 6, which is the fitness function in PSOPRS. A feature subset with $Fitness_1(P) = 1.0$ means this feature subset can completely separate each class from the other classes.

$$Fitness_1(P) = \frac{\sum_{i=1}^n |\underline{P}U_i|}{|U|} \quad (6)$$

PSOPRS. As discussed in Section 2.2, the definitions of lower approximation and upper approximation limit the application of standard rough set theory. Therefore, a filter feature selection algorithm (PSOPRS) based on PSO and probabilistic rough set theory was proposed in [2]. In probabilistic rough set

Algorithm 1. Pseudo-Code of MOPSOPRS

```

begin
  initialise the set of leaders LeaderSet and Archive
  calculate the crowding distance of each member in LeaderSet;
  while Maximum Iteration is not reached do
    for each particle do
      select a leader (gbest) from LeaderSet for each particle by using a
      binary tournament selection based on the crowding distance;
      update the velocity and the position of particle i;
      apply bit-flip mutation;
      evaluate two objective values of each particle;          /* number of
      features and  $Fitness_2(P)$  value of the feature subset) */
      update the pbest of each particle;
    identify the non-dominated solutions (particles) to update LeaderSet;
    send leaders to Archive;
    calculate the crowding distance of each member in LeaderSet;
  calculate the classification error rate of solutions in Archive on the test set;
  return the solutions in Archive and their training and test classification
  error rates;

```

theory, for the target set U_1 , $\mu_P[x] = \frac{|[x]_P \cap U_1|}{|[x]_P|}$. $\mu_P[x]$ quantifies the proportion of $[x]_P$ is in U_1 . $\underline{apr}_P U_1 = \{x | \mu_P[x] \geq \alpha\}$ defines the lower approximation of P according to U_1 rather than $\underline{P}U_1$. $[x]_P$ does not have to completely contained in U_1 . α can be adjusted to restrict or relax $\underline{apr}_P U_1$. When $\alpha = 1.0$, $\underline{apr}_P U_1 = \underline{P}U_1$. The fitness function of PSOPRS is shown by Equation 7.

$$Fitness_2(P) = \frac{\sum_{i=1}^n |\underline{apr}_P U_i|}{|U|} \quad (7)$$

PSOPRSN. PSOPRS using probabilistic rough set theory can avoid the problems caused by standard rough set, but the number of features is not considered in the fitness function. For the same α value, if there are more than one feature subsets that have the same fitness, PSOPRS does not intend to search for the smaller feature subset. Therefore, the number of features was added to the fitness function to form another algorithm (PSOPRSN) [2], which aims to maximise the representation power of the feature subset and also to minimise the number of features.

$$Fitness_3(P) = \gamma * \frac{\sum_{i=1}^n |\underline{apr}_P U_i|}{|U|} + (1 - \gamma) * \left(1 - \frac{\#features}{\#totalFeatures}\right) \quad (8)$$

where $\gamma \in (0, 1]$ shows the relative importance of the representation power while $(1 - \gamma)$ shows the relative importance of the number of features.

Table 1. Datasets

Dataset	#Features	#Classes	#Instances	Dataset	#Features	#Classes	#Instances
Spect	22	2	267	Dermatology	33	6	366
Soybean Large	35	19	307	Chess	36	2	3196
Statlog	36	6	6435	Waveform	40	3	5000

3.2 MOPSOPRS

PSOPRSN combines the two main objectives of feature selection into a single fitness function. However, γ needs to be predefined and its best value is problem-dependent. Therefore, we propose a multi-objective PSO based feature selection algorithm. However, PSO was originally proposed for single objective optimisation. Sierra and Coello [15] proposed a multi-objective PSO based on the ideas of mutation, crowding and dominance, which is a continuous algorithm and has achieved good performance. In this work, we extend it to a binary version of multi-objective PSO based on which we propose a multi-objective feature selection algorithm using probabilistic rough set theory (MOPSOPRS). The two objectives in MOPSOPRS is to maximise the representation power of the feature subset evaluated by $Fitness_2$ and to minimise the number of features.

Algorithm 1 shows the pseudo-code of MOPSOPRS. To select a *gbest* for each particle, MOPSOPRS employs a leader set to store the non-dominated solutions as the potential leaders. A crowding factor is employed to decide which non-dominated solutions should be added into the leader set and kept during the evolutionary process. A binary tournament selection is used to select two solutions from the leader set and the less crowded solution is chosen as the *gbest*. The maximum number of non-dominated solutions in the leader set is usually set as the same as the population size. In order to keep the diversity of the swarm and improve the search ability of the algorithm, MOPSOPRS randomly divides the whole swarm into three different groups in the initialisation procedure. The first group does not have any mutation. The second group employs uniform mutation to keep the global search ability and the third group employs non-uniform mutation to keep the local search ability. Furthermore, the three groups have the same leader set, which allows them to share their success to take advantages of different behaviors to search for the Pareto non-dominated solutions.

In all the algorithms, the dimensionality of the search space is the total number of features. Each particle is encoded in a binary string, where the “1” means the corresponding feature is selected, otherwise the feature is removed.

4 Experimental Design

As rough set theory only works on discrete values, six categorical datasets (Table 1) of varying difficulty are chosen from UCI machine learning repository [5] to test the performance of the algorithms. In each dataset, 70% of the instances are randomly chosen as the training set and others (30%) are the test set.

Table 2. Results of PSOPRS and PSOPRS with DT as the learning algorithm

Dataset	Spect			Dermatology			Chess		
Method	Size	Ave(Min)	Std	Size	Ave(Min)	Std	Size	Ave(Min)	Std
All	22	19.1		33	17.21		36	1.5	
PSORS	17.5	19.03(15.73)	2.28	21	13.99(2.46)	4.76	30.8	1.68(1.31)	0.261
PSOPRS									
$\alpha = 0.9$	17.3	19.44(15.73)	2.21	21	13.99(2.46)	4.76	30.7	1.6(1.31)	0.221
$\alpha = 0.8$	17.5	19.96(17.98)	1.96	21	13.99(2.46)	4.76	29.97	1.72(1.5)	0.279
$\alpha = 0.75$	15.57	18.2(17.98)	0.841	21	13.99(2.46)	4.76	30.3	1.53(1.31)	0.129
$\alpha = 0.5$	16.6	19.96(15.73)	2.11	20.73	13.99(2.46)	5.07	28.8	1.9(1.31)	0.525

All the algorithms firstly run on the training set to select a feature subset(s). The classification performance of the selected feature subset(s) will be evaluated by a learning/classification algorithm on the unseen test set. To test the claim that filter feature selection methods are general, three different learning algorithms, DT, naïve bayes (NB) and KNN with $K=5$ (5NN), are used in the experiments.

In all algorithms, the fully connected topology is used, the maximum velocity $v_{max} = 6.0$, the population size is 30 and the maximum iteration is 500. $w = 0.7298$, $c_1 = c_2 = 1.49618$. These values are chosen based on the common settings in the literature [14]. The algorithm has been conducted for 30 independent runs on each dataset. In PSOPRS, five different α values (1.0, 0.9, 0.8, 0.75, 0.5) are used in the experiments. When $\alpha = 1$, PSOPRS is the same as PSORS. Therefore, the results of $\alpha = 1$ in PSOPRS is not presented in Section 5. In PSOPRSN, the results of $\gamma = 0.9$ and $\gamma = 0.5$ are used to compare with that of the multi-objective algorithm (MOPSOPRS).

To further examine the performance of MOPSOPRS, two conventional filter feature selection methods (CfsF and CfsB) [7] implemented in Weka [20] are used for comparison and the classification performance is calculated by DT.

5 Experimental Results and Discussions

5.1 Experimental Results of PSORS and PSOPRS

Experiments about PSORS and PSOPRS have been conducted on the six datasets and DT, NB and 5NN were used for classification on the test sets. Due to the page limit, only the results of three datasets (Spect, Dermatology and Chess) using DT for classification are presented in Table 2. In the table, “All” means that all of the available features are used for classification. “Size” means the average number of features selected in the 30 independent runs. “Ave”, “Min” and “Std” represent the average, the lowest and the standard deviation of the classification error rates achieved by DT across the 30 runs.

Results of PSORS. According to Table 2, in most cases, PSORS selected feature subsets, which included around two thirds of the available features and achieved similar classification performance to all features. In almost all datasets, the best classification performance of PSOPRS (Min) is better than all features. The results suggest that PSORS based on PSO and standard rough set theory can be successfully used for feature selection.

Results of PSOPRS. According to Table 2, in most cases, PSOPRS with different α can achieve similar classification performance to all features. The number of features generally decreases when α reduces. Meanwhile, the best results achieved by PSOPRS are always better than all features in all cases. The results suggests that by using probabilistic rough set for feature selection, PSOPRS can further reducing the number of features without reducing its classification performance. A smaller α means more relax on the lower and upper approximations, which usually can slightly remove more unnecessary features to further reduce dimensionality of the datasets.

Note that considering *all* experimental results on PSOPRS (not only the results in Table 2), in most cases, $\alpha = 0.75$ achieved better classification performance than other α values. Therefore, $\alpha = 0.75$ is used in the experiments in PSOPRSN and MOPSOPRS.

5.2 Experimental Results of PSOPRSN and MOPSOPRS

PSOPRSN obtains a single solution in each of the 30 independent runs. MOPSOPRS obtains a set of non-dominated solutions in each run. To compare these two kinds of results, the 30 solutions (from 30 runs) resulted from PSOPRSN are presented in this section. 30 sets of feature subsets achieved by MOPSOPRS are firstly combined into one union set. In the union set, for the feature subsets including the same number of features (e.g. m), their classification error rates are averaged. Therefore, a set of average solutions is obtained by using the average classification error rates and the corresponding numbers (e.g. m). The set of average solutions is called the *average* Pareto front and presented here. Meanwhile, the non-dominated solutions in the union set are called the *best* Pareto front and are also presented to compare with the solutions achieved by PSOPRSN.

Figure 1 shows the experimental results of MOPSOPRS and PSOPRSN with $\gamma = 0.5$ and $\gamma = 0.9$ on the test sets, where DT was used as the classification algorithm. In the figure, each chart corresponds to one of the dataset used in the experiments. On the top of each chart, the numbers in the brackets show the number of available features and the classification error rate using all features. In each chart, the horizontal axis shows the number of features selected and the vertical axis shows the classification error rate. As the results of using NB and 5NN are similar to that of using DT, the results of using NB and 5NN are not presented here due the page limit.

In Figure 1, “AvePar” and “BestPar” stand for the *average* and the *best* Pareto fronts resulted from MOPSOPRS in the 30 independent runs. $\gamma = 0.5$ and $\gamma = 0.9$ show the results of PSOPRSN with $\gamma = 0.5$ and $\gamma = 0.9$, respectively. In some datasets, PSOPRSN may evolve the same feature subset in different runs and they are shown in the same point in the chart. Therefore, although 30 results are presented in $\gamma = 0.5$ and $\gamma = 0.9$, there may be less than 30 distinct points shown in one chart.

Results of PSOPRSN. As can be seen from Figure 1, in most cases, PSOPRSN with both $\gamma = 0.5$ and $\gamma = 0.9$ selected feature subsets with a smaller

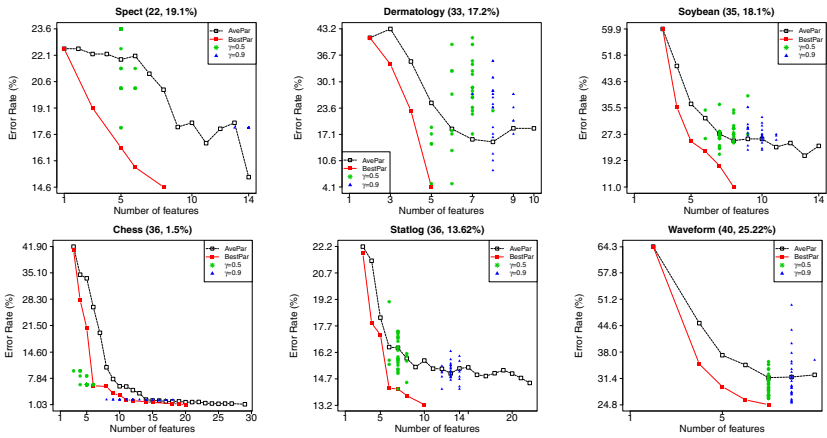


Fig. 1. Results of MOPSOPRS and PSOPRSN on test sets evaluated by DT

number of features and achieved similar or even better classification performance than using all features. $\gamma = 0.9$ achieved similar or better classification than $\gamma = 0.5$ and $\gamma = 0.5$ usually achieved a smaller number of features than $\gamma = 0.9$. The reason is that the number of features is assigned more important in $\gamma = 0.5$ than in $\gamma = 0.9$.

Results of MOPSOPRS. According to Figure 1, in three of the six datasets, the average Pareto front of MOPSOPRS (AvePar) includes two or more solutions, which selected a smaller number of features and achieved a similar or even lower classification error rate than using all features. For the same number of features, there are a variety of combinations of features with different classification performances. The feature subsets obtained in different runs may include the same number of features but different classification error rates. Therefore, although the solutions obtained in each run are non-dominated, some solutions in the average Pareto front may dominate others. This also happens when using 5NN or NB as the classification algorithms.

According to Figure 1, in *all* datasets, the non-dominated solutions in MOPSOPRS (BestPar) selected one or more feature subsets, which included less than one third of features and achieved better classification performance than using all features.

Comparisons Between MOPSOPRS and PSOPRSN. In most datasets, solutions in AvePar achieved similar results to both $\gamma = 0.5$ and $\gamma = 0.9$ in terms of both the number of features and the classification performance, but AvePar included more different sizes of feature subsets. In five of the six datasets, BestPar achieved better classification performance with a smaller number of features than both $\gamma = 0.5$ and $\gamma = 0.9$, especially in the datasets with a large number of features, such as the Statlog and Waveform datasets.

Figure 1 shows that MOPSOPRS can further reduce the number of features and increase the classification performance, which indicates that MOPSOPRS

Table 3. Results of CfsF and CfsB with DT as the learning algorithm

Dataset	Spect		Dermatology		Soybean		Chess		Statlog		Waveform	
Method	Size Error (%)		Size Error (%)		Size Error (%)		Size Error (%)		Size Error (%)		Size Error (%)	
CfsF	4	30	17	12.73	12	19.51	5	21.9	5	28.38	32	28
CfsB	4	30	17	12.73	14	14.63	5	21.9	5	28.38	32	28

as a multi-objective technique can explore the search space of a feature selection problem better than the single objective algorithm, PSOPRSN.

Note that the results of using the three classification algorithms (DT, NB and 5NN) show that the performance of MOPSOPRS and PSOPRSN are consistent when using different classification algorithms, which suggests the proposed filter algorithm with probabilistic rough set as the evaluation criterion are general to these three classification algorithms.

5.3 Comparisons with Two Traditional Algorithms

Table 3 shows the results of CfsF and CfsB for feature selection, where DT was used for classification. Comparing MOPSOPRS (results of using DT shown in Figure 1) with CfsF and CfsB, it can be seen that in *all* datasets, MOPSOPRS (BestPar) outpermed both CfsF and CfsB in terms of the classification performance and the number of features.

6 Conclusions

This work conducted the first study on PSO and rough set theory for multi-objective feature selection. We proposed a novel feature algorithm (MOPSOPRS) based on a multi-objective PSO and probabilistic rough set theory with the goal of obtaining a set of non-dominated feature subsets, which reduced the number of features and achieved similar or even better classification performance than all features. MOPSOPRS was examined and compared with three single objective algorithms (PSOPRS, PSOPRS and PSOPRSN) and two traditional methods on six datasets of varying difficulty. DT NB and 5NN were used to test the generality of MOPSOPRS. Experimental results show that in almost all cases, MOPSOPRS can automatically evolve a set of non-dominated feature subsets that include a smaller number of features and achieve better classification performance (evaluated by the three classification methods) than using all features. MOPSOPRS outperformed the three PSO based single objective and the two traditional algorithms in terms of both the number of features and the classification performance. The results also show that MOPSOPRS are general to the three different classification algorithms. This study finds that as a multi-objective algorithm, MOPSOPRS can search the solution space effectively to obtain a set of non-dominated solutions instead of a single solution. Examining the Pareto front achieved by the multi-objective algorithm can assist users in choosing their preferred solutions to meet their own requirements.

In future, we will further investigate the use of multi-objective PSO and probabilistic rough set for feature selection to better explore the Pareto front of non-dominated solutions in feature selection problems.

References

1. Almuallim, H., Dietterich, T.G.: Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence* 69, 279–305 (1994)
2. Cervante, L., Xue, B., Shang, L., Zhang, M.: A Dimension Reduction Approach to Classification Based on Particle Swarm Optimisation and Rough Set Theory. In: Thielscher, M., Zhang, D. (eds.) *AI 2012. LNCS*, vol. 7691, pp. 313–325. Springer, Heidelberg (2012)
3. Chakraborty, B.: Genetic algorithm with fuzzy fitness function for feature selection. In: *International Symposium on Industrial Electronics*, vol. 1, pp. 315–319 (2002)
4. Dash, M., Liu, H.: Feature selection for classification. *Intelligent Data Analysis* 1(4), 131–156 (1997)
5. Frank, A., Asuncion, A.: *UCI machine learning repository* (2010)
6. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. *The Journal of Machine Learning Research* 3, 1157–1182 (2003)
7. Hall, M.A.: *Correlation-based Feature Subset Selection for Machine Learning*. Ph.D. thesis, The University of Waikato, Hamilton, New Zealand (1999)
8. Jensen, R.: Performing Feature Selection with ACO. In: Abraham, A., Grosan, C., Ramos, V. (eds.) *Swarm Intelligence in Data Mining. SCI*, vol. 34, pp. 45–73. Springer, Heidelberg (2006)
9. Kennedy, J., Eberhart, R.: A discrete binary version of the particle swarm algorithm. In: *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, pp. 4104–4108 (1997)
10. Marill, T., Green, D.: On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory* 9(1), 11–17 (1963)
11. Mohemmed, A., Zhang, M., Johnston, M.: Particle swarm optimization based adaboost for face detection. In: *IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 2494–2501 (2009)
12. Neshatian, K., Zhang, M.: Pareto front feature selection: using genetic programming to explore feature space. In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA, pp. 1027–1034 (2009)
13. Pawlak, Z.: Rough sets. *International Journal of Parallel Programming* 11, 341–356 (1982)
14. Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: *IEEE International Conference on Evolutionary Computation (CEC 1998)*, pp. 69–73 (1998)
15. Sierra, M.R., Coello Coello, C.A.: Improving PSO-Based Multi-objective Optimization Using Crowding, Mutation and ϵ -Dominance. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) *EMO 2005. LNCS*, vol. 3410, pp. 505–519. Springer, Heidelberg (2005)
16. Stearns, S.: On selecting features for pattern classifier. In: *Proceedings of the 3rd International Conference on Pattern Recognition*, pp. 71–75 (1976)
17. Unler, A., Murat, A.: A discrete particle swarm optimization method for feature selection in binary classification problems. *European Journal of Operational Research* 206(3), 528–539 (2010)
18. Wang, X., Yang, J., Teng, X., Xia, W., Jensen, R.: Feature selection based on rough sets and particle swarm optimization. *Pattern Recognition Letters* 28(4), 459–471 (2007)
19. Whitney, A.: A direct method of nonparametric measurement selection. *IEEE Transactions on Computers* C-20(9), 1100–1103 (1971)
20. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn. Morgan Kaufmann (2005)
21. Yao, Y., Zhao, Y.: Attribute reduction in decision-theoretic rough set models. *Information Sciences* 178(17), 3356–3373 (2008)

A New Crossover for Solving Constraint Satisfaction Problems

Reza Abbasian and Malek Mouhoub

Department of Computer Science
University of Regina
Regina, Canada
{abbasiar,mouhoubm}@cs.uregina.ca

Abstract. In this paper we investigate the applicability of Genetic Algorithms (GAs) for solving Constraint Satisfaction Problems (CSPs). Despite some success of GAs when tackling CSPs, they generally suffer from poor crossover operators. In order to overcome this limitation in practice, we propose a novel crossover specifically designed for solving CSPs. Together with a variable ordering heuristic and an integration into a parallel architecture, this proposed crossover enables the solving of large and hard problem instances as demonstrated by the experimental tests conducted on randomly generated CSPs based on the model RB. We will indeed demonstrate, through these tests, that our proposed method is superior to the known GA based techniques for CSPs. In addition, we will show that we are able to compete with the efficient MAC-based Abscon 109 solver for random problem instances.

Keywords: Parallel Genetic Algorithms, Constraint Satisfaction Problem (CSP), Evolutionary Techniques.

1 Introduction

Many real life applications under constraints can be efficiently represented and solved through a Constraint Satisfaction Problem (CSP). More formally, a CSP consists of a finite set of variables with finite domains, and a finite set of constraints restricting the possible combinations of variable values [2]. A solution tuple to a CSP is a set of assigned values to variables that satisfy all the constraints. A binary CSP is a CSP where each constraint involves at most two variables. A binary CSP is often represented by a graph where vertices correspond to variables while edges represent the constraints between these variables. In this paper we focus on the case of binary CSPs using the graph representation.

A CSP is known to be an NP-complete problem in general¹, a backtrack search algorithm of exponential time cost is needed to find a complete solution. In order to overcome this difficulty in practice, systematic solving methods based on constraint propagation techniques have been proposed in the literature [2]. The

¹ There are special cases where CSPs are solved in polynomial time, for instance, the case where the related constraint network is a tree [2].

goal here is to reduce the size of the search space before and during the backtrack search. While these proposed techniques have a lot of merits when tackling small and medium size problems, their combination with the backtracking algorithm suffers from the exponential time cost of this latter especially for large size problems. An alternative is to use approximation methods such as Genetic Algorithms (GAs). Despite some success of GAs when tackling CSPs [3,5,6], they generally suffer from poor crossover operators in solving constraint problems. The main reason for such a phenomenon is that in CSPs, changing the value of a variable can have direct effects on other variables that are in constraint relation with the changing variable and indirect effect on other variables. As a result, performing a random crossover can often reduce the quality of the solution.

In this paper we propose a novel crossover, that we call Parental Success Crossover (PSC), specially designed for solving CSPs. In order to assess the performance of our proposed crossover over the basic one-point crossover, Asexual Crossover for CSP (ASXC), Multi-Parent Crossover (MPC) as well as known heuristic based GAs for CSPs [3,6], we conducted several experiments on CSP instances randomly generated using the model RB proposed in [17]. This model is a revision of the standard Model B [16], has exact phase transition and the ability to generate asymptotically hard instances. The test results clearly demonstrate that our proposed crossover outperforms the other known GA methods on all problem instances in terms of success rate and time needed to reach the solution. In addition, we evaluated the performance of an integration of our crossover, together with a variable ordering heuristic [13], within a proposed Hierarchical Parallel GA (HPGA). The results are very appealing especially when dealing with hard instances. Moreover, our HPGA method is able to compete with the efficient MAC-based Abscon solver [7] as demonstrated by the comparative experiments conducted on random CSPs with different sizes. Finally, our GA based solving method incorporates many greedy principles and has the ability to solve a CSP by giving a solution with a quality (number of solved constraints) depending on the time allocated for computation. This is clearly demonstrated through several tests we conducted both on consistent and inconsistent random CSPs. This “anytime behaviour” is very relevant in practice especially for real time applications where a solution needs to be returned within a given deadline. The user can, for instance, get a solution with a given quality at a particular time point or let the program run for another amount of time, if this can be afforded, to get a better quality.

The rest of the paper is structured as follows. A literature review on Parallel GAs and crossovers is introduced in the next section. Our proposed crossover and its integration into the Parallel GA is then covered in section 3. Section 4 reports the results of comparative experiments we conducted on randomly generated CSPs. Finally, concluding remarks and future directions are listed in section 5.

2 Background

2.1 Parallel Genetic Algorithms

Genetic Algorithms (GAs) [4] are evolutionary algorithms based on the idea of natural selection and evolution. GAs have been successfully applied to a wide variety of problems. In GAs, there is a population of potential solutions called individuals. The GA performs different genetic operations on the population, until the given stopping criteria are met. The Parallel Genetic Algorithm (PGA) is an extension of the GA. The well-known advantage of PGAs is their ability to facilitate different sub populations to evolve in diverse directions simultaneously [8]. It is shown that PGAs speed up the search process and can produce high quality solutions on complex problems [9,15]. There are mainly three different types of PGAs [1]. First, the Master-Slave PGA (MSPGA) in which, there is only one single population divided into fractions. Each fraction is assigned to one slave process on which genetic operations are performed [8]. Second, the Multi-Population PGA which contains a number of sub populations, which can occasionally exchange individuals. The exchange of individuals is called migration. Migration is controlled using several parameters. Multi-population PGAs are also known as Island PGAs (IPGAs), since they resemble the “island model” in population genetics that considers relatively isolated demes. Finally, the Fine-Grained PGA which consists of only one single population, that is designed to run on closely linked massively parallel processing systems. In this paper we use the Master-Slave architecture for designing the PGA.

2.2 Crossovers

Several crossovers have been proposed in the literature. The simplest one is the One Point Crossover (OPC) that works as follows. First, we randomly choose a crossing point. All the genomes (individuals) from the first parent that are before the crossing point will be included in the offspring. In addition, all the genomes in the second parent that are after the crossing point, will be included as well in the offspring. The Asexual Crossover for CSPs (we refer to it as ASXC) is suggested by Eiben et al [3,5]. The Idea here is as follows. To produce the offspring we use an asexual crossover that picks a group of variables having the largest number of violated constraints and changes their value such that the number of violations (or conflicts) are minimized. The group size should be determined at the beginning, but was suggested 1/4 of the individual size [3,5]. As stated in [3,5], the Multi-Parent Crossover for CSPs operates by scanning the genes of the parents consecutively. It then chooses a value that has occurred the most for that gene (CSP variable) amongst the parents and assigns it to the corresponding gene of the offspring. The value selection can also be performed randomly or based on the fitness of the parents.

3 Proposed Crossover within a PGA

3.1 Individual Representation

In GAs each possible solution to the given problem is represented using an encoding known as the chromosome (or individual). Each chromosome contains a number of genes (or *alleles*). To represent a possible solution to a CSP in GAs, we normally use an array structure. Each index of the array is considered as a gene that corresponds to a variable in the CSP. The data that each gene carries is from the domain value of the variable it represents. Moreover, each individual has a fitness corresponding to the total number of constraint violations. An individual with a fitness equal to zero is a solution to the problem.

3.2 Parental Success Crossover (PSC)

The random crossover performs poorly for constraint problems. In these problems, changing the value of a variable X has direct effect on other variables that are in constraint relation with X . It can also have indirect effects on other variables. As a result, a random crossover (having a random crossing point) does not provide interesting results. To improve the crossover for CSPs, we propose the following crossing method. Each individual in the population maintains two records; the total number of times it has participated in reproduction (N_p), and the number of times the offspring it produced was fitter. We refer to the latter as Parental Success Number and denote it by P_s . The parental success ratio, denoted by S , then can be calculated as follows: $S = \frac{P_s}{N_p}$. Furthermore, we define the term Fitness Around Variable (FAV) as the number of conflicts between a given variable and its neighbors in the constraint graph. Each individual keeps a record of the fitness around all of its variables. Using these new parameters, we create a Crossover Mask. The offspring is then produced according to this mask. Let p_1 and p_2 be two parents chosen for the crossover. The crossover mask specifies which allele in the new chromosome should inherit from which parent. Since the crossover is performed here using two parents, the crossover mask consists of binary digits. Let 1 specify choosing the allele from p_1 and 0 specify choosing it from p_2 . To create the mask, we compare each allele in p_1 with its correspondence in p_2 . If the FAV of the allele in p_1 is less than the one in p_2 , we put a 1 in the mask. If the FAV of the allele in p_2 is less than the one in p_1 , we put a 0 in the mask. In case of equality, we use the following probabilities for choosing the allele: $P(\text{choosing from } p_1) = 1/2 + (S_{p_1} - S_{p_2}) \times 1/2$

$P(\text{choosing from } p_2) = 1/2 + (S_{p_2} - S_{p_1}) \times 1/2$ where, S_{p_1} and S_{p_2} are respectively the parental success ratios of p_1 and p_2 . Algorithm 1 describes the procedure for generating the PSC Mask.

3.3 Reproduction

Reproduction is performed amongst a number of fittest individuals in the population. To generate new offsprings, we randomly chose two individuals among the

Algorithm 1. Generating PSC Mask

```

function PSC-MASK( $FAV_{p_1}, FAV_{p_2}$  as Arrays and  $S_{p_1}, S_{p_2}$  as Doubles)
  Define:  $mask$  as Array
  Define:  $P_{p_1}$  as Double ▷ probability of choosing from  $p_1$ 
   $P_{p_1} = 0.5 + (S_{p_1} - S_{p_2}) \times 0.5$ 
  for  $i = 0$  to  $individualLength$  do
    if  $FAV_{p_1}[i] < FAV_{p_2}[i]$  then
       $mask[i] = 1$ 
    else if  $FAV_{p_1}[i] > FAV_{p_2}[i]$  then
       $mask[i] = 0$ 
    else
      if  $U(0, 1) \leq P_{p_1}$  then ▷  $U$  is uniform random function
         $mask[i] = 1$  ▷ choose from  $p_1$ 
      else
         $mask[i] = 0$  ▷ choose from  $p_2$ 
      end if
    end if
  end for
  return  $mask$ 
end function

```

fittest ones in the population as the parents. We then pass them to the Parental Success Crossover. Also, every I iterations, we pick the parents totally random as a means to preserve the diversity in the population.

3.4 Mutation

We propose two different methods for the mutation. The first method, that we call *mutation to minimize the number of conflicts*, is used to locally improve the solution while performing the mutation. The second method, that we call *stochastic value change*, performs a complete random mutation. Since *mutation to minimize the number of conflicts* has a greedy strategy to modify the individuals, using *stochastic value change* is necessary to preserve the diversity of the population and to avoid being trapped in a local minimum. The details of these two methods are as follows.

Mutation to Minimize the Number of Conflicts. Given a constraint graph representing a CSP, $N_{mutation}$ random vertices of the individual are selected and the numbers of conflicts between the chosen vertices and their adjacent vertices are minimized. Say vertex A is randomly chosen for the mutation. Then, according to the adjacency matrix of the constraint graph, for every vertex B that is adjacent to A , if there is a conflict between A and B , B will take a new random value that is consistent to A 's value. $N_{mutation}$ is computed as follows using another parameter that we introduce called Allele Mutation Percentage. Suppose we have an individual of size 100 and an allele mutation percentage

of 20%. Then we have $20\% \cdot 100 = 20$. This number 20 is now the maximum possible value for $N_{mutation}$. Each time we perform a mutation, $N_{mutation}$ takes a random value between 1 and 20. Algorithm 2 presents the pseudocode of this method.

Algorithm 2. Mutation to Minimize the Number of Conflicts

function MUTATE1(*indiv* as **Array of Integers**)

Define: A as **Integer**

▷ a vertex index of individual

for $i = 0$ **to** $N_{mutation}$ **do**

$A \leftarrow$ a unique, randomly chosen vertex index.

for all B **adjacent to** A **do**

if $indiv[A] == indiv[B]$ **then**

$indiv[B] \leftarrow$ random value consistent with A 's value

end if

end for

end for

end function

Stochastic Value Change. Algorithm 3 presents the pseudocode of this method. Here, we randomly choose $N_{mutation}$ vertices and assign a random value to each.

Algorithm 3. Stochastic Value Change

function MUTATE2(*indiv* as **Array of Integers**)

Define: A as **Integer**

▷ a vertex index of individual

for $i = 0$ **to** $N_{mutation}$ **do**

$A \leftarrow$ a unique, randomly chosen vertex index.

$indiv[A] \leftarrow$ random value

end for

end function

3.5 The Genetic Modification (GM) Operator

We extend the PGA with an additional operator that we propose, namely the Genetic Modification (GM). The GM operator runs concurrently with the PGA and generates good individuals outside the scope of the GA. The PGA then incorporates these newly generated, near optimal individuals to its population to give them a chance to participate in reproduction. The idea here is that the GM concurrently and independently operates beside the PGA. Whenever the GM produces a population of individuals, the PGA keeps them until the next reproduction. Then, just before the reproduction, the PGA distributes them between

the sub populations. The GM uses a variable ordering for individual generation. Whenever the GM needs to create a new individual, it starts from the first variable in the ordering and generates a random value for each variable in turn. Following a look ahead principle, when a variable is assigned a value, the GM propagates this change by removing the values that would result in conflicts from the domain of its neighbors. This way, it is guaranteed that at each time, the chosen value for a variable will not cause a conflict. However, at the end of initializing variables, we might end up with some variables that have empty domains. In this case, the GM randomly chooses a value for them. The GM generates P_{GM} individuals and signals the PGA's master process. The master process will then distribute the generated individuals amongst the sub-population for the next reproduction. Algorithm 4 lists this process.

Algorithm 4. Genetic Modification Process

function GM(*ordering* as **Array of Integers**)

 Define: P as **Set of Individuals**

 Define: *indiv* as **Array of Integers**

$P \leftarrow \emptyset$

for $i = 0$ **to** P_{GM} **do**

indiv \leftarrow new individual based on *ordering*

$P \leftarrow P \cup \textit{indiv}$

end for

 Notify *master process* of P

end function

We used the heuristic proposed in [13] for variable ordering. This heuristic is based on Hill Climbing (HC) for weighing constraints and works as a Static Variable Ordering (SVO) algorithm as it runs prior to the actual backtrack search algorithm. More precisely, HC is run for a given number of cycles, during which, the constraints gain weight. After this information gathering phase, each variable gets a weighted degree, which is the sum of the weights of the constraints that the variable is involved in. Variables are then sorted based on their weights and those with larger weight get more priority in the ordering. We converted this heuristic into a Dynamic Variable Ordering (DVO) one by adapting it into each of the PGAs' master process. The idea is that HC will operate as part of each PGA and continues to run through the whole runtime of the PGAs. At the end of each k generations, the master process picks the best solution found by the slaves and calculates new weights for the constraints. It then creates an ordering based on the weights and passes it to the GM. The GM would then use this new ordering to create new individuals. The integration of our DVO into PGA's master process is described in Algorithm 6.

3.6 Hierarchical PGA (HPGA)

A Hierarchical PGA (HPGA) can be obtained by any combination of the PGA types. Figure 1 shows the architecture of our proposed HPGA. To design the HPGA, we use the Island PGA (IPGA) for the top level and Master-Slave PGA (MSPGA) for the lower level. Each MSPGA, is actually an island of the IPGA. However, there is a Coordinator Process (CP) in the IPGA which is in charge of assigning different CSP problems to each island of the IPGA. The CP can communicate with each MSPGA using the chosen Inter-Process Communication (IPC) technique. Consider M as the total number of MSPGAs, $M_{suspended}$ as the number of MSPGAs in the *suspend* state. Algorithm 5 and Algorithm 6 respectively illustrate the procedures for IPGA and MSPGAs.

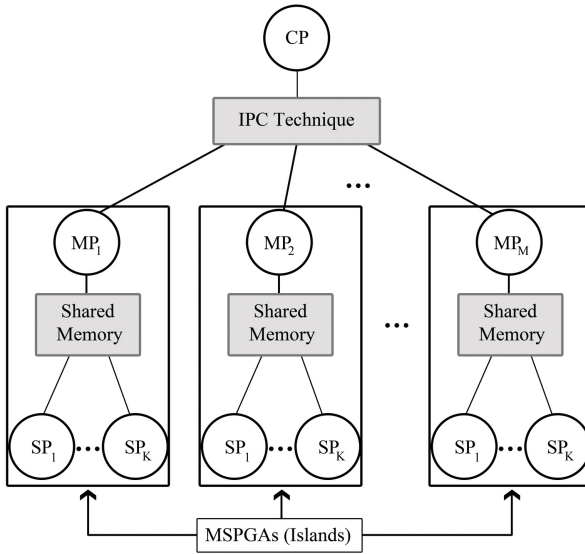


Fig. 1. Architecture of the HPGA

Algorithm 5. IPGA Algorithm (CP) for MSPGAs

Require: All MSPGAs are suspended at the beginning.

- 1: Assign to each MSPGA, a CSP.
 - 2: Start suspended MSPGAs.
 - 3: Wait for a MSPGA to find a solution. If a solution is found halt the algorithm and return the solution. Meanwhile, if the Stopping Criteria are satisfied, stop the algorithm and return the best result so far.
-

Algorithm 6. MSPGA Integrated with HC

- 1: *generationNumber* = 0
 - 2: **In Parallel:** generate a random population of size P . Calculate the fitness of each individual.
 - 3: If a solution is found (an individual with zero conflicts), signal the CP and wait for a task from the CP. Else, go to the next step.
 - 4: **if** (*generationNumber mod k* = 0) **then**
 - 5: Pick the best solution found by slaves.
 - 6: Calculate new weights for constraints (based on HC).
 - 7: Create a new ordering based on new weights.
 - 8: Pass the new ordering to GM.
 - 9: **end if**
 - 10: Before entering the reproduction, check if the GM process has created a modified population. If so, distribute them amongst sub populations.
 - 11: **In Parallel:** perform reproduction, mutation, and fitness calculation.
 - 12: Increment *generationNumber* by 1. Go to step 3.
-

4 Experimentation

The algorithms are implemented in Java programming language. In the experiments we used a machine with 2.5 GHz Core 2 Duo CPU, 4 GB of RAM running JDK 1.6. The mutation is implemented as described in Section 3.4. The number of variables to change is also determined randomly from the following range $[2, individualLength/10]$ where *individualLength* is the length of each individual. The mutation chance is set to 0.2. For all the tests reported in this section, each problem instance is solved 20 times by the given method and the average running time needed to return the solution is computed.

Following the model RB [17], we generate each CSP instance in two steps as shown below and using the parameters n , p , α and r where :

- n is the number of variables,
 - p ($0 < p < 1$) is the constraint tightness which can be measured, as shown in [14], as the fraction of all possible pairs of values from the domain of two variables that are not allowed by the constraint,
 - and r and α ($0 < \alpha < 1$) are two positive constants (respectively set to 0.5 and 0.8).
1. Select with repetition $rn \ln n$ random constraints. Each random constraint is formed by selecting without repetition 2 of n variables.
 2. For each constraint we uniformly select without repetition pd^2 incompatible pairs of values from the domains of the pair of variables involved by the constraint. $d = n^\alpha$ is the domain size of each variable.

First, we compared our proposed PSC against the One Point Crossover (OPC), the Asexual Crossover (ASXC) and the Multi-Parent Crossover (MPC) proposed in [3]. The population size is fixed here to 2000 and the mutation chance to

0.2. Table 1 (left sub table) shows the results of running these methods with mutation. Consistent CSP instances are randomly generated with 100 variables and tightness (T) values ranging from 0.05 to 0.6. For each test we report the running time (in seconds) together with the Success Rate, SR (for reaching a complete solution) and the best fitness, BF (number of violated constraints, in the case where a complete solution is not obtained). When the timeout (60 seconds) is reached for a particular case, “-” is displayed. Note that for the ASXC method, the time is not reported in table 1 as this method fails to solve all the problem instances in the allocated runtime limit in the case where mutation is used. From the two tables it is obvious to see that our PSC based GA method is the only one that is successful for solving under constrained and middle constrained problems (when the tightness is less than 0.4). Moreover, for hard instances (tightness between 0.4 and 0.6) our method returns better quality solutions (BF values) than all the other techniques.

The parallelism provides the ability to investigate different regions of the search space simultaneously. In order to assess the performance of the integration of our PSC within the HPGA, we conducted more tests on consistent CSPs as follows. As we mentioned in section 2.1, our PGA is implemented using the Master-Slave architecture. Within the HPGA, two islands (IPGAs) are used with the number of slaves for each fixed to 10 and the population size per slave equal to 200. Table 1 (right sub table) reports the results of the tests we conducted on the same instances used for the sequential methods. We evaluate the performance of two methods: our PSC within the HPGA (HPGA+PSC) and the PSC with the proposed Genetic Modification within the HPGA (HPGA+GM+PSC). It is clear from Table 1 that the parallelization of our method allows us to successfully solve all the instances up to 0.5 tightness value without the GM and up to 0.55 when GM is used. In addition, with GM we can even solve the hardest instances (tightness of 0.6) with a high success rate.

Table 1. Comparing Different Crossovers in a Sequential and Parallel GA

T	MPC		OPC		PSC		HPGA+PSC		HPGA+GM+PSC	
	SR,BF	Time	SR,BF	Time	SR,BF	Time	(SR,BF)	Time (s)	(SR,BF)	Time (s)
0.05	100%,0	2.448	100%,0	2.57	100%,0	1.88	100%,0	0.31	100%,0	0.24
0.1	34%,0	57.13	100%,0	11.46	100%,0	2.11	100%,0	0.42	100%,0	0.26
0.15	0,4	-	100%,0	16.43	100%,0	3.18	100%,0	0.77	100%,0	0.40
0.2	0,9	-	27%,0	51.03	100%,0	4.1	100%,0	0.97	100%,0	0.43
0.25	0,17	-	0,3	-	100%,0	6.92	100%,0	1.09	100%,0	0.47
0.3	0,25	-	0,5	-	100%,0	7.26	100%,0	1.14	100%,0	0.49
0.35	0,28	-	0,10	-	100%,0	13.86	100%,0	1.28	100%,0	0.50
0.4	0,37	-	0,12	-	62%,0	44.56	100%,0	1.54	100%,0	0.51
0.45	0,45	-	0,27	-	0,5	-	100%,0	1.72	100%,0	0.52
0.5	0,53	-	0,30	-	0,9	-	100%,0	2.03	100%,0	0.56
0.55	0,69	-	0,42	-	0,15	-	73%,0	37.46	100%,0	1.72
0.6	0,78	-	0,62	-	0,17	-	0,3	-	79%,0	31.29

Figure 2 reports the results of comparative tests between our HPGA+GM+PSC method and Abscon 109 [7] for solving randomly generated consistent CSPs with tightness equal to 0.35 and the number of variables varying from 100 to 1000. In the case of HPGA+GM+PSC, we have used the following parameters tuned to their best values. For instances with less than 600 variables, the number of islands is 4 with 5 slaves per island and 100 individuals per slave. For instances with 600 and more variables, the number of islands is 8 with 5 slaves per island and 100 individuals per slave. From figure 2 it is clearly shown that our method is better than Abscon 109 for all problem instances in terms of running time needed to return a complete solution.

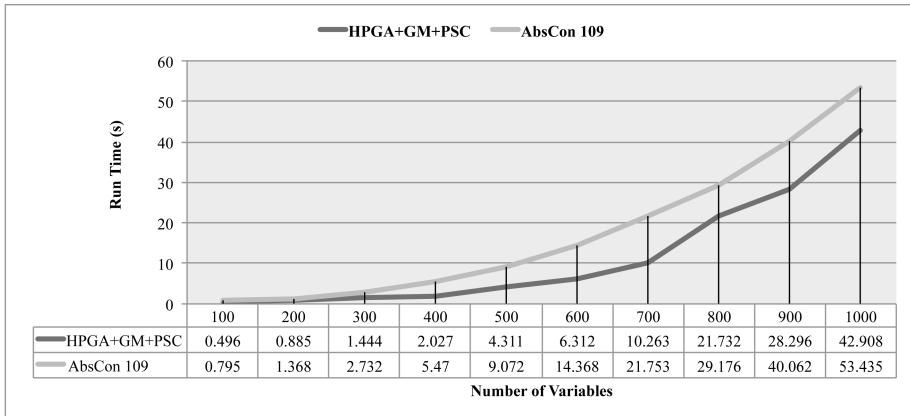


Fig. 2. Running Time Comparison of HPGA+GM versus AbsCon 109 on Randomly Generated Instances

5 Conclusion and Future Work

In order to overcome the difficulty when solving CSPs, we have proposed in this paper a novel crossover operator namely the Parental Success Crossover (PSC) and its integration within a Hierarchical PGA (HPGA). Through different experiments on randomly generated CSPs, we showed that PSC is far more efficient than the One Point Crossover and the well known heuristic based GAs. In addition, our tests also demonstrate that our HPGA compete with the well known Abscon solver. These promising results motivated us to follow this work further into exploring more general problems including variants of CSPs such as dynamic CSPs [10,11], as well as CSPs under preferences, change and uncertainty [12]. These latter problems are optimization problems where the goal is to come up with a solution maximizing some objective functions.

References

1. Cantu-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers (2000)
2. Dechter, R.: *Constraint Processing*. Morgan Kaufmann (2003)
3. Eiben, A.E., van der Hauw, J.K.: Adaptive Penalties for Evolutionary Graph Coloring. In: Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) *AE 1997*. LNCS, vol. 1363, pp. 95–106. Springer, Heidelberg (1998)
4. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston (1989)
5. van der Hauw, J.: *Evaluating and Improving Steady State Evolutionary Algorithms on Constraint Satisfaction Problems* (1996)
6. Jashmi, B.J., Mouhoub, M.: Solving temporal constraint satisfaction problems with heuristic based evolutionary algorithms. In: *Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 02, pp. 525–529. IEEE Computer Society, Washington, DC (2008)
7. Lecoutre, C., Tabary, S.: Abscon 109: a generic csp solver. In: *2nd International Constraint Solver Competition, held with CP 2006 (CSC 2006)*, pp. 55–63 (2008)
8. Lim, D., Ong, Y.S., Jin, Y., Sendhoff, B., Lee, B.S.: Efficient hierarchical parallel genetic algorithms using grid computing. *Future Gener. Comput. Syst.* 23(4), 658–670 (2007)
9. Liu, Z., Liu, A., Wang, C., Niu, Z.: Evolving neural network using real coded genetic algorithm (ga) for multispectral image classification. *Future Gener. Comput. Syst.* 20(7), 1119–1129 (2004)
10. Mouhoub, M.: Dynamic Path Consistency for Interval-based Temporal Reasoning. In: *21st International Conference on Artificial Intelligence and Applications (AIA 2003)*, pp. 393–398. ACTA Press (2003)
11. Mouhoub, M.: Systematic versus non systematic techniques for solving temporal constraints in a dynamic environment. *AI Communications* 17(4), 201–211 (2004)
12. Mouhoub, M., Sukpan, A.: Managing dynamic csps with preferences. *Applied Intelligence* 37(3), 446–462 (2012)
13. Mouhoub, M., Jashmi, B.J.: Heuristic techniques for variable and value ordering in csps. In: Krasnogor, N., Lanzi, P.L. (eds.) *GECCO*, pp. 457–464. ACM (2011)
14. Sabin, D., Freuder, E.C.: Contradicting conventional wisdom in constraint satisfaction. In: *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pp. 125–129. John Wiley and Sons, Amsterdam (1994)
15. Sena, G.A., Megherbi, D., Isern, G.: Implementation of a parallel genetic algorithm on a cluster of workstations: traveling salesman problem, a case study. *Future Gener. Comput. Syst.* 17(4), 477–488 (2001)
16. Smith, B., Dyer, M.: Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence* 81, 155–181 (1996)
17. Xu, K., Li, W.: Exact Phase Transitions in Random Constraint Satisfaction Problems. *Journal of Artificial Intelligence Research* 12, 93–103 (2000)

A Population-Based Strategic Oscillation Algorithm for Linear Ordering Problem with Cumulative Costs

Wei Xiao¹, Wenqing Chu², Zhipeng Lü^{2,*}, Tao Ye², Guang Liu¹,
and Shanshan Cui¹

¹ Systems Engineering Research Institute, Beijing, 100094, China

² SMART, School of Computer Science and Technology,
Huazhong University of Science and Technology, Wuhan, 430074, China
zhipeng.lv@hust.edu.cn

Abstract. This paper presents a Population-based Strategic Oscillation (denoted by PBSO) algorithm for solving the linear ordering problem with cumulative costs (denoted by LOPCC). The proposed algorithm integrates several distinguished features, such as an adaptive strategic oscillation local search procedure and an effective population updating strategy. The proposed PBSO algorithm is compared with several state-of-the-art algorithms on a set of public instances up to 100 vertices, showing its efficacy in terms of both solution quality and efficiency. Moreover, several important ingredients of the PBSO algorithm are analyzed.

Keywords: LOPCC, Strategic Oscillation Procedure, Local Search; Population Updating.

1 Introduction

The linear ordering problem with cumulative costs (LOPCC) was originally introduced in [4]. Given a complete graph with nonnegative vertex weight d_i and nonnegative arc weight c_{ij} , the objective of the LOPCC is to find a Hamiltonian path $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ which minimizes the following objective function:

$$C(\pi) = \sum_{i=1}^n \alpha_{\pi_i} \quad (1)$$

where

$$\alpha_{\pi_n} = d_{\pi_n}$$

$$\alpha_{\pi_i} = d_{\pi_i} + \sum_{j=i+1}^n c_{\pi_i \pi_j} \alpha_{\pi_j} \quad \text{for } i = n-1, n-2, \dots, 1.$$

* Corresponding author.

The cumulative backward computation of the α -values, from n to 1, makes the objective function nonlinear.

The LOPCC is arisen from the practical problem in wireless cellular communication systems, where successive interference cancellation (SIC) is an effective technique for multiuser access interference reduction. In this context, the LOPCC acts as the SIC optimization model to ensure a proper reception for the cellular network users. The interested readers are referred to [4, 14, 15] for more details.

Given the relevance of the LOPCC, a large number of procedures have been reported in the literature for solving this model. Among them are several exact methods using branch-and-bound and Mixed-Integer linear Programming model. Bertacco et al. (2005) proposed a Mixed-Integer linear Programming model (MIP) as well as an ad hoc enumerative algorithm for the exact solution of the LOPCC. Righini (2008) presented a branch-and-bound algorithm for the exact optimization of the LOPCC [8].

However, the high computational complexity of the LOPCC has led to the fact that, problems of sizes larger than $n = 50$ cannot be solved by these exact methods in a reasonable time. For larger instances, the exact methods become prohibitively expensive to apply. By contrast, various metaheuristic algorithms have been introduced to solve the LOPCC and shown to be effective to find high-quality solutions in an acceptable time.

Benvenuto et al. (2005) presented a greedy heuristic as well as a greedy randomized (GR) procedure [3]. Duarte et al. (2011) proposed a Tabu Search algorithm for the LOPCC [5], which is adapted from the Tabu Search procedure for the LOP [13]. Duarte et al. (2012) proposed an iterated greedy-strategic oscillation algorithm with the path relinking post-processing to obtain improved outcomes [6], identifying 87 new best objective function values.

In this paper, we present a population based strategic oscillation local search for the LOPCC, which integrates a strategic oscillation local search with a population updating approach. Our proposed algorithm is characterized by several features that enhance its effectiveness. First, in order to enhance intensification, we propose an improved local search, which can be considered as a strategic oscillation algorithm. Second, our algorithm keeps a set of local optimum solutions and relies on an effective replacement strategy to maintain the population diversity. These features distinguish our algorithm from the previous metaheuristic algorithms reported in [3–6, 8]. To assess the performance and the competitiveness of our algorithm in terms of both solution quality and efficiency, we provide computational results on a set of 50 random instances up to 100 vertices, showing the efficacy of the proposed PBSO algorithm.

2 Population-Based Strategic Oscillation

2.1 Main Scheme

Generally speaking, our PBSO algorithm can be considered as a population-based algorithm, whereas it generates new solution without recombination operators. In our procedure, each solution in the population has possibility to

be chosen for generating new solution and the solution having better objective function value is given higher possibility. In addition, to reach more intensive search, we implement a strategic oscillation local search algorithm to optimize each newly generated solution.

The general architecture of the PBSO algorithm is described in Algorithm 1. It is composed of three main components: population initialization, a strategic oscillation local search procedure and a population updating rule. Starting from an initial random population, PBSO uses the strategic oscillation local search to optimize each individual to reach a local optimum (lines 4-6). Then, a solution is randomly selected from the population and is perturbed (lines 8-9), whereupon a new round of strategic oscillation local search is again launched to optimize the objective function (lines 9-10). Subsequently, the population updating rule decides whether such an improved solution should be inserted into the population and which existing solution should be replaced (line 11). In the following subsections, the main components of our algorithm are described in details.

Algorithm 1. The pseudocode of PBSO for the LOPCC

```

1: Input: arc weight matrix  $C$  and vertex weight array  $D$ 
2: Output: the best solution  $s^*$  found so far
3:  $P = \{s_1, s_2, \dots, s_p\} \leftarrow \text{Population\_Initialization}()$  /* Section 2.2 */
4: for  $i = \{1, 2, \dots, p\}$  do
5:    $s_i \leftarrow \text{Strategic\_Oscillation}(s_i)$  /* Section 2.3 */
6: end for
7: repeat
8:   randomly choose  $s_k$  from  $P$ 
9:    $s_k \leftarrow \text{Pertubation}(s_k)$  /* Section 2.4 */
10:   $s_0 \leftarrow \text{Strategic\_Oscillation}(s_k)$  /* Section 2.3 */
11:   $\{s_1, s_2, \dots, s_p\} \leftarrow \text{Pool\_Updating}(s_0, s_1, s_2, \dots, s_p)$  /* Section 2.5 */
12: until the stop criterion is met
13: return the best solution  $s^*$  found so far

```

2.2 Initial Population

In the PBSO algorithm, the individuals of the initial population are generated randomly. First, the place of each vertex is assigned a number from 1 to n in order. Then, two vertices are selected randomly and their places are swapped. This swapping procedure is repeated for n^3 times. In the following procedure, each individual is further optimized by a strategic oscillation local search procedure. Let us mention that in general the initial solutions have limited influence on the solution quality obtained by the PBSO algorithm.

2.3 Strategic Oscillation Procedure

As demonstrated in [10, 16, 17], Strategic Oscillation is one of the most successful approaches for the 0-1 integer linear programming (ILP) problems. In the

literature, there is a kind of simple local search introduced in the Tabu Search for the LOPCC [5]. To enhance the intensification of our algorithm, we employ a Strategic Oscillation algorithm as our local search procedure.

First, we introduce the simple local search introduced in [5]. A neighborhood of a given permutation is obtained by an insert operation: the element in position i is inserted in another position j . The simple local search explores the neighborhood of a permutation and scans for the best improving move. If the best improving move cannot improve the objective function value any more, the simple local search is stopped. Note that in our local search procedure, we employ an effective technique to accelerate the computation of the neighborhood moves originally proposed for LOP in [9]. Interested readers are referred to [9] for more details.

Strategic Oscillation was initially proposed with the purpose of crossing back and forth between the feasible and infeasible search space. Occupying a key position among tabu search strategies, Strategic Oscillation has notably also been used in settings for transitioning between multiple neighborhoods, decision rules and search regions [10]. One of the explanations suggested for the success of the approach lies in its ability to integrate diversification with intensification, without resorting to “randomized” forms of diversification.

In order to adapt the strategic oscillation method to the local search procedure for the LOPCC, we divide the neighborhood into two parts. One is defined by backward neighborhood and the other is called forward neighborhood. In backward neighborhood, an element is removed from its current position i and inserted into another position j , where i is smaller than j . On the other hand, in forward neighborhood i is greater than j . Then, we design some rules for transitioning between the two neighborhoods.

The strategic oscillation local search consists of several periods. At first, the backward neighborhood acts as the feasible neighborhood and the forward neighborhood works as the infeasible neighborhood. In the next period, the feasible neighborhood and infeasible neighborhood are just the opposite. There are two very important parameters β and γ in our strategic oscillation procedure. Generally speaking, β controls when the local search switches to the next period. γ decides how many periods there are in a single strategic oscillation local search. In particular, we consider the moves in the feasible space constructive ones (A1) and the moves in the infeasible space destructive ones (A2).

A1. In the feasible neighborhood, if there exists a neighboring solution better than the previous best solution, select this best move.

A2. If there does not exist any neighboring solution better than the previous best solution in the feasible neighborhood, choose a random move in the infeasible neighborhood.

Applied to the LOPCC problem, we start from a random solution, the preceding method first executes a series of improving moves employing step A1 that takes it to a locally optimal solution. Upon reaching this juncture the method applies to step A2 to make a little destruction to the current solution. Then the procedure continues to select A1 or A2 according to whether there is an

improving move in the feasible neighborhood. If the step A2 have been taken for β times consecutively, the local search enters into next period in which the feasible neighborhood and the infeasible neighborhood are exchanged. When this transitioning happens for γ times consecutively and the objective function value is not improved, the strategic oscillation local search procedure is stopped.

The Pseudo-code of this strategic oscillation local search is presented in Algorithm 2. Strategic oscillation local search starts from a random permutation S . Then it generates an improved permutation S^* by constructing and destructing this solution (lines 6-19). In this procedure, we set β to be 5 and γ to be 3 (line 3). The function *construction* is constructive step A1 and the function *destruction* is destructive step A2.

Algorithm 2. The pseudocode of our strategic oscillation local search

```

1: Input: a random permutation  $S$ 
2: Output: the best permutation  $S^*$  found so far
3: //  $S_c$ : the best neighboring solution of  $S$  in the feasible neighborhood
4: //  $S_d$ : a random neighboring solution of  $S$  in the infeasible neighborhood
5:  $S^* \leftarrow S$ ,  $\beta = 5$ ,  $\gamma = 3$ ,  $i = 0$ ,  $j = 0$ 
6: while  $i < \gamma$  do
7:   repeat
8:      $S_c \leftarrow \text{Construction}(S)$ 
9:     if  $S_c$  is better than  $S^*$  then
10:       $S^* \leftarrow S_c$ ;  $S' \leftarrow S_c$ 
11:       $i \leftarrow 0$ ;  $j \leftarrow 0$ 
12:     else
13:       $S_d \leftarrow \text{Destruction}(S)$ 
14:       $S' \leftarrow S_d$ ;  $j \leftarrow j + 1$ 
15:     end if
16:      $S \leftarrow S'$ 
17:   until  $j$  reaches  $\beta$ 
18:   exchange the feasible and the infeasible neighborhoods,  $i \leftarrow i + 1$ 
19:    $S \leftarrow S^*$ 
20: end while

```

2.4 Perturbation Operator

In our PBSO algorithm, when a solution is selected from the population P , we perturb it and improve the perturbed solution by launching the strategic oscillation local search again. The perturbation phase works as follows. Each time two vertices are randomly selected and swapped. This procedure is repeated for a given number of times, which is called perturbation strength.

$$\text{PerturbStrength} = n/10 + \text{rand}(10),$$

where n is the size of the problem and $\text{rand}(10)$ takes a random value from 1 to 10.

2.5 Population Updating

When a new local optimum solution is reached by the strategic oscillation local search, our algorithm updates the population using this new local optimum solution. Specifically, this new local optimum solution replaces the worst solution in the population according to the objective function value.

3 Computational Results and Comparisons

In this section, we report experimental results of our PBSO algorithm on two well-known sets of random instances originally proposed by *Reinelt* 1985 [12] and reported in *Duarte* [6], respectively with 35 and 100 vertices. In addition, we compare them with those of the best performing algorithms in the literature.

3.1 Problem Instances and Experimental Protocol

Two sets of problem instances are considered in the experiments, in total constituting 50 instances. The first set of benchmarks is composed of 25 small instances with 35 vertices. The second set of benchmarks consists of 25 random instances with 100 vertices. They are frequently used by many researchers, see for example [3–6, 8].

These instances can be classified into two categories. For the 25 RANDOM instances with size 35, the optimal solutions are known. For the remaining 25 instances with size 100, the optimal solutions are unknown, and a list of best-known solutions are found by many researchers using various algorithms. In this section, we report our computational results on these two sets of instances.

Our PBSO algorithm is programmed in C++ and compiled using GNU G++ on a PC running Windows XP with 2.8GHz CPU and 2Gb RAM. To obtain our computational results, most instances are independently solved 5 times with different random seeds. Each run is stopped when the generation number reaches 100. All the computational results were obtained without special tuning of the parameters, i.e., all the parameters mentioned in our algorithm are constant for all instances.

3.2 Computational Results

Tables 1 and 2 respectively summarize the computational statistics of the proposed PBSO algorithm for the *RANDOM* instances with size 35 and 100. Columns 2-3 give the features of the tested instances: the numbers of the vertices (n) and the previous best known results (or the optimal solutions) (f_{prev}). Columns 4-6 give our results: the best objective value (f_{best}), the best solution gap to the best known objective values (g_{best}), the average CPU time in minutes for reaching the best objective values (Time). Furthermore, the last row (Average) indicates the summary of the average performance of our algorithm.

From Table 1, one observes that PBSO algorithm can stably reach the optimal solutions for all the instances except one (t1d35.10). The average CPU time for

Table 1. Computational results on the 25 RANDOM instances with $n = 35$

Instances	n	f_{prev}	f_{best}	g_{best}	Time(s)
t1d35.1	35	0.923	0.923	0%	6.94
t1d35.2	35	0.167	0.167	0%	7.04
t1d35.3	35	0.154	0.154	0%	6.96
t1d35.4	35	0.196	0.196	0%	7.77
t1d35.5	35	1.394	1.394	0%	7.05
t1d35.6	35	0.200	0.200	0%	7.91
t1d35.7	35	0.120	0.120	0%	6.65
t1d35.8	35	0.226	0.226	0%	7.73
t1d35.9	35	0.436	0.436	0%	7.13
t1d35.10	35	0.205	0.223	8.780%	7.29
t1d35.11	35	0.369	0.369	0%	7.19
t1d35.12	35	0.234	0.234	0%	7.26
t1d35.13	35	0.196	0.196	0%	7.62
t1d35.14	35	0.138	0.138	0%	7.43
t1d35.15	35	1.376	1.376	0%	6.39
t1d35.16	35	0.286	0.286	0%	7.49
t1d35.17	35	0.199	0.199	0%	7.27
t1d35.18	35	0.381	0.381	0%	7.87
t1d35.19	35	0.236	0.236	0%	7.91
t1d35.20	35	0.068	0.068	0%	7.35
t1d35.21	35	0.202	0.202	0%	6.28
t1d35.22	35	0.177	0.177	0%	6.83
t1d35.23	35	0.345	0.345	0%	8.07
t1d35.24	35	0.132	0.132	0%	7.51
t1d35.25	35	0.143	0.143	0%	7.24
Average		0.340	0.341	0.351%	7.29

Table 2. Computational results on the 25 RANDOM instances with $n = 100$

Instances	n	f_{prev}	f_{best}	g_{best}	Time(m)
t1d100.1	100	253.988	246.290	-3.031%	25.5
t1d100.2	100	288.372	282.933	-1.886%	25.6
t1d100.3	100	1307.432	1243.856	-4.863%	24.2
t1d100.4	100	7539.979	6995.723	-7.218%	23.42
t1d100.5	100	169.336	163.509	-3.441%	24.3
t1d100.6	100	395.035	391.431	-0.912%	22.4
t1d100.7	100	5936.281	5812.945	-2.078%	25.8
t1d100.8	100	2760.619	2750.802	-0.356%	25.1
t1d100.9	100	62.942	62.095	-1.346%	25.64
t1d100.10	100	162.942	154.812	-4.989%	24.0
t1d100.11	100	233.586	231.139	-1.048%	25.9
t1d100.12	100	236.696	234.014	-1.133%	25.1
t1d100.13	100	593.319	589.834	-0.587%	22.9
t1d100.14	100	249.162	243.739	-2.176%	26.74
t1d100.15	100	406.478	406.478	0%	23.1
t1d100.16	100	707.413	707.413	0%	25.15
t1d100.17	100	725.790	718.661	-0.982%	23.26
t1d100.18	100	622.942	621.686	-0.201%	30.1
t1d100.19	100	228.486	228.486	0%	22.6
t1d100.20	100	255.151	239.937	-5.962%	22.7
t1d100.21	100	228.590	222.969	-2.458%	23.2
t1d100.22	100	159.336	140.253	-11.977%	22.7
t1d100.23	100	1658.168	1588.758	-4.186%	23.4
t1d100.24	100	469.658	461.479	-1.741%	22.7
t1d100.25	100	644.782	627.066	-2.748%	21.9
Average	100	1051.859	1014.974	-3.507%	24.3

reaching the best known results is about 7.29 minutes. The average gap to the previous best objective values is -0.001 for these small instances.

Table 2 discloses that PBSO algorithm can stably obtain better results for 22 instances and match the previous best objective values for the remaining three instances. The average CPU time for reaching the best known results is about 24.33 minutes. The average gap to the previous best objective values is -37.198 for these instances.

In summary, our PBSO algorithm reached the optimal solutions for 24 out of 25 instances with 35 vertices and improved 22 previous best results out of 25 instances with 100 vertices. Our algorithm obtains worse results only for one small instance and matches the previous best results for 3 instances whose sizes are 100. These results demonstrate the competitiveness of our PBSO algorithm in terms of both solution quality and efficiency.

3.3 Comparison with other Reference Algorithms

In order to further evaluate our PBSO algorithm, we compare our results with some best performing algorithms in the literature. For this purpose, we restrict our attention to comparisons with four methods that have produced the best results for many challenging instances. These methods are respectively named *GR* in [3], *TB&B* in [8], *EVPR* in [6], and *TS* in [5].

Table 3 shows the best results of our PBSO algorithm compared with these reference algorithms for the 25 large instances with $n = 100$. The results of the four reference algorithms are directly extracted from [5]. Note that the number of BKS in the table denotes the number of previous best known solutions that a reference algorithm. From Table 3, it is clear that TB&B and GR are not able to match any best known values, while TS obtains 12 best known values and EVPR obtains 13 best known values. However, one should notice that our PBSO algorithm can improve 22 out of these 25 previous best known solutions.

In particular, PBSO has -3.51% deviation with respect to the previous best known solutions over these large random instances, while EVPR obtains a deviation 0.74% and the rest 3 algorithms obtain even worse deviations. However, PBSO needs more computational time than other reference algorithms. It is difficult to exactly compare the computational time since difference algorithms use different machines. On the other hand, our algorithm can obtain better solutions in most cases, which deserves more computational time to some extent.

Table 3. Comparisons with other reference algorithms on the large instances with $n = 100$

	GR	TB&B	TS	EVPR	PBSO
Obj. function	288137.55	9614.23	1161.46	1058.78	1014.86
Avg. deviation	13001.93%	430.00%	5.79%	0.74%	-3.51%
Number of BKS	0	0	12	13	25

4 Analysis and Discussion

We now turn our attention to discussing and analyzing several important features of the proposed PBSO algorithm, including the advantage of the strategic oscillation over the simple local search and the parameters in the strategic oscillation.

4.1 Strategic Oscillation vs. Local Search

In order to ensure that the strategic oscillation local search makes a meaningful contribution, we conduct experiments to compare the performance of the strategic oscillation local search with simple local search on 10 random instances t1d100.1, t1d100.2, ... and t1d100.10 whose sizes are 100.

We replace the strategic oscillation local search with the simple local search and compare two versions of our PBSO algorithm under exactly the same conditions as before (the number of generations) and the results are reported in Table 4. Once again, the following criteria are provided for each instance: the best solution gap to the previous best known objective values (g_{best}) and the average computational time ($time$).

Table 4. different local searches on 25 random instances ($n = 100$)

Instance	f_{prev}	simple local search			strategic oscillation local search		
		f_{best}	g_{best}	$t_{avr}(m)$	f_{best}	g_{best}	$t_{avr}(m)$
t1d100.1	253.988	263.224	3.6%	15.0	246.290	-3.03%	23.9
t1d100.2	288.372	318.757	10.5%	14.9	282.933	-1.88%	23.5
t1d100.3	1307.432	1383.311	5.8%	16.0	1243.856	-4.86%	22.8
t1d100.4	7539.979	7894.483	4.7%	17.5	6995.723	-7.21%	21.6
t1d100.5	169.336	176.63	4.3%	16.8	163.509	-3.44%	22.4
t1d100.6	395.035	455.503	15.3%	19.8	391.431	-0.91%	20.7
t1d100.7	5936.281	6441.776	8.5%	14.4	5812.945	-2.07%	24.48
t1d100.8	2760.619	3098.517	12.2%	16.2	2750.802	-0.36%	23.6
t1d100.9	62.942	62.781	0.3%	14.9	62.905	-1.34%	24.2
t1d100.10	162.942	180.91	11%	16.5	154.812	-4.9%	22.5
Average	1887.693	2027.589	7.62%	16.2	1810.521	-3.00%	22.9

For the 10 random instances, strategic oscillation local search performs better than the simple local search in terms of the best solution gap (-3.00% for the strategic oscillation local search against 7.62% for the simple local search) while strategic oscillation local search performs worse in terms of the average computational time (22.963 for the strategic oscillation local search against 16.249 for the general local search). This indicates that strategic oscillation local search need more computational time with the same number of generations, implying that strategic oscillation can avoid the disadvantage of falling into local optimum trap to some extent. This experiment highlights the advantage of the strategic oscillation local search and provides an empirical justification of its use for the LOPCC.

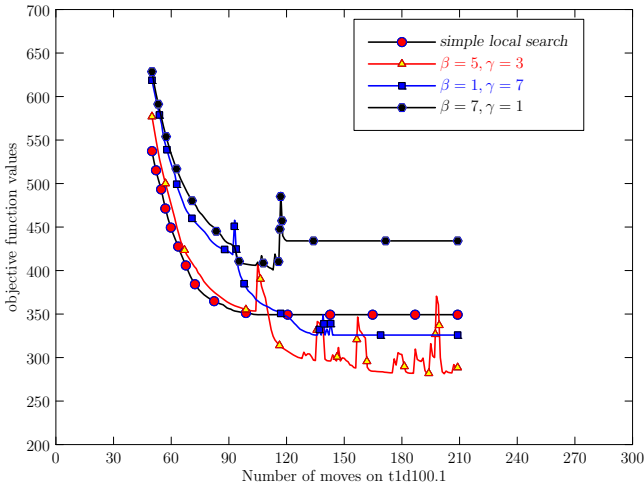


Fig. 1. Comparison among different parameters

4.2 β and γ in the Strategic Oscillation

Although the strategic oscillation local search combined with population-based algorithm achieves better effect than the simple local search, we want to find more details to where its advantages are from. So we are interested in observing the change process of the objective function values in one descent. Otherwise, we are curious about the two parameters β and γ . We want to know what their function are in the strategic oscillation local search and the impacts of different assignments on the local search. For this purpose, experiments are performed on various instances. We present below in detail the results on the instance t1d100.1, but these results are valid for other cases.

To implement this experiment, we consider four different local searches. One is the simple local search, the rest three are strategic oscillation local search with different values of the parameters of β and γ : $(\beta, \gamma) = (7, 1)$, $(\beta, \gamma) = (5, 3)$, $(\beta, \gamma) = (1, 7)$. For each procedure, we record the objective function values after every move until the procedures are over. Fig.1 shows the change process of the objective function values during these different procedures.

From Fig.1, we first notice that these four local search take different numbers of total moves to come to an end. The simple local search moves for about 100 steps and its descent is smooth. The strategic oscillation local search with $(\beta, \gamma) = (7, 1)$ has about 150 steps and there is a large undulation in its descent. The strategic oscillation local search with $(\beta, \gamma) = (1, 7)$ need about 120 steps and descend with some serried small undulations. The strategic oscillation local search with $(\beta, \gamma) = (5, 3)$ moves about 210 steps and many adjacent medium undulations appear in its drop line. Obviously, the last local search obtains best objective function values.

The simple local search is easy to fall into some trap and it ends at the earliest. The rest three strategic oscillation local search are all able to overcome some traps. The β decides when the local search enters into the next period and it can adjust the size of the search space. The γ decides how many periods there are and it controls the strength of the local search. When β is small and γ is big (like $(\beta, \gamma) = (1, 7)$), the local search will search in a little space for many times as well as serried small undulations. When β is big and γ is small (like $(\beta, \gamma) = (7, 1)$), the local search will search in a large space for few times as well as a large undulation. The increase of β bring about the larger search space, so it deserve more search and γ should be bigger. That is why the strategic oscillation local search with $(\beta, \gamma) = (5, 3)$ can have best performance. The two parameters β and γ can also be considered as two aspects of the procedure, diversification and intensification. If the value of β is bigger, the local search has high possibility to find new variant solutions. When γ is very large, the procedure scans very intensively in some search space. β and γ two important parameters and they are related to each other very closely. This experiment also shows a clear advantage to setting appropriate values for β and γ in order to achieve a desired effect.

5 Conclusions

In this paper, we have presented PBSO, a population based strategic oscillation local search for solving the LOPCC problem. The proposed algorithm integrates a number of original features. First, we have proposed a strategic oscillation local search procedure for solving the problem. Second, we incorporate a population-based strategy with the strategic oscillation local search. These strategies provide the algorithm with a good tradeoff between intensification and diversification.

Experiments tested on a sets of 50 well-known random benchmark instances have shown that this metaheuristic algorithm obtains competitive results in comparison with the previous best known objective values within a reasonable computation time.

The success of the PBSO algorithm on the LOPCC problem reminds us that the application of strategic oscillation local search and the population-based algorithm for solving LOPCC is useful. Moreover, it is meaningful to apply the strategic oscillation to other permutation problems.

Acknowledgments. This work was partially supported by the National Natural Science Foundation of China (Grant No. 61100144) and the Doctoral Fund of Ministry of Education of China (Grant No. 20110142120081).

References

1. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. *European Journal of Operational Research* 203(1), 241–250 (2010)
2. Lü, Z., Glover, F., Hao, J.K.: A hybrid metaheuristic approach to solving the UBQP Problem. *European Journal of Operational Research* 207(3), 1254–1262 (2010)

3. Benvenuto, N., Carnevale, N., Tomasin, S.: Optimum power control and ordering in SIC receivers for uplink CDMA systems. In: IEEE International Conference on Communications, ICC 4, pp. 2333–2337 (2005)
4. Bertacco, L., Brunetta, L., Fischetti, M.: The linear ordering problem with cumulative costs. *European Journal of Operational Research* 189(3), 1345–1357 (2005)
5. Duarte, A., Laguna, M., Marti, R.: Tabu search for the linear ordering problem with cumulative costs. *Computational Optimization and Applications* 48, 697–715 (2011)
6. Duarte, A., Marti, R., Alvarez, A., Angel-Bello, F.: Metaheuristics for the linear ordering problem with cumulative costs. *European Journal of Operational Research* 216(2), 270–277 (2012)
7. Villanueva, D.T., Fraire Huacuja, H.J., Duarte, A., Pazos R., R., Carpio Valadez, J.M., Puga Soberanes, H.J.: Improving Iterated Local Search Solution for the Linear Ordering Problem with Cumulative Costs (LOPCC). In: Setchi, R., Jordanov, I., Howlett, R.J., Jain, L.C. (eds.) KES 2010, Part II. LNCS (LNAI), vol. 6277, pp. 183–192. Springer, Heidelberg (2010)
8. Righini, G.: A branch-and bound algorithm for the linear ordering problem with cumulative costs. *European Journal of Operational Research* 186(3), 965–971 (2008)
9. Schiavinotto, T., Stützle, T.: The linear ordering problem: Instances, search space analysis and algorithms. *European Journal of Operational Research* 177, 2033–2049 (2007)
10. Glover, F., Hao, J.K.: The case for strategic oscillation. *Annals OR* 183(1), 163–173 (2011)
11. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the linear ordering problem with cumulative costs. *European Journal of Operational Research* 177, 2033–2049 (2007)
12. Reinelt, G., Hofmann, H.H., Wille, R.: The linear ordering problem: Algorithms and applications. *Research and Exposition in Mathematics* 8 (1985)
13. Laguna, M., Marti, R., Campos, V.: Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & OR* 26(12), 1217–1230 (1999)
14. Proakis, J.G.: *Digital Communications*, 4th edn. McGraw-Hill (2004)
15. Holma, H., Toskala, A.: *WCDMA for UMTS: Radio access for Third generation mobile communications*. Wiley, New York (2000)
16. Glover, F., Laguna, M.: General purpose heuristics for integer programming-part II. *J. Heuristics* 3(2), 161–179 (1997)
17. Glover, F.: Multi-start and strategic oscillation methods - Principles to exploit adaptive memory. In: Laguna, M., Gonzales Velarde, J.L. (eds.) *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pp. 1–24 (2000)

A Study of Adaptive Perturbation Strategy for Iterated Local Search

Una Benlic and Jin-Kao Hao

LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers Cedex 01, France
{benlic,hao}@info.univ-angers.fr

Abstract. We investigate the contribution of a recently proposed adaptive diversification strategy (ADS) to the performance of an iterated local search (ILS) algorithm. ADS is used as a diversification mechanism by breakout local search (BLS), which is a new variant of the ILS meta-heuristic. The proposed perturbation strategy adaptively selects between two types of perturbations (directed or random moves) of different intensities, depending on the current state of search. We experimentally evaluate the performance of ADS on the quadratic assignment problem (QAP) and the maximum clique problem (MAX-CLQ). Computational results accentuate the benefit of combining adaptively multiple perturbation types of different intensities. Moreover, we provide some guidance on when to introduce a weaker and when to introduce a stronger diversification into the search.

Keywords: adaptive perturbation strategy, iterated local search, breakout local search, quadratic assignment, maximum clique.

1 Introduction

To be successful, a heuristic approach needs to find a suitable balance between an intensified and a diversified search. Intensification is the ability of the method to examine in depth specific search areas while diversification is the capacity of the method to diversify the search in order to find promising new search areas. If the diversification is too weak, the search has a great chance to end up cycling between two or several previously encountered local optima. On the other hand, a too strong diversification is no better than a random restart and may reduce the chances of finding better solutions in the following iterations. Determining the right degree of diversification is not a straightforward task, since it greatly depends on structural characteristics of the given instance such as the distribution of local optima, the correlation between solutions, the number of global optima, etc. Additionally, the optimal diversification degree required at one stage of the search is not necessarily optimal at another stage. These facts constitute the motivation for our adaptive diversification mechanism.

In this paper, we investigate the contribution of a new adaptive diversification strategy (ADS) employed by a recent breakout local search (BLS) meta-heuristic [3,4,5,6]. BLS is a variation of iterated local search (ILS) [9] since it

combines a descent-based local search with a perturbation mechanism. However, BLS has a particular focus on the importance of the perturbation phase. Based on some information on the search history, it dynamically determines the number of perturbation moves, and adaptively chooses between two or several types of perturbation moves of different intensities. In this work, we fix the number of perturbation moves, and evaluate the efficiency of this adaptive multi-type diversification on the quadratic assignment problem (QAP) and the maximum clique problem (MAX-CLQ). More precisely, we integrate ADS into a basic ILS algorithm and compare the performance of this adaptive diversification based ILS against two other ILS versions based respectively on random and directed perturbation moves. The obtained computational results accentuate the benefit of combining adaptively multiple perturbation types of different intensities. Furthermore, we analyze the distribution of local optima to provide some guidance on when to introduce a weaker or a stronger diversification into the search.

2 Iterated Local Search with a Adaptive Diversification Strategy

2.1 General Framework

The basic idea of iterated local search (ILS) is to alternate iteratively between a local search phase to attain local optima, and a perturbation phase (applied to the current or best found local optimum) to direct the search towards new unexplored regions of the search space.

Alg. 1 shows the general framework of our adaptive diversification based ILS (denoted by AD-ILS), which we later apply to two *NP*-hard problems considered in sections 3 and 4. Starting from an initial feasible solution, AD-ILS first initializes the best-found solution S_{best} , the tabu list H (see Section 2.2), the counter ω for consecutive non-improving local optima, and the global iteration counter $iter$ (lines 1-5 of Alg. 1). While a stopping condition is not satisfied, AD-ILS applies a simple descent (ascent in case of maximization) local search to reach a local optimum S (lines 8-12 of Alg. 1). Each iteration of this descent-based procedure searches the given neighborhood for the best solution to replace the current solution, and stops if no improving neighbor exists (i.e., once local optimality is reached). After each solution transition, AD-ILS updates the tabu list H (see Section 2.2) and increments the global iteration counter $iter$.

If the quality of the local optimum S , reached in the last descent phase, is better than the quality of the best-found solution S_{best} , AD-ILS updates S_{best} and re-initializes the number of consecutive non-improving local optima ω (lines 13-15 of Alg. 1). Otherwise, ω is incremented by one (lines 16-17 of Alg. 1). If ω exceeds a certain threshold T , it is reset to zero (lines 19-21 of Alg. 1).

In order to escape from the current local optimum S , AD-ILS applies its perturbation mechanism ADS to S , and returns a perturbed solution which becomes a new starting point for the next phase of the descent/ascent procedure (line 22 of Alg. 1).

Algorithm 1. Adaptive_Diversification-based_Iterated_Local_Search

Require: The jump magnitude L , the threshold T and the tabu tenure γ .
Ensure: Solution S_{best} .

```

1:  $S \leftarrow Initial\_Solution$ 
2:  $S_{best} \leftarrow S$ ; /* Initialize the best-found solution  $S_{best}$  */
3:  $H \leftarrow 0$ ; /* Initialize the tabu list  $H$  */
4:  $\omega = 0$ ; /* Initialize the number of consecutive non-improving local opt.  $\omega$  */
5:  $iter = 0$ ; /* Initialize the global iteration counter  $iter$  */
6: while stopping condition not reached do
7:   Let  $m$  be the best move eligible for  $S$ 
8:   while  $f(S \oplus m)$  is better than  $f(S)$  do
9:      $S \leftarrow S \oplus m$  /* Perform the best-improving move */
10:     $H_m \leftarrow Iter + \gamma$  /* Update tabu list,  $\gamma$  is the tabu tenure */
11:     $Iter \leftarrow Iter + 1$ 
12:   end while
13:   if  $f(S)$  is better than  $f(S_{best})$  then
14:      $S_{best} \leftarrow S$ 
15:      $\omega = 0$ 
16:   else
17:      $\omega = \omega + 1$ 
18:   end if
19:   if  $\omega > T$  then
20:      $\omega = 0$ 
21:   end if
22:    $S \leftarrow Adaptive\_Diversification\_Strategy(S, H, \omega, L, iter, T, \gamma)$  /* Sect. 2.2*/
23: end while

```

Since the local search phase is a simple decent/ascent procedure, it alone cannot escape from a local optimum. The performance of AD-ILS thus strongly depends on its perturbation mechanism ADS which is detailed in the next section.

2.2 Adaptive Diversification Strategy (ADS)

The AD-ILS algorithm, that we apply to QAP (Section 3) and MAX-CLQ (Section 4), employs a directed and a random perturbation to guide the search towards new regions of the search space. This adaptive perturbation mechanism is illustrated in Alg. 2.

The directed perturbation (DIRP) is based on the idea of tabu list from tabu search [7]. It uses a selection rule that favors the move that minimizes the degradation of the objective, under the constraint that this move is not prohibited by the tabu list. The information for move prohibition is maintained in a tabu list H , such that each element in H is the iteration number when the corresponding move was last performed, plus the tabu tenure γ (represented as a natural number). The tabu status of a move is neglected only if the move leads to a new solution better than the best solution found so far. The directed perturbation relies thus both on 1) history information which keeps track, for each move, of the last time (iteration) when it was performed and 2) on the quality of the moves to be applied for perturbation in order not to deteriorate too much the perturbed solution. History-based diversifications have previously been used in [2,8].

Algorithm 2. Adaptive_Diversification_Strategy($S, H, \omega, L, iter, T, \gamma$)

Require: Local optimum S , tabu list H , number of consecutive non-improving local optima visited ω , jump magnitude L , global iteration counter $Iter$, threshold T , tabu tenure γ .

- 1: $P \leftarrow \text{Determine_Probability_for_Directed_Perturbation}(\omega, T)$ /* see Eq. 1 */
- 2: **if** ($P > \text{random}[0.0, 1.0]$) **then**
- 3: $PERT = \text{DIRP}$ /* L moves of DIRP will be applied to S */
- 4: **else**
- 5: $PERT = \text{RNDP}$ /* L moves of RNDP will be applied to S */
- 6: **end if**
- 7: **for** $i := 1$ to L **do**
- 8: $S \leftarrow \text{Perturbation_Move}(S, PERT)$ /* Apply a move m of the predetermined perturb.*/
- 9: $H_m \leftarrow Iter + \gamma$
- 10: $Iter \leftarrow Iter + 1$
- 11: **if** $f(S)$ is better than $f(S_{best})$ **then**
- 12: $S_{best} \leftarrow S$
- 13: $\omega = 0$
- 14: **end if**
- 15: **end for**

The random perturbation (RNDP) is the most popular type of perturbation for ILS algorithms. It consists in performing randomly selected moves.

It is obvious that DIRP and RNDP introduce different balances between intensification and diversification. More precisely, DIRP is more oriented towards search intensification than RNDP since it considers the quality of moves in order not to degrade too much the resulting solution. The search thus has greater chances to end cycling between two or more local optima if DIRP is used alone. On the other hand, RNDP may prevent the search from cycling, but it may as well decrease the chances of finding a global optimum by passing too quickly to new regions while promising regions were not sufficiently exploited.

To insure the best balance as possible between an intensified and a diversified search, the adaptive diversification strategy ADS applies probabilistically DIRP and RNDP. The probability of applying a particular type of perturbation is determined dynamically depending on the search state, i.e., the current number ω of consecutive non-improving attractors visited (lines 13-21 of Alg. 1). The idea is to apply more often directed perturbations (with a higher probability) as the search progresses towards improved new local optima (the non-improving consecutive counter ω is small). With the increase of ω , the probability of using the directed perturbations progressively decreases while the probability of applying the random moves increases for the purpose of a stronger diversification.

Additionally, it has been observed from an experimental analysis that it is useful to guarantee a minimum of applications of DIRP. Therefore, we constrain the probability P of applying DIRP to take values no smaller than a threshold P_0 :

$$P = \begin{cases} e^{-\omega/T} & \text{if } e^{-\omega/T} > P_0 \\ P_0 & \text{otherwise} \end{cases} \quad (1)$$

Once the type of perturbation is determined (lines 1-6 of Alg 2), AD-ILS applies L moves of the selected perturbation to the current local optimum S (lines 7-15 of Alg 2).

3 Case Study I: Quadratic Assignment Problem (QAP)

3.1 Problem Description

The *quadratic assignment problem* (QAP) is a well-known *NP*-hard problem which consists in assigning at minimal cost a set of n locations to a given set of n facilities, given a flow f_{ij} from facility i to facility j for all $i, j \in \{1, \dots, n\}$ and a distance d_{ab} between locations a and b for all $a, b \in \{1, \dots, n\}$. Let Π denote the set of the permutation functions $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$, QAP can mathematically be formulated as follows:

$$\min_{\pi \in \Pi} C(\pi) = \sum_{i=1}^n \sum_{j=1}^n f_{ij} d_{\pi_i \pi_j} \quad (2)$$

where f and d are the flow and distance matrices respectively, and $\pi \in \Pi$ is a solution where π_i represents the location assigned to facility i . The problem objective is then to find a permutation π^* in Π that minimizes the sum of the products of the flow and distance matrices, i.e., $C(\pi^*) \leq C(\pi), \forall \pi \in \Pi$.

3.2 Neighborhood Relation and Its Exploitation

A candidate solution for QAP can be encoded as a permutation π of $\{1, \dots, n\}$, such that π_i denotes the location assigned to facility $i \in \{1, \dots, n\}$. Like many existing local search methods for QAP, our ILS employs the swap move to π which consists in exchanging the locations of two facilities.

The neighborhood $N(\pi)$ of a solution π is then defined as the set of all the permutations that can be obtained by exchanging any two values π_u and π_v , i.e., $N(\pi) = \{\pi' : \pi'_u = \pi_v, \pi'_v = \pi_u, u \neq v \text{ and } \pi'_i = \pi_i, \forall i \neq u, v\}$. The size of $N(\pi)$ is thus equal to $n(n-1)/2$.

The local search phase of ILS explores the whole neighborhood $N(\pi)$ to find the best swap move which is then applied to π to obtain a new solution. This process is repeated until a local optimum is reached. To evaluate the whole swap neighborhood $N(\pi)$ in $O(n^2)$ time, we use an effective strategy which incrementally updates the objective variation of each move [10].

3.3 Perturbation Types Combined with ADS

The *directed perturbation* (see Section 2.2) applies a swap move that minimizes the value of the objective function C , under the constraint that the move has not been applied during the last γ iterations (γ is the tabu tenure that takes a random value from a given range). The eligible moves for the directed perturbation are identified by the set A such that:

$$A = \{\text{swap}(u, v) : \min\{\delta(\pi, u, v)\}, H_{uv} < \text{Iter} \text{ or } (\delta(\pi, u, v) + c) < c_{\text{best}}, u \neq v\}$$

where H is the tabu list that keeps track of the iteration number when a move was last performed plus γ , Iter the current iteration number, c the cost of the

current solution, and c_{best} the cost of the best solution discovered so far. A larger value of γ implies stronger diversification.

The *random perturbation* simply performs swap moves that are selected uniformly at random.

Our AD-ILS for QAP combines and applies these two types of perturbations as explained in Section 2.2.

3.4 Experimental Results and Comparisons

To evaluate the efficiency of the proposed AD-ILS algorithm for QAP, we carry out experiments on a set of 16 difficult QAPLIB instances¹ of three different types (unstructured instances, real-life like instances, grid-based instances).

We contrast the results of AD-ILS with those obtained with two other ILS versions which respectively employ perturbation strategies based on directed (DIR-ILS) and random moves (RND-ILS). For all the three ILS versions, the number of perturbation moves is $L = 0.15n$. For both AD-ILS and DIR-ILS, the tabu tenure γ takes a random value in the range $[0.9n, 1.1n]$. For AD-ILS, the setting of the parameters used for adaptive perturbation is $P_0 = 0.9$ and $T = 2500$. This setting of parameters is determined by a preliminary experiment and can be justified to some extent by the analysis provided in Section 5. We make 20 independent executions per instance, with the time limit per run set to 2 hours.

Table 1. Computational comparison of AD-ILS with DIR-ILS and RND-ILS on 16 hard QAP instances

Instance		AD-ILS			DIR-ILS			RND-ILS		
Name	BKR	$\% \rho_{best}$	$\% \rho_{avg}$	t(m)	$\% \rho_{best}$	$\% \rho_{avg}$	t(m)	$\% \rho_{best}$	$\% \rho_{avg}$	t(m)
tai40a	3139370	0.000(12)	0.030	30.2	0.000(14)	0.022	41.5	0.000(3)	0.116	49.4
tai50a	4938796	0.000(4)	0.121	62.3	0.000(3)	0.136	60.8	0.301(0)	0.576	58.0
tai60a	7205962	0.000(1)	0.359	65.9	0.191(0)	0.400	57.9	0.313(0)	0.837	47.9
tai80a	13499184	0.651(0)	0.764	67.8	0.600(0)	0.755	66.7	0.812(0)	1.179	43.9
tai100a	21052466	0.626(0)	0.804	59.7	0.648(0)	0.788	50.6	0.948(0)	1.218	68.2
tai80b	818415043	0.000(9)	0.423	34.2	0.000(1)	0.755	28.4	0.000(9)	0.001	65.4
tai100b	1185996137	0.000(12)	0.253	17.9	0.000(6)	0.382	37.9	0.000(10)	0.001	39.2
tai150b	498896643	0.000(1)	0.322	68.6	0.161(0)	0.429	80.2	0.023(0)	0.138	84.5
sko81	90998	0.000(20)	0.000	11.8	0.000(20)	0.000	2.7	0.011(0)	0.032	65.0
sko90	115534	0.000(7)	0.045	14.7	0.000(6)	0.063	24.6	0.011(0)	0.040	74.2
sko100a	152002	0.000(12)	0.006	11.7	0.000(8)	0.022	15.9	0.045(0)	0.069	64.1
sko100b	153890	0.000(20)	0.000	15.4	0.000(16)	0.019	16.9	0.016(0)	0.045	55.2
sko100c	147862	0.000(20)	0.000	9.8	0.000(19)	0.021	11.6	0.009(0)	0.046	69.4
sko100d	149576	0.000(10)	0.002	61.2	0.000(11)	0.066	29.6	0.044(0)	0.076	59.6
sko100e	149150	0.000(18)	0.000	40.3	0.000(16)	0.034	16.7	0.011(0)	0.030	69.8
sko100f	149036	0.000(20)	0.000	23.3	0.000(15)	0.013	23.1	0.023(0)	0.063	60.2
Average		0.080	0.196	38.9	0.100	0.244	37.5	0.160	0.279	60.9

Table 1 shows for each algorithm the best (column $\% \rho_{best}$) and average (column $\% \rho_{avg}$) percentage deviation from the best-known result (column BKR) obtained over 20 runs. The percentage deviation $\% \rho$ is computed as $\% \rho = 100(z - BKR)/z[\%]$, where z is the result obtained by a given approach and

¹ <http://www.seas.upenn.edu/qaplib/>

BKR the best-known objective value. Next to the percentage deviation ρ_{best} , we indicate in parentheses the number of times the best-known solution was found over 20 executions. Moreover, we provide the average times in minutes required to reach the returned solution after a trial. The best results are indicated in bold. The averaged results are provided in the last row.

From the results in Table 1, we can make the following conclusions. In most cases, the best performance is obtained with AD-ILS which reports an average $\% \rho_{avg}$ of 0.196 (vrs. 0.244 for DIR-ILS and 0.279 for RND-ILS) over the 16 QAP instances. Indeed, AD-ILS is unable to attain the best-known result from the literature only for two instances (*tai80a* and *tai100a*), while DIR-ILS and RND-ILS are unable to reach the best-known objective value for 4 and 13 instances respectively. The worst performance on the QAP instances is thus obtained with the RND-ILS algorithm, except for three real-life like instances (i.e., *tai80b*, *tai100b* and *tai150b*) for which RND-ILS algorithm insures the best performance. The Posthoc test revealed that AD-ILS statistically outperforms RND-ILS with a p -value of 0.016. In Section 5, we provide an explanation for these performances based on a landscape analysis. In terms of computing times, AD-ILS and DIR-ILS show comparable performances with an average time of 38.7 and 37.5 minutes respectively for the 16 instances, while RND-ILS requires on average around 70 minutes. These results show the advantage of applying directed or adaptive perturbations over the classic random perturbations.

4 Case Study II: Maximum Clique Problem (MAX-CLQ)

4.1 Problem Description

Given an undirected graph $G = (V, E)$ where V is the set of vertices and E the set of edges, a clique C of G is a subset of V such that all the vertices in C are pairwise adjacent, i.e., $\forall v, u \in C, \{v, u\} \in E$. The *maximum clique problem* (MAX-CLQ) is to find a clique C of the maximal cardinality. It is one of the first problems shown to be *NP*-complete.

4.2 Neighborhood Relations and Their Exploitation

For solution transformations, ILS employs four distinct move operators (moves for short) whose basic idea is to generate a new clique from the current clique C by adding vertices $v \in V \setminus C$ to C , swapping vertices u and v such that $u \in C$ and $v \in V \setminus C$, or removing vertices $v \in C$ from C .

Three sets PA , OM and OC are involved in the definition of these moves. The vertex set PA consists of nodes excluded from the clique C that are connected to all the vertices in C , i.e., $PA = \{v : v \notin C, \forall u \in C, \{v, u\} \in E\}$.

The OM set consists of vertex pairs (v, u) such that v is excluded from C and is connected to all vertices in C except to vertex $u \in C$, i.e., $OM = \{(v, u) : v \notin C \text{ and } u \in C, |N(v) \cap C| = |C| - 1, \{v, u\} \notin E\}$, where $N(v) = \{i : i \in V, \{i, v\} \in E\}$.

The OC set consists of all the vertices excluded from the clique C , i.e., $OC = \{v : V \setminus C\}$.

The four moves M_1 to M_4 can then be defined as follows:

- M_1 : Select a vertex $v \in PA$ and insert it into C . After this move, the change in the objective function is given by the following expression: $\Delta = w_v$.
- M_2 : Select a vertex pair $(v, u) \in OM$. Insert v into C and remove u from C . The change in the objective function can be computed as: $\Delta = w_v - w_u$.
- M_3 : Select a vertex $v \in C$ and remove it from C . The change in the objective function is given as: $\Delta = -w_v$.
- M_4 : Select a vertex $v \in OC$ such that $(w_v + \sum_{\{v,u\} \in E, u \in C} w_u) \geq \alpha * f(C)$, where $f(C)$ is the current solution cost and $0 < \alpha < 1$. Add v to C . Repair the resulting clique C by removing from C all vertices x such that $\{v, x\} \notin E$.

The descent-based local search phase of ILS consists in identifying the best move m from the union $M_1 \cup M_2$ and applying it to C to obtain a new solution. This procedure is repeated until a local optimum is reached. The directed perturbation (see Section 4.3) applies a move m from $M_1 \cup M_2 \cup M_3$. For random perturbation (see Section 4.3), m is selected from M_4 .

4.3 Perturbation Types Combined with ADS

As previously explained in Section 2.2, the *directed perturbation* is based on the tabu search principles and favors non-tabu moves that minimize the cost degradation. Move prohibition is determined in the following way. Each time a vertex v is added into the clique C , it can be removed from C without restrictions. However, each time v is dropped from C , it is forbidden to place it back to C for γ iterations. The value of γ is determined by the following relation:

$$\gamma = \phi + \text{random}(|OM|),$$

where ϕ is a coefficient and *random* is a function which returns at random a value ranging from 1 to $|OM|$ (the number of elements in the OM set, see Section 4.2).

The eligible moves for the directed perturbation are identified by the set A such that:

$$A = \{m : m \in \{M_1 \cup M_2 \cup M_3\}, \max\{\Delta_m\}, \text{prohibited}(m) = \text{false or } (\Delta_m + f(C)) > f_{\text{best}}\}$$

where Δ_m is the change in the objective function after performing move m (see Section 4.2). Note that the directed perturbation considers all the eligible moves from the union of three types of moves M_1 , M_2 and M_3 (see Section 4.2).

The *random perturbation*, which is significantly stronger than the directed perturbation, consists in performing moves randomly selected from the set of moves M_4 (see Section 4.2). The degree of random perturbation can be adjusted by changing the value of parameter α ($0 < \alpha < 1$). If $\alpha \approx 0$, the random perturbation is very strong and can be compared to a random restart. If $\alpha \approx 1$, the strength of the random perturbation is insignificant.

4.4 Experimental Results and Comparisons

We report computational results using 6 instances from the BHOSLIB benchmark² and 11 instances from the more popular DIMACS benchmark³. To evaluate the significance of ADS, we compare the performances of AD-ILS, DIR-ILS and RND-ILS. For AD-ILS, the setting of the parameters used for adaptive perturbation is $P_0 = 0.9$ and $T = 2000$. $\alpha = 0.8$ for both AD-ILS and RND-ILS. For both AD-ILS and DIR-ILS, the coefficient ϕ for tabu tenure is set to 7. The number of perturbation moves L is set to $L = 0.05|V|$ for AD-ILS and DIR-ILS, while $L = 0.01|V|$ for RND-ILS. This setting of parameters is determined by a preliminary experiment. Each ILS version is independently executed 50 times, with the time-limit per run set to 90 minutes.

Table 2 reports the computational results. Column *BR* indicates the best-known or optimal (indicated with an asterisk) result. For each ILS, we report the best (column $|C|_{best}$) and the average result (column $|C|_{avg}$) obtained over 50 independent runs, as well as the average computing time in minutes required to reach the best reported result from column $|C|_{best}$. Next to the best-found clique value $|C|_{best}$, we indicate in parentheses the number of times the best-known solution was found over 50 executions.

Table 2. Computational comparison of AD-ILS with DIR-ILS and RND-ILS on 11 hard DIMACS instances and 6 large BHOSLIB instances

Instance Name	BR	AD-ILS			DIR-ILS			RND-ILS		
		$ C _{best}$	$ C _{avg}$	t(m)	$ C _{best}$	$ C _{avg}$	t(m)	$ C _{best}$	$ C _{avg}$	t(m)
brock800_1	23*	23(9)	21.36	43.8	23(4)	21.16	28.9	21(0)	20.98	4.0
brock800_2	24*	24(27)	22.62	34.6	24(4)	21.24	36.7	24(4)	21.24	25.6
brock800_3	25*	25(41)	24.46	41.6	25(15)	22.9	28.3	25(5)	22.3	47.5
brock800_4	26*	26(45)	25.5	22.5	26(37)	24.7	45.3	26(21)	23.1	43.7
C1000.9	68	68(50)	68.0	0.5	68(50)	68.0	0.1	59(0)	58.5	31.9
C2000.9	80	79(0)	77.66	55.0	79(0)	78.36	43.4	64(0)	62.88	44.3
keller6	59	59(50)	59.0	3.4	59(50)	59.0	0.5	50(0)	47.66	11.9
san1000	15*	15(12)	11.2	27.4	15(33)	13.34	30.9	15(1)	9.66	0.1
san400_0.7_1	40*	40(50)	40.0	18.5	40(50)	40.0	2.5	23(0)	21.84	43.5
san400_0.7_3	22*	22(50)	22.0	0.0	22(50)	22.0	0.0	22(50)	22.0	16.5
hamming10-4	40	40(50)	40.0	0.0	40(50)	40.0	0.0	38(0)	35.92	53.6
frb53-24-1	53*	53(2)	52.04	33.6	53(4)	52.08	36.0	46(0)	44.76	36.9
frb53-24-3	53*	53(22)	52.44	33.4	53(50)	53.0	20.7	46(0)	44.92	62.2
frb53-24-5	53*	53(37)	52.74	35.2	53(50)	53.0	17.8	46(0)	44.72	50.6
frb56-25-1	56*	56(1)	54.88	6.3	56(15)	55.3	33.2	49(0)	46.88	82.9
frb56-25-3	56*	56(2)	55.0	56.1	56(13)	55.26	56.7	48(0)	47.06	22.2
frb56-25-5	56*	56(33)	55.62	43.9	56(49)	55.98	10.9	48(0)	47.04	28.5

Like for QAP, AD-ILS significantly and statistically outperforms RND-ILS with a p -value = 3.751173e-04 according to the Posthoc test, which once again highlights the drawback of the classic random perturbation often used within the general ILS framework. However, the contribution of ADS is less significant in comparison with the directed perturbation strategy. Although both AD-ILS and DIR-ILS can attain the best-known result for all the used instances except for instance C2000.9, DIR-ILS outperforms AD-ILS in terms of average results

² http://iridia.ulb.ac.be/~fmscia/maximum_clique/BHOSLIB-benchmark

³ <http://cs.hbg.psu.edu/txn131/cliue.html>

on all the BHOSLIB instances and 2 DIMACS instances (*C2000.9* and *san1000*). However, AD-ILS shows better performance than DIR-ILS on the four hard *brock* instances. In Section 5, we justify these results with an analysis of distribution of high quality local optima. In terms of average computing times, the difference between AD-ILS and DIR-ILS is not very obvious.

5 Analysis

We observed from the computational comparisons (see sections 3.4 and 4.4) that the best performance with ILS is often obtained when directed (weaker) and random (stronger) perturbations are adaptively combined. On the other hand, the results also showed that for some instances (e.g., BHOSLIB instances from the MAX-CLQ benchmark) it is more useful to apply only the weak (directed) perturbation, while for several other instances (i.e., QAP real-life like instances) the best performance is achieved with random perturbation. In this section, we try to justify such results by investigating the minimal distances between pairs of medium or high quality local optima. To measure distance between solutions for QAP and MAX-CLQ, we use the well-known hamming distance. Medium and high quality local optima may be viewed as ‘strong’ attractors since it is more likely that they are visited during the search than a low quality local optimum. More precisely, given a set of medium or high quality local optima S , for all $lo_i \in S$ we determine the distance d_{min} between lo_i and some other solution $lo_j \in S$ which is the closest to lo_i , i.e., $d_{min} = \min_{lo_j \in S, lo_j \neq lo_i} d(lo_i, lo_j)$. For each possible distance $d_i \in [0, Max]$ (Max is the maximal distance), we then count the number of time that d_i is the distance between $lo \in S$ and another solution in S which is the closest to lo .

The results of this study for 4 MAX-CLQ instances and 3 QAP instances are given in Figure 1. The x -axis shows the normalized minimal distance between two ‘strong’ attractors, while the y -axis shows the number of pairs of ‘strong’ attractors separated by the given distance. Figure 1 indicates that there exists a significant difference in the distribution of medium and high quality local optima for QAP and MAX-CLQ instances. For *brock800-2*, *tai100b* and *sko100a* the minimal distances between two strong attractors are generally small, compared to instances *C2000.9*, *frb53-24-1* and *tai100a*. Intuitively, a weaker diversification introduced into the search for such instances may cause the search to cycle between ‘strong’ attractors that are not globally optimal solutions. For an effective solving of these instances, strong diversifications are required. On the other hand, for instances *C2000.9*, *frb53-24-1* and *tai100a*, the distribution of local optima prevents the search from cycling even with weak diversification. For this reason it may be worthwhile to perform a more intensive search. These observations justify to some extent why DIR-ILS provides the best performance on *C2000.9* and *frb* instances, while RND-ILS seems to be the best for real-life like instances (i.e., *tai80b*, *tai100b* and *tai150b*).

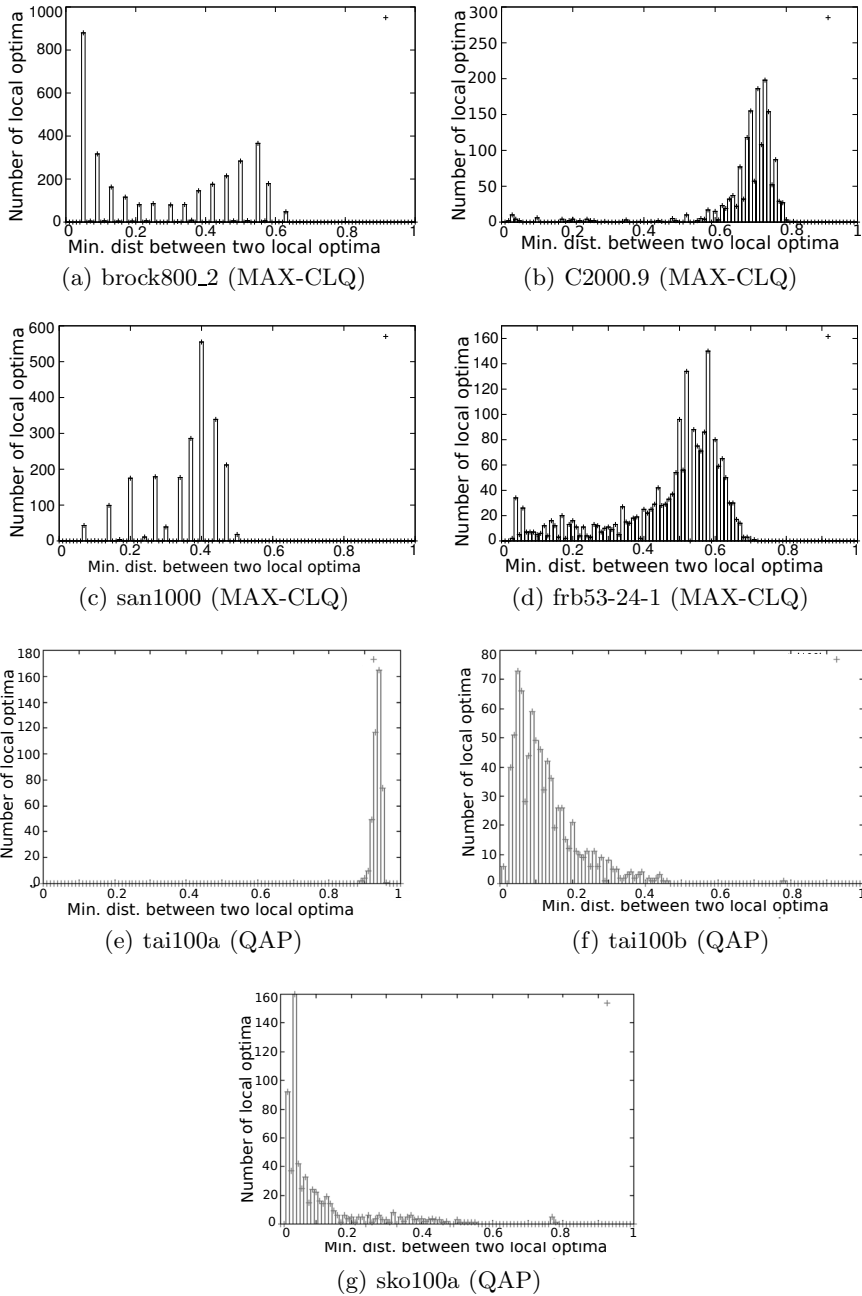


Fig. 1. Distribution of medium and high quality local optima (i.e., ‘strong’ attractors) for 4 MAX-CLQ and 3 QAP instances of different types and structures

6 Conclusion

The purpose of this paper is to investigate the performance of the adaptive diversification strategy (ADS) which constitutes an essential component of the recently proposed breakout local search (BLS). ADS adaptively applies a directed (weaker) and a random (stronger) perturbation according to the current search progress. We integrated ADS into the basic iterated local search (ILS) framework and evaluated its performance on the quadratic assignment problem (QAP) and the maximum clique problem (MAX-CLQ). Numerical results showed that the AD-ILS outperforms the standard ILS based on random moves on almost all the tested instances, which highlights the drawback of this classic perturbation strategy. Moreover, AD-ILS outperforms on most QAP instances and on several hard MAX-CLQ instances the ILS version which applies solely directed perturbation moves. We performed an analysis of the distribution of local optima to provide some guidance on when to introduce a weaker and when to introduce a stronger diversification into the search.

Acknowledgment. We are grateful to the referees for their comments. The work is partially supported by the Pays de la Loire Region (France) within the RaDaPop (2009-2013) and LigeRO (2010-2013) projects.

References

1. Aarts, E.H.L., Lenstra, J.K.: Simulated Annealing. In: *Local Search in Combinatorial Optimization*, ch. 1, pp. 1–17. Wiley (1997)
2. Battiti, R., Protasi, M.: Reactive Search, a history-based heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics* 2 (1996)
3. Benlic, U., Hao, J.-K.: A Study of Breakout Local Search for the Minimum Sum Coloring Problem. In: Bui, L.T., Ong, Y.S., Hoai, N.X., Ishibuchi, H., Suganthan, P.N. (eds.) *SEAL 2012*. LNCS, vol. 7673, pp. 128–137. Springer, Heidelberg (2012)
4. Benlic, U., Hao, J.K.: Breakout local search for maximum clique problems. *Computers & Operations Research* 40(1), 192–206 (2013)
5. Benlic, U., Hao, J.K.: Breakout local search for the max-cut problem. *Engineering Applications of Artificial Intelligence* (in press, 2013)
6. Benlic, U., Hao, J.K.: Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation* 219(9), 4800–4815 (2013)
7. Glover, F., Laguna, M.: *Tabu Search*, 408 p. Kluwer Academic Publishers, Boston (1998) ISBN: 0-7923-8187-4
8. Kelly, J.P., Laguna, M., Glover, F.: A study of diversification strategies for the quadratic assignment problem. *Computers & Operations Research* 21(8), 885–893 (1994)
9. Lourenco, H.R., Martin, O., Stützle, T.: Iterated local search. In: *Handbook of Meta-heuristics*. Springer, Heidelberg (2003)
10. Taillard, E.: Robust taboo search for the quadratic assignment problem. *Parallel Computing* 17, 443–455 (1991)

Adaptive MOEA/D for QoS-Based Web Service Composition

Mihai Suci¹, Denis Pallez², Marcel Cremene³, and Dumitru Dumitrescu¹

¹ Babes-Bolyai University, Cluj Napoca, Romania

² University of Nice Sophia-Antipolis

³ Technical University of Cluj-Napoca, Romania
`mihai.suciu@ubbcluj.ro`

Abstract. QoS aware service composition is one of the main research problem related to *Service Oriented Computing (SOC)*. A certain functionality may be offered by several services having different Quality of Service (QoS) attributes. Although the QoS optimization problem is multiobjective by its nature, most approaches are based on single-objective optimization. Compared to single-objective algorithms, multiobjective evolutionary algorithms have the main advantage that the user has the possibility to select *a posteriori* one of the Pareto optimal solutions. A major challenge that arises is the dynamic nature of the problem of composing web services. The algorithms performance is highly influenced by the parameter settings. Manual tuning of these parameters is not feasible. An evolutionary multiobjective algorithm based on decomposition for solving this problem is proposed. To address the dynamic nature of this problem we consider the hybridization between an adaptive heuristics and the multiobjective algorithm. The proposed approach outperforms state of the art algorithms.

1 Introduction

The composition of web services with optimal Quality of Service (QoS) parameters is a well known problem in the service oriented computing field. Given a business workflow that includes a set of abstract services and a set of concrete service, that implements each abstract service, the goal is to find the optimal combination of concrete services based on their QoS parameters. Given m abstract services and n concrete services for each abstract service, there are n^m possibilities.

The search space is a discrete one since for each abstract service we need to chose one concrete service (any combination is possible). This is a combinatorial optimization problem. Finding the solution with the optimal QoS is an NP-hard problem.

The problem stated previously is well known in domains like Service Oriented Computing (SOC) and Search-based Software Engineering (SBSE) [3], [6], [17], [25]. Various solutions are proposed based on different approaches such as: integer programming, genetic and hill climbing algorithms [1], [3], [29], [19].

Despite the fact that the QoS optimization problem is multiobjective by nature few approaches based on multiobjective algorithms can be found in the literature [15], [22], [24], [26], [28]. In most cases single-objective algorithms are used to solve this problem. The user might prefer to see several good solutions (Pareto optimal) and decide which is the best for himself. Criteria aggregation offers only one solution. It is more natural to let the user decide the importance of each objective than aggregating the objectives and ask the user to specify *a priori* his/her preferences (this is not a trivial task). By using multiobjective optimization, it is not necessary to define *a priori* an aggregation function. For solving the QoS web composition problem few applications based on multiobjective optimization algorithms can be found in the literature

There are many variants of EAs which have different control parameters: population size, operators used, crossover and mutation probabilities, etc. Selecting appropriate values is mainly done based on empirical studies, often a "trial and error" fashion is used for adjusting the values. Typically one parameter is adjusted at a time, which may lead to sub-optimal choices, since often it is not known how the parameters interact. Such an approach is time consuming. In the last couple of years there has been an increasing interest in designing methods that self-adapt these parameters [4], [9], [18].

The hybridization between a decomposition based multiobjective optimization algorithm and an adaptive technique is proposed. An algorithm based on a decomposition technique [2], [13] seems appropriate for solving this problem. The new approach is validated on some well known multiobjective test problems, and then we apply it to the web service composition problem. We compare our results with some state of the art algorithms from the literature.

2 Multiobjective Optimization Prerequisites

Let us consider m objectives defined by the set $\{f_i\}_{i \in \{1, \dots, m\}}$ of real valued functions $f_i : X \rightarrow \mathbb{R}, X \subseteq \mathbb{R}^n$. $F : X \rightarrow \mathbb{R}^m$ is the vector valued function $F(x) = (f_1(x), \dots, f_m(x))$. A *Multiobjective Optimization Problem (MOP)* can be defined as: find $x^* = (x_1, \dots, x_n)$ which optimizes the vector function $F(x)$ and satisfies the defined constraints.

In the case of minimization the standard MOP may be written as:

$$MOP : \begin{cases} \text{minimize } F(x) = (f_1(x), f_2(x), \dots, f_m(x)) \\ \text{subject to: } g_i(x) \geq 0, \quad i=1,2,\dots,k, \\ \quad \quad \quad h_j(x) = 0, \quad j=k+1,\dots,q. \end{cases} \quad (1)$$

where $x = (x_1, \dots, x_n) \in X$.

x is a *decision vector*. X represents the *parameter space* and Y is the *objective space*.

A decision vector $x \in X$ is said to *Pareto-dominate* $y \in X$ (x is better than y), denoted as $x \prec y$, if and only if $\forall i \in 1, \dots, m, f_i(x) \leq f_i(y)$, and $\exists j \in 1, \dots, n$ such that $f_j(x) < f_j(y)$.

A solution $x \in X$ is *Pareto-optimal* if and only if $\nexists y \in X$ such that $y \prec x$.

The *Pareto-Optimal Set* (POS) is defined as the set of all Pareto-optimal solutions $POS = \{x \in X | \nexists y \in X, y \prec x\}$.

The *Pareto-Optimal Front* (POF) is defined as the set of all objective values corresponding to the solution in POS. $POF = \{F(x) | x \text{ is nondominated}\}$.

The QoS-based service optimization is a combinatorial multiobjective optimization problem. Using a decomposition technique many-criteria problems may be decomposed in a class of single-objective problems [2], [13]. Decomposition techniques have good performance for combinatorial problems, another advantage is the small computational load. One drawback to decomposition evolutionary approach is the dependency between the problem type and the algorithm parameters. The same algorithm must solve different instances of this problem. One instance is represented by the business workflow that describes the web services composition (the interconnections of the composing web services, see [20] for more details). Different web services are described by different workflows, thus the search space changes for each workflow making it a very dynamical problem. A set of parameters that work for one particular instance of the workflow may not yield good results for another workflow. It would be very difficult to tune parameters for each particular workflow. A self-adaptive technique seems the obvious choice for this kind of problem.

2.1 MOEA/D Technique

MOEA/D [30] is a multiobjective algorithm based on decomposition. It is a simple and powerful scalarizing based algorithm. MOEA/Ds advantages are scalability with the number of objectives, computational efficiency, high performance for combinatorial optimization problems.

It uses the weighted Tchebycheff approach in order to decompose the MOP in a number of single-objective problems, each problem is represented by an individual in the current population. The Tchebycheff norm is:

$$g_{\lambda}(x, z) = \max_{i=1, \dots, m} \{\lambda_i |f_i(x) - z_i|\}, \forall x \in X$$

where z is an optimal point, the goal is to minimize g_{λ} .

Each sub-problem is characterized by different weight vectors. The number of uniformly distributed weight vectors used is equal to the number of sub-problems that are to be optimized. The number of weights can be computed as $N = \binom{H+m-1}{m-1}$ where H is a predefined integer. These vectors satisfy the conditions:

$$\sum \lambda_i = 1 \text{ and } \lambda_i \in \{0, \frac{1}{H}, \dots, \frac{H}{H}\}, i = 1, \dots, m.$$

Another key feature of MOEA/D is the use of neighboring solutions for generating offsprings more efficiently. Based on Euclidean distance for each λ weight vector T neighbors are computed. The set of T neighbors of λ is denoted by $B(\lambda)$. From this set the parents are selected. For each weight vector λ one offspring is generated by crossover and mutation. If the offspring is better than its parents it replaces them in the current population. A key step here is that the offspring

is also compared with each neighbor in $B(\lambda)$ based on its decomposition, if it is better it replaces that neighbor. This is an elitism step, it assures that only the best solutions propagate through the search.

An archive is used to store non-dominated solutions. The archive has no effect on the search. It is used only for storing the best individuals found during each generation. If a newly generated offspring is better than an individual inside the archive it simply replaces it.

Some adaptive techniques for MOEA/D parameters have been proposed in [5], [11], [12],[16], [31]. These techniques try to self adapt the algorithms parameters: weight vectors, neighbour size T , recombined individuals.

As a scalarizing function MOEA/D uses the Tchebycheff approach. In [11] the possibility of using different decomposition techniques, according to the complexity of the problem, is explored.

In [5] an adaptive mechanism for selective mating is proposed. The decomposed subproblems are classified in solved and unsolved, a subproblem is considered as solved if it is not improved in α generation. In the current generation if a subproblem is solved it is skipped, the unsolved ones are recombined and evaluated. Also the mating pool is adjusted. The mating candidates are selected according to Euclidean distance in decision space.

An adaptive scheme for weight vectors generation is proposed in [12]. Here the weight vectors are adapted according to the geometrical characteristics of the Pareto front. Uniform distributed vectors are generated using Mixture Uniform Design. The hypervolume is used to evaluate these vectors and the Simplex Method is used to adapt them in order to maximize the hypervolume.

In [31] an adaptation scheme for the neighborhood size is proposed, it is shown that the adaptive version gives better results than the classic MOEA/D with the neighborhood size fixed.

All approaches do not consider the adaptation of the evolutionary mechanism used. So in this paper we address the adaptation of the parameters of the evolutionary mechanism of MOEA/D and we apply this new approach to web services composition.

2.2 Adaptation of Differential Evolution

Differential Evolution [23] is a continuous function evolutionary algorithm. There are many adaptive versions of Differential Evolution [7]. From all the techniques CoDE [27] and SaDE [21] seem the most simple and efficient.

Self adaptive Differential Evolution (*SaDE*) [21] is an adaptation technique that uses the experience from previous generations. It adapts the trial vector generation strategies and the parameters C_r and F . It assigns a probability to each generation strategies, after a predefined number of generations these probabilities are update by taking into account the best strategies (strategies that generate individuals that pass to the next generation are assigned a higher probability). Normally the algorithm uses four different trial vector generation strategies: *DE/rand/1/bin*, *DE/rand-to-best/2/bin*, *DE/rand/2/bin*, and

DE/current-to-rand/1. For the sake of simplicity for the multiobjective version we use only: *DE/rand/1/bin* and *DE/best/2/bin*.

Each strategy is initialized with a probability $p_i = 0.5$ and after a learning period LP this probability is updated according to [21]. *SaDE* also adapts Cr and F parameters. Cr is more sensible to problem type and complexity while F parameter is tied to convergence speed. The algorithm performance depends on Cr values and usually good values lie in a small interval. Cr values are random generated according to a distribution with mean values Cr_m and deviation 0.1 , initially $Cr_m = 0.5$. After 5 generations new Cr values are randomly generated using the distribution with $\mu = Cr_m$ and $\sigma = 0.1$. For 25 generations this process is repeated keeping Cr values for which trial vectors are selected in favor of parents to advance to the next generation. After these 25 generation the new mean for the distribution is computed using the successful Cr values discovered. F takes random values in the interval $(0, 2]$ with $\mu = 0.5$ and $\sigma = 0.3$.

Within most DE variants one strategy for generating the trial vector is used per generation and only one control parameter setting for each trial vector. For this reason the search ability of the algorithm could be limited. An improved version of adaptive DE is proposed in [27]. By combining several effective trial vector generation strategies with some suitable control parameter settings better results can be obtained. This new method is called Composite DE (*CoDE*) and it uses three trial vector generation strategies and three control parameter settings that are randomly combined to generate the trial vector.

Three trial vector strategies are chosen: *DE/rand/1/bin*, *DE/rand/2/bin*, and *DE/current-to-rand/1*. The three control parameter settings proposed: (a) [$F = 1.0, Cr = 0.1$], (b) [$F = 1.0, Cr = 0.9$], (c) [$F = 0.8, Cr = 0.2$].

In each generation for each target vector three trial vectors are generated using the above strategies and a random control parameter setting chosen from the candidate pool. The best one is selected for the next generation.

The values in the parameter settings pool are chosen because they exhibit some specific advantages: a large value for Cr encourages diversity because little information is inherited from the target vector. For small Cr values the trial vector differs from the target vector only by one gene thus optimizing each parameter independently. For this case better results are obtained for separable problems. Large F values increase diversity thus promoting exploration and low values focus the search on neighborhoods increasing exploitation.

3 Proposed Approach

As the QoS-based web service optimization problem is a combinatorial one we use *MOEA/D* algorithm to solve it. To cope with the dynamic nature of the QoS web service composition problem we endow *MOEA/D* with an adaptation mechanism. *MOEA/D* is based on DE. Some very simple and yet powerful adaptation techniques for DE have been propose [18]. We propose two adaptive variants of *MOEA/D* obtained by considering the DE adaptive mechanisms *SaDE* [21] and *CoDE* [27]. The new models are called *MOEA/D_C* (Algorithm 1) and *MOEA/D_S* (Algorithm 2).

Algorithm 1. Adaptive *MOEA/D_C*

input : N , T - number of sub-problems, neighborhood size
output: EP - external population that holds the non-dominate solutions

- 1 Initialization: $EP = \emptyset$, generate weight vectors and compute $B(\lambda)$;
- 2 **for** $i \leftarrow 1$ **to** N **do**
- 3 generate 3 offsprings using a random combination between a trial vector generation strategy and the control parameters;
- 4 update the neighboring solutions;
- 5 update z^* and EP ;
- 6 If stopping criteria is satisfied output the EP . Otherwise go to step 2;

Algorithm 2. Adaptive *MOEA/D_S*

input : N , T , LP - number of sub-problems, neighborhood size, learning period
output: EP - external population that holds the non-dominate solutions

- 1 Initialization: $EP = \emptyset$, generate weight vectors and compute $B(\lambda)$, $Cr_m = 0.5$;
- 2 **for** $i \leftarrow 1$ **to** N **do**
- 3 generate 2 offsprings based on the strategies *rand/1/bin* and *best/2/bin*;
- 4 update the neighboring solutions;
- 5 update z^* and EP ;
- 6 after LP generations update Cr_m and the probabilities p_i for the trial vector generation strategies;
- 7 If stopping criteria is satisfied output the EP . Otherwise go to step 2;

In *MOEA/D_C* the DE trial vector generation strategy *DE/rand/1/bin* used in *MOEA/D* is replaced with the *CoDE* strategy - three trial vectors are created and the best one is kept (lines 3-4). In *MOEA/D_S* the DE trial vector generation strategy is replaced with the *SaDE* strategy and after LP generations the parameter Cr_m and the probabilities for using each trial vector generation strategy from the *SaDE* candidate pool are computed (line 6). By using an adaptive scheme we avoid the drawback of manual tuning the algorithm parameters for each specific workflow.

The genome we use for our problem is depicted in figure 1. It consists of an array of integer values and has the length equal to the number of abstract services. Each gene stores the index of the concrete service that realizes the corresponding abstract service.

4 Numerical Experiments

For validating the new adaptive approaches we compare the classic version of *MOEA/D* with *MOEA/D_C* and *MOEA/D_S*. As a basis for comparison the *WFG* test suite [10] is considered. All *WFG* problems considered are bi-objective with $k = 4$ and $n = 24$. For all algorithms we use a population size $N = 100$, a neighborhood $T = 8$ and a maximum number of generations $g = 250$. For the

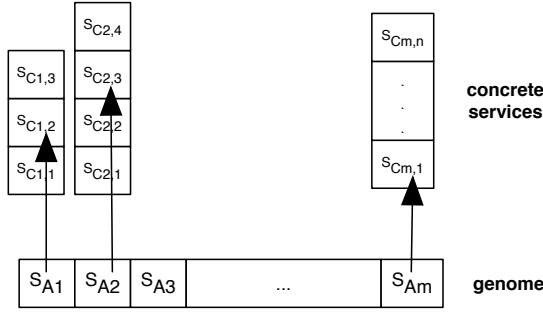


Fig. 1. Genome encoding

classic version of *MOEA/D*: $Cr = 0.7$, $F = 0.2$, and for *MOEA/D_S* we use a learning period $LP = 25$. The best results are obtained with these parameter values.

We run each algorithm for 50 times and then we compute on the final population the mean values for the quality indicators: Inverted Generational Distance (IGD) and Hypervolume (HV). The IGD computes the average distance of the reference Pareto set (P^*) to the nearest solution in the solution set found (A). IGD indicates the spread of A , small values are desirable.

$$IGD(A, P^*) = \frac{1}{|P|} \sum_{u \in P} \min_{v \in A} d(u, v)$$

The hypervolume represents the surface covered by the solution set and a reference point, high values mean that the front is near to the theoretical front and it assures a good spread. The reference point for the HV indicator is $(7, 7)$.

These values are presented in tables 1 and 2. The standard deviation for all experiments is less than 10^{-4} . From this tables we can see that *MOEA/D_C* performs better for all WFG problems, it gives higher HV values. The IGD is the same for *MOEA/D* and *MOEA/D_C* for six of the eight problems and better than *MOEA/D_S*. The classic version of the algorithm outperforms the adaptive counterparts only for the WFG1 problem.

The advantages of the adaptive approach is illustrated in Figures 2, 3 for a stopping criterion of $g_{max} = 400$ generations. Higher solution diversity is assured because the DE parameters for crossover and mutation are chosen to balance the search and exploitation. The Pareto front found by the adaptive version is closer to the theoretical front. From our experiments we observe also an advantage in convergence speed, at the same generation the front found by the adaptive approach is closer to the real front.

After validating our approach we apply it to the QoS-based web service composition. We compute the hypervolume indicator for various test scenarios.

Because *MOEA/D_C* gives better results than the classic version of *MOEA/D* and *MOEA/D_S* we apply it to the QoS-based web service composition problem.

Table 1. IGD and HV for *MOEA/D*, *MOEA/D_C*, and *MOEA/D_S* algorithms (average values over 50 independent runs). WFG 1-5 test problems are considered.

WFG	1		2		3		4		5	
Alg.	IGD	HV	IGD	HV	IGD	HV	IGD	HV	IGD	HV
<i>MOEA/D</i>	0.002	31.126	0.002	42.358	1.645	43.766	0.006	40.12	0.009	40.213
<i>MOEA/D_C</i>	0.003	34.057	0.002	43.266	1.597	43.968	0.006	41.827	0.009	41.486
<i>MOEA/D_S</i>	0.005	29.144	0.003	34.84	6.236	36.457	0.006	36.55	0.009	39.62

Table 2. IGD and HV values for *MOEA/D*, *MOEA/D_C*, and *MOEA/D_S* algorithms (average values over 50 independent runs). WFG 6-8 test problems are considered.

WFG	6		7		8	
Alg.	IGD	HV	IGD	HV	IGD	HV
<i>MOEA/D</i>	0.009	41.358	0.01	42.091	0.004	40.652
<i>MOEA/D_C</i>	0.009	41.486	0.01	42.242	0.004	40.782
<i>MOEA/D_S</i>	0.009	39.355	0.009	34.71	0.004	33.162

We conducted experiments for 12 scenarios that include all combinations of $m \in \{10, 20, 30, 40\}$ abstract services and $n \in \{20, 30, 50\}$ concrete services. Three QoS parameters are considered: access time to a web service and cost (these two objectives need to be minimized), and the third objective is user rating (that needs to be maximized). We compare *MOEA/D_C* with NSGA2 [8], and GDE3 [14] algorithms. For all algorithms the population was limited to 100 individuals which were evolved for 400 generations.

Figure 4 presents a case with a business workflow of $m = 20$ abstract services each of them having $n = 20$ concrete alternative services. For a low complexity workflow the GDE3 algorithm finds a more diverse final non-dominated set.

If we increase the complexity of the workflow to $m = 40$ abstract services and $n = 40$ concrete alternative services for each abstract service, the adaptive version is able to assure good solution diversity and find non-dominated solutions compared to the other algorithms. Figure 5 depicts these results.

Table 3 presents the Hypervolume indicator values for some scenarios. It can be observed that for low complexity business workflows the classic non-adaptive algorithms produce better results. For $m \in \{10, 20\}$ GDE3 is better. When we increase the complexity of the problem, $m \in \{30, 40\}$, the search space is considerably bigger, the adaptive version assures better performance with respect to the hypervolume indicator. The hypervolumes sometimes increase although the problems complexity is higher because the larger search space yields solutions that are unavailable to smaller search spaces.

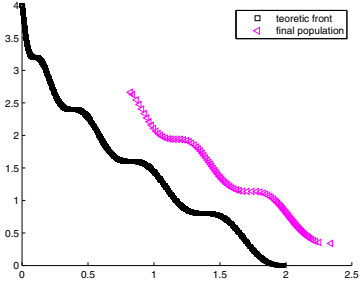


Fig. 2. *MOEA/D* final population and theoretical front for WFG1 test problem

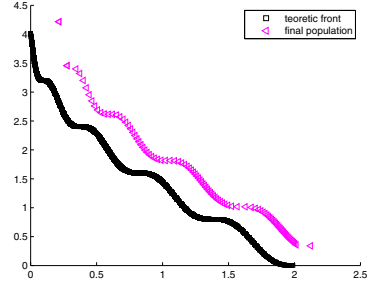


Fig. 3. *MOEA/DC* final population and theoretical front for WFG1 test problem

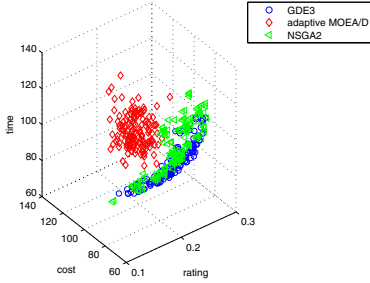


Fig. 4. Final populations for GDE3, NSGA2, and *MOEA/DC* algorithms for a business workflow with $m = 20$ abstract services and $n = 20$ concrete services

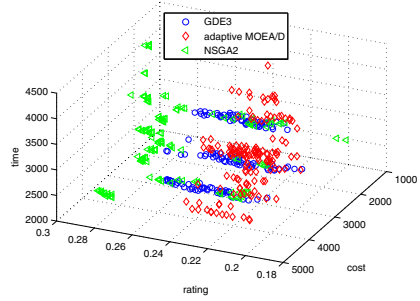


Fig. 5. Final populations for GDE3, NSGA2, and *MOEA/DC* algorithms for a business workflow with $m = 40$ abstract services and $n = 40$ concrete services

Table 3. Hypervolume indicator values for different workflow complexities $m \in \{10, 20, 30, 40\}$ and $n \in \{20, 50\}$ (mean values for 30 independent runs)

Algorithm	n/m							
	10/20	10/50	20/20	20/50	30/20	30/50	40/20	40/50
GDE3	652.84	480.12	450.92	465.2	465.4	440.3	410.3	411.5
NSGA2	649.49	475.28	444.12	461.7	461.8	424.2	402.6	390.9
<i>MOEA/DC</i>	620.38	468.14	439.1	455.2	470.3	480.6	468.1	465.7

5 Conclusions

An adaptive approach for the NP-hard problem of composing web services based on their Quality of Service properties is proposed. An evolutionary multiobjective optimization approach for QoS problem is considered. A new adaptive version of MOEA/D is proposed addressing the considered combinatorial problem.

We compare the proposed algorithm with the classic version of MOEA/D then apply it to the QoS-based service composition problem and some state of the art multiobjective algorithms are considered for comparison.

The results show the potential of this approach. Better performance is obtained (with respect to multiobjective quality indicators) when the adaptive approach is applied to standard test problems and some business workflows of high complexity.

Acknowledgements. This research was supported by the national project code TE 252 financed by the Romanian Ministry of Education and Research CNCSIS-UEFISCSU. The first would like to thank for the financial support provided from program co-financed by the Sectoral Operational Programme Human Resources Development, Contract POSDRU/107/1.5/S/76841 with the title Modern Doctoral Studies: Internationalization and Interdisciplinarity.

References

1. Bahadori, S., Kafi, S., Far, K.Z., Khayyambashi, M.R.: Optimal web service composition using hybrid ga-tabu search. *Journal of Theoretical and Applied Information Technology* 9(1) (2009)
2. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(1), 238–252 (1962)
3. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 1069–1075 (2005)
4. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent Developments. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) *Adaptive and Multilevel Metaheuristics*. SCI, vol. 136, pp. 3–29. Springer, Heidelberg (2008)
5. Chiang, T.-C., Lai, Y.-P.: Moea/d-ams: Improving moea/d by an adaptive mating selection mechanism. In: *IEEE Congress on Evolutionary Computation, CEC 2011*, pp. 1473–1480. IEEE (2011)
6. Comes, D., Baraki, H., Reichle, R., Zapf, M., Geihs, K.: Heuristic Approaches for QoS-Based Service Selection. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) *ICSOC 2010*. LNCS, vol. 6470, pp. 441–455. Springer, Heidelberg (2010)
7. Das, S., Suganthan, P.N.: Differential evolution: A survey of the state-of-the-art. *IEEE Trans. Evolutionary Computation* 15(1), 4–31 (2011)
8. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation* 6(2), 182–197 (2002)

9. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 3(2), 124–141 (1999)
10. Huband, S., Hingston, P., Barone, L., While, L.: A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation* 10(5), 477–506 (2006)
11. Ishibuchi, H., Sakane, Y., Tsukamoto, N., Nojima, Y.: Adaptation of Scalarizing Functions in MOEA/D: An Adaptive Scalarizing Function-Based Multiobjective Evolutionary Algorithm. In: Ehrgott, M., Fonseca, C.M., Gandibleux, X., Hao, J.-K., Sevaux, M. (eds.) *EMO 2009*. LNCS, vol. 5467, pp. 438–452. Springer, Heidelberg (2009)
12. Jiang, S., Cai, Z., Zhang, J., Ong, Y.-S.: Multiobjective optimization by decomposition with pareto-adaptive weight vectors. In: *ICNC*, pp. 1260–1264 (2011)
13. Kathrin, K., Tind, J.: Constrained optimization using multiple objective programming. *Journal of Global Optimization* 37, 325–355 (2007)
14. Kukkonen, S., Lampinen, J.: GDE3: the third evolution step of generalized differential evolution. In: *Congress on Evolutionary Computation*, pp. 443–450 (2005)
15. Li, L., Cheng, P., Ou, L., Zhang, Z.: Applying Multi-objective Evolutionary Algorithms to QoS-Aware Web Service Composition. In: Cao, L., Zhong, J., Feng, Y. (eds.) *ADMA 2010, Part II*. LNCS, vol. 6441, pp. 270–281. Springer, Heidelberg (2010)
16. Liu, B., Fernández, F.V., Zhang, Q., Pak, M., Sipahi, S., Gielen, G.G.E.: An enhanced MOEA/D-DE and its application to multiobjective analog cell sizing. In: *IEEE Congress on Evolutionary Computation*, pp. 1–7 (2010)
17. Liu, X., Xu, Z., Yang, L.: Independent global constraints-aware web service composition optimization based on genetic algorithm. In: *IATED International Conference on Intelligent Information Systems*, pp. 52–55 (2009)
18. Neri, F., Tirronen, V.: Recent advances in differential evolution: a survey and experimental analysis. *Artif. Intell. Rev.* 33(1-2), 61–106 (2010)
19. Parejo, J.A., Fernandez, P., Cortes, A.R.: Qos-aware services composition using tabu search and hybrid genetic algorithms. *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos* 2(1), 55–66 (2008)
20. Pop, F.-C., Pallez, D., Cremene, M., Tettamanzi, A., Suci, M.A., Vaida, M.-F.: Qos-based service optimization using differential evolution. In: *GECCO*, pp. 1891–1898 (2011)
21. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation* 13(2), 398–417 (2009)
22. Ross, S.M.: *Introduction to Probability Models*, 9th edn. Academic Press, Inc., Orlando (2006)
23. Storn, R., Price, K.: Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization* 11, 341–359 (1997)
24. Taboada, H.A., Espiritu, J.F., Coit, D.W.: MOMS-GA: A Multi-Objective Multi-State Genetic Algorithm for System Reliability Optimization Design Problems. *IEEE Transactions on Reliability* 57(1), 182–191 (2008)
25. Vanrompay, Y., Rigole, P., Berbers, Y.: Genetic algorithm-based optimization of service composition and deployment. In: *Proceedings of the 3rd International Workshop on Services Integration in Pervasive Environments, SIPE 2008*, pp. 13–18 (2008)

26. Wada, H., Champrasert, P., Suzuki, J., Oba, K.: Multiobjective Optimization of SLA-Aware Service Composition. In: Proceedings of the 2008 IEEE Congress on Services - Part I, SERVICES 2008, pp. 368–375. IEEE Computer Society (2008)
27. Wang, Y., Cai, Z., Zhang, Q.: Differential evolution with composite trial vector generation strategies and control parameters. *IEEE Trans. Evolutionary Computation* 15(1), 55–66 (2011)
28. Yao, Y., Chen, H.: QoS-aware service composition using NSGA-II. In: Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ICIS 2009, pp. 358–363 (2009)
29. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng.* 30, 311–327 (2004)
30. Zhang, Q., Li, H.: MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation* 11, 712–731 (2007)
31. Zhao, S.-Z., Suganthan, P.N., Zhang, Q.: Decomposition-based multiobjective evolutionary algorithm with an ensemble of neighborhood sizes. *IEEE Trans. Evolutionary Computation* 16(3), 442–446 (2012)

An Analysis of Local Search for the Bi-objective Bidimensional Knapsack Problem

Leonardo C.T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
{lleonaci,manuel.lopez-ibanez,stuetzle}@ulb.ac.be

Abstract. Local search techniques are increasingly often used in multi-objective combinatorial optimization due to their ability to improve the performance of metaheuristics. The efficiency of multi-objective local search techniques heavily depends on factors such as (i) neighborhood operators, (ii) pivoting rules and (iii) bias towards good regions of the objective space. In this work, we conduct an extensive experimental campaign to analyze such factors in a Pareto local search (PLS) algorithm for the bi-objective bidimensional knapsack problem (bBKP). In the first set of experiments, we investigate PLS as a stand-alone algorithm, starting from random and greedy solutions. In the second set, we analyze PLS as a post-optimization procedure.

1 Introduction

The efficiency of many successful heuristic algorithms for combinatorial optimization problems (COPs) is based on the proper use of local search procedures. In fact, many metaheuristics have incorporated the possibility of using local search for example as daemon actions in ant colony optimization (ACO) and as improvement procedures in genetic algorithms.

Pareto local search (PLS) [13] is a straightforward but effective extension of single-objective local search to multi-objective problems. Given a set of solutions, a PLS algorithm consists of selecting one solution at a time and exploring its neighborhood, thus, generating new solutions. These new solutions are added to the initial set, dominated solutions are eliminated, and the algorithm continues until each of the solutions in the solution set has been explored.

The performance of PLS algorithms usually tends to depend on (i) the quality of the input solutions, (ii) the definition of the neighborhood structure, (iii) the pivoting rule used for exploring of the neighborhood and the possible use of candidate lists, and (iv) restrictions on the set of solutions to keep the runtimes manageable. For such reasons, PLS algorithms are well suited for analyzing the impact of design features on the performance of local search procedures for multi-objective combinatorial optimization problems (MCOPs).

In this paper, we implement a PLS algorithm for the bi-objective bidimensional knapsack problem (bBKP), using the local search components commonly found in the literature. Full factorial designs are used to investigate factors and their eventual interactions. In the first set of experiments, we investigate PLS

as a stand-alone optimization method. The experimental setup used aims at isolating the effect of the initial solution set, and the effect of the neighborhood size. Results confirm PLS’s dependence on high-quality input solutions, and that large neighborhoods have to be combined with candidate lists to limit exploration and keep runtimes reasonable. In the second set of experiments, we analyze PLS as a post-optimization method. Two algorithms are used for generating input solutions: (i) a simply greedy procedure, and (ii) AutoMOACO, the best-performing multi-objective ant colony optimization (MOACO) algorithm for the bBKP. Results show that PLS is able to significantly improve the quality and size of the approximation fronts generated by both algorithms.

The remainder of this paper is organized as follows. Section 2 introduces some basic definitions and presents PLS. Section 3 defines the bBKP and presents the PLS algorithm implemented. Section 4 explains the experimental setup, while the experimental results are discussed in Sections 5 and 6. Finally, conclusions and possibilities for future work are discussed in Section 7.

2 Pareto Local Search

Pareto local search (PLS) is a stochastic local search method for tackling multi-objective problems based on a natural extension of single-objective local search approaches [13]. To fully understand PLS, some background notions on multi-objective optimization are required.

In an MCOP, solutions are compared not based on a single objective value, but on a vector of objective values. Given a maximization problem with objectives $f^i, i = 1, \dots, k$, a solution s is said to be better (or to *dominate*) another solution s' if $\forall i, f^i(s) \geq f^i(s')$ and $\exists i, f^i(s) > f^i(s')$. If neither solution dominates the other, they are said to be *nondominated*. The goal of multi-objective optimizers is to find the set of nondominated solutions w.r.t. all feasible solutions, the Pareto set. Since this may prove to be computationally unfeasible, multi-objective metaheuristics have been used to find approximation sets, i.e., sets whose image in the objective space best approximates the Pareto set image.

PLS algorithms use the concept of dominance to extend single-objective local search to multi-objective problems. Starting from an initial set of solutions \mathcal{A}_0 (which can also be a singleton), PLS selects at each iteration an unexplored solution $s \in \mathcal{A}$, the current set of non-dominated solutions, and explores the neighborhood of s , generating new solutions. If these new solutions satisfy the algorithm’s acceptance criterion, they are added to \mathcal{A} . Once the neighborhood of s is explored, s is marked as explored. The algorithm stops when all solutions in \mathcal{A} have been explored. The three main steps of PLS as shown in Algorithm 1 can be summarized as follows:

Selecting a solution to be explored. The method `NextSolution` chooses the next solution to be explored. The original PLS chooses the next solution uniformly at random. More recently, other possibilities have been considered, such as selection based on the *optimistic hypervolume improvement* [6].

Algorithm 1. Pareto Local Search**Input:** An initial set of nondominated solutions \mathcal{A}_0

```

1: explored(s) ← FALSE  ∀ s ∈  $\mathcal{A}_0$ 
2:  $\mathcal{A} \leftarrow \mathcal{A}_0$ 
3: repeat
4:   s ← NextSolution( $\mathcal{A}_0$ )
5:   for all s' ∈ Neighborhood(s) do
6:     if Acceptance(s, s',  $\mathcal{A}$ ) then
7:       explored(s') ← FALSE
8:        $\mathcal{A} \leftarrow \text{Update}(\mathcal{A}_0, s')$ 
9:     end if
10:  end for
11:  explored(s) ← TRUE
12:   $\mathcal{A}_0 \leftarrow \{s \in \mathcal{A} \mid \text{explored}(s) = \text{FALSE}\}$ 
13: until  $\mathcal{A}_0 = \emptyset$ 

```

Output: \mathcal{A}

Exploring the neighborhood. Given a solution s , the method $\text{Neighborhood}(s)$ generates the set of neighbor solutions. Two pivoting rules are useful here: (i) *first*, where the neighborhood exploration stops at the first accepted neighbor, or; (ii) *full*, where the neighborhood of s is explored fully and all possible neighbors of s are examined.

Accepting new solutions. Given a solution s and a neighbor solution s' , the method $\text{Acceptance}(s, s', \mathcal{A})$ determines whether s' is considered an acceptable solution or not. Two common possibilities are: (i) *dominance*, where s' is only accepted if s' dominates s , or; (ii) *nondominance*, where s' is accepted in case s' is nondominated w.r.t. \mathcal{A} . The first criterion generates a higher pressure towards good solutions but it may lead to early stagnation. When using nondominance, the output is likely a well distributed set, but it may lead to high computation times.

Dubois-Lacoste et al. [7] present a review of PLS, highlighting its use both as a stand-alone procedure and in hybrid algorithms. Regarding stand-alone PLS, the authors identify studies focusing on time-limited experiments [10], on any-time behavior [6] and on how to restart or continue the search after PLS converges [1,9,4]. Regarding PLS as a post optimization procedure, Dubois-Lacoste et al. [5] have proposed algorithms that are currently state-of-the-art for several bi-objective flowshop problems.

3 Applying PLS to the bBKP

The bi-objective bidimensional knapsack problem is a widely-used bi-objective benchmark problem [14,12], and is a special case of the general multi-objective multidimensional knapsack problem (moMKP), which is formalized as follows:

$$\max f^c(x) = \sum_{i=1}^n p_i^c x_i \quad c = 1, \dots, k \quad \text{s.t.} \quad \sum_{i=1}^n w_i^j x_i \leq W_j \quad j = 1, \dots, m \quad (1)$$

where each item i has k profits and m costs, f^c is the c -th component of the objective vector f , n is the number of items, p_i^c is the c -th profit of item i , w_i^j is

Table 1. Methods used for computing the weights used by SolutionOrdering

Method	Formula	Description
<i>equal</i>	$\lambda = 0.5$	equal weights
<i>random-discrete</i>	$\lambda \in \{0, 1\}$	uniformly randomly chosen
<i>random-continuous</i>	$\lambda \in [0, 1]$	uniformly randomly chosen
<i>largest-gap</i>	$\lambda = i, \min(w_s^i)$	privileges dimension with more free space
<i>smallest-gap</i>	$\lambda = i, \max(w_s^i)$	privileges dimension with less free space
<i>highest-profit</i>	$\lambda = i, \max(f^i(s))$	privileges objective with the highest value
<i>lowest-profit</i>	$\lambda = i, \min(f^i(s))$	privileges objective with the lowest value
<i>proportional-same</i>	$\lambda = w_s^1 / (w_s^1 + w_s^2)$	proportional to the loads
<i>proportional-opposite</i>	$\lambda = w_s^2 / (w_s^1 + w_s^2)$	inversely proportional to the loads

the j -th cost of item i , W_j is the j -th capacity of the knapsack, and $x_i \in \{0, 1\}$ defines if item i is included in the knapsack ($x_i = 1$) or not ($x_i = 0$). The set of feasible solutions is $X \subseteq \{0, 1\}^n$. The bBKP is a special case of the moMKP where $k = m = 2$.

In this paper, a solution x for the bBKP is also represented as a list s of size n , where the first $n_s \leq n$ items are considered to be in the knapsack. Furthermore, each solution has as an associated profit vector $\mathbf{p}_s = (p_s^1, p_s^2)$, where $p_s^c = \sum_{i=1}^n p_i^c x_i^s$, $c = 1, 2$, and a load vector $\mathbf{w}_s = (w_s^1, w_s^2)$, where $w_s^j = \sum_{i=1}^n w_i^j x_i^s$, $j = 1, 2$.

Two methods have been implemented for generating the initial solution(s) for PLS: (i) *random*, where one or more random solutions are generated, and (ii) *greedy*, where a set of greedy solutions is generated. Random solutions are generated by choosing an item uniformly at random at each construction step, until no more items can be added due to the capacity constraints. When using greedy solutions, a set of linearly uniformly distributed weights $\Lambda = \{\lambda_1, \dots, \lambda_z\}$ is generated, where z is the number of input solutions. For each $\lambda \in \Lambda$, a greedy solution is generated using one of the following heuristic functions [12]:

$$\eta_1(i) = \frac{\lambda p_i^1 + (1 - \lambda) p_i^2}{\sum_{j=1}^m w_i^j} \quad \eta_2(i) = \frac{\lambda p_i^1 + (1 - \lambda) p_i^2}{\sum_{j=1}^m \frac{w_i^j}{W_j - w_s^j + 1}} \quad (2)$$

All actions related to neighborhood exploration are encapsulated in the procedure **Neighborhood**. This procedure systematically explores the neighborhood of a solution s , returning a set of neighbors. The neighborhood operator used is the r -remove operator, which removes up to r items from the knapsack. In our solution representation, removing one item at the i -th position of the list means exchanging it with the last selected item, i.e., the item at position n_s of the list, and decreasing the value of n_s by one.

Solutions are reconstructed by filling the knapsack with items found at positions $i = n_s + r, \dots, n$ of the list, in the order they appear. Since biasing the search

is important, in the beginning of the algorithm items not selected in the input solution are ordered. Formally, given an input solution s , let $\mathcal{IN}(s) = \{i \mid s_i = 1\}$ be the set of n_s items inside the knapsack and $\mathcal{OUT}(s) = \{i \mid s_i = 0\}$ be the set of items outside the knapsack. The procedure **SolutionOrdering** is used to order items $x_i \in \mathcal{OUT}(s)$ in a nondecreasing order according to their heuristic value. The heuristics used for this ordering are the same presented in Eq. 2. The weights used by the heuristic functions are generated on a *per solution* basis. Given a solution s , we tested nine methods for computing λ (Table 1).

For efficiency, candidate lists are used to constrain the set of items that are considered for removal; in other words, items that are not member of the candidate list are never considered for removal in a current solution. Given a solution s , the candidate list of items for removal contains the L last items of the list, i.e., items at positions $i, n_s - L < i \leq n_s$. Two methods have been used for determining parameter L : (i) *all*, where $L = n_s$, and (ii) *input*, where L is input by the user.

The pseudocode for **Neighborhood** can be seen on Algorithm 2. C_r stands for a combination of r items, whereas C_* stands for a combinations of any number of items. **Accepted** checks if s and s' are nondominated. Finally, the search is controlled by one of the following pivoting-rules:

1. *remove-first*: given that the first accepted neighbor is generated by the removal of the item found at the i -th position of s , $n_s - L < i \leq n_s$, **Neighborhood** does not explore the insertion possibilities generated by the removal of items found at the j -th position of s , $\forall j, n_s - L < j < i$;
2. *remove-full*: **Neighborhood** explores the insertion possibilities generated by the removal of each of the items in the candidate list.
3. *insert-first*: given the insertion possibilities generated by the removal of an item x_i , **Neighborhood** stops at the first combination of items that produces an accepted neighbor.
4. *insert-full*: given the insertion possibilities resulting from the removal of an item x_i , **Neighborhood** generates all acceptable neighbors.

The pseudocode for the bBKP-PLS algorithm can be seen on Algorithm 3. The algorithm initially generates (reads) the input set of solutions (line 1). Then, a weight λ is generated for each solution $s \in \mathcal{A}_0$ (line 3), and used in the **SolutionOrdering** procedure (line 4). Finally, PLS is called (line 6). In this paper, we consider random selection of solutions in PLS and acceptance based on nondominance. In addition, when **Neighborhood** checks the nondominance between the current solution and each of its neighbors, it also checks nondominance w.r.t. to the set \mathcal{A} . This is done to ensure that \mathcal{A} is extended by at least one non-dominated solution if such a solution exists in the neighborhood of s .

4 Experimental Setup

For the analysis of PLS, we use 50 bBKP instances for each size $n \in \{100, 250, 500, 750\}$ [2] as a test set. We use a full factorial design, and each configuration

Algorithm 2. Procedure Neighborhood

Input: An input solution s

- 1: $\mathcal{N} \leftarrow \emptyset$
- 2: **for all** $C_r \in \text{candlist}(s)$ **do**
- 3: $s' \leftarrow \text{Remove}(s, C_r)$
- 4: **for all** $C_* \in \text{OUT}(s)$ **do**
- 5: $s'' \leftarrow \text{Insert}(s', C_*)$
- 6: **if** $\text{Accepted}(s'', s')$ **then**
- 7: $\mathcal{N} \leftarrow \mathcal{N} \cup s''$
- 8: **if** *insertion-first* **then**
- 9: $\text{found} \leftarrow \text{true}$; **break**;
- 10: **end if**
- 11: **end if**
- 12: **end for**
- 13: **if** found and *removal-first* **then**
- 14: **break**
- 15: **end if**
- 16: **end for**

Output: A set of neighbor solutions \mathcal{N}

Algorithm 3. bBKP-PLS

Input: An input method $\text{input} \in \{\text{random}, \text{greedy}\}$

- 1: $\mathcal{A}_0 \leftarrow \text{InitialSolutions}(\text{input})$
- 2: **for all** $s \in \mathcal{A}_0$ **do**
- 3: $\lambda \leftarrow \text{GenerateLambda}(s)$
- 4: $\text{SolutionOrdering}(s, \lambda)$
- 5: **end for**
- 6: $\mathcal{A} \leftarrow \text{PLS}(\mathcal{A}_0)$

Output: \mathcal{A}

is run 10 times per instance. All parameters are analyzed both individually and for possible interactions through plots of the median hypervolume [14] of the approximation sets they produce.

For the computation of the hypervolume measure of an approximation set, a reference point is required. Here, we first normalize the objective vectors of all sets generated by all runs of all configurations separately for each instance. This normalization also transforms the original maximization problem into a minimization one, with objective values in the range $[1, 2]$, due to requirements of the software we use [8]. We use the point $(2.1, 2.1)$ as the reference point.

When comparing two algorithms using boxplots of their hypervolumes or plots of the differences of their empirical attainment functions (EAFs) [11], we use the four instances by Zitzler and Thiele [14] of sizes $n \in \{100, 250, 500, 750\}$, called ZTZ instances, and we run the algorithms 25 independent times per instance. When more algorithms are simultaneously compared, we use the boxplots of the hypervolumes for a sample of 4 instances of each size: the ZTZ instances plus three instances from the test set used for the analysis of PLS.

All algorithms are implemented in C and all experiments are run on a single core of Intel Xeon E5410 CPUs, running at 2.33GHz with 6MB of cache size under Cluster Rocks Linux version 6.0/CentOS 6.3. In the paper, only few representative results are given. The complete set of results are made available as a supplementary page [3].

Table 2. Parameter values used for the analysis of stand-alone PLS

Parameter	Values
η	η_1, η_2
λ	<i>equal, random-discrete, random-continuous, largest-gap, smallest-gap, highest-profit, lowest-profit, proportional-same, proportion-opposite</i>
candidate list	<i>all, L = 15, L = 30, L = 50</i>
removal rule	<i>removal-first, removal-full</i>
insertion rule	<i>insertion-first, insertion-full</i>

5 Experiments with Stand-Alone PLS

In this section, we present the results of the analysis of stand-alone PLS. To properly isolate the effect of the neighborhood operator, this analysis is divided in two stages: (i) experiments removing one item ($r = 1$), and (ii) experiments removing more than one item ($r > 1$). The parameter space used for this analysis comprises all possible values previously described, summarized in Table 2.

5.1 Removing a Single Item

The different possibilities of choosing the weights (λ) and the heuristic information (η) do not have a strong influence on the results, hence, we do not present here a detailed analysis of these parameters. Instead, we analyze the initialization method, the length of the candidate list, and the removal and insertion pivoting rules. The results presented here are aggregated across all possible settings of λ and η . Detailed results can be found on the supplementary material.

We first analyze stand-alone PLS initialized with a single random solution. Fig. 1 (left) shows the median hypervolume on the y-axis, and the instances grouped by sizes on the x-axis. The lines represent the median hypervolume obtained by different configurations, and are ordered according to performance on the larger instance. It is clear that using candidate lists when $r = 1$ is a bad decision, since the hypervolume quickly degenerates as the instance size grows. However, as shown on Fig. 1 (right), the runtimes of configurations that do not use candidate list tend to grow very quickly (y-axis).

The analysis of PLS starting from greedy solutions requires an additional parameter, namely the number of weights used for generating input solutions. Experiments were conducted for 2, 10 and 50 input weights, and this parameter proved critical for the performance of the algorithm. When only two input weights are used, the performance of the algorithm is really poor and similar to when a random initial solution is used. On the other hand, using 50 weights not only improves the final result quality, but also helps the algorithm converge faster compared to other settings.

Therefore, we focus on the experiments using 50 initial weights for generating the greedy solutions. Also in this case, not using candidate lists leads to long runtimes. However, the solution quality does not degenerate when larger values

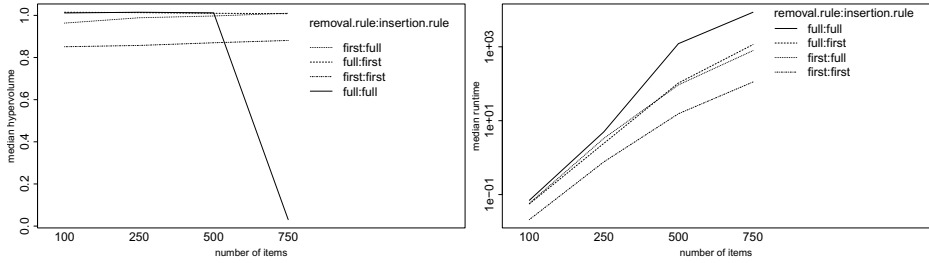


Fig. 1. Median hypervolume (left) and runtime (right) of different combinations of pivoting rules for PLS starting from a random initial solution and $r = 1$

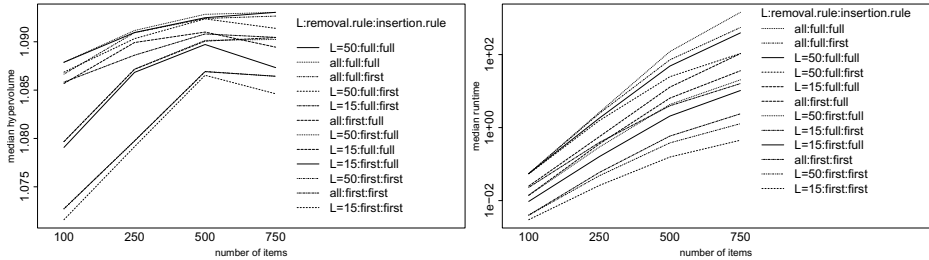


Fig. 2. Median hypervolume (left) and runtime (right) of different combinations of pivoting rule and candidate list sizes for PLS starting from greedy solutions and $r = 1$

of L are adopted. Fig. 2 (left) shows the median hypervolume (y-axis) grouped by instance sizes (x-axis). The best performing versions are the ones that use $L = \{30, 50\}$, *removal-full* and *insertion* $\in \{first, full\}$ ($L = 30$ not shown here due to space reasons). When analyzing runtimes (see also Fig. 2, right), a candidate list of size 50 combined with *removal-full* and *insertion-first* is a setting that takes computation times similar to those that are used as time limits in the analysis of state-of-the-art algorithms [2,12].

5.2 Removing More Than One Item

For the analysis of PLS with $r > 1$, the same parameter space used in the previous experiments is adopted. However, given that in the previous experiments with $r = 1$ we observed that the parameters used for *SolutionOrdering* (that is, the heuristic and the values of λ) behave very similarly, we narrowed down the number of configurations to be tested by selecting η_1 as the heuristic function and *highest-profit* as the method for defining λ . The experiments were limited to a maximum runtime of 5 hours per run.

Figure 3 shows the results for $r = 2$. In terms of solution quality, there is a big difference between configurations that use *removal-first* and *insertion-first*

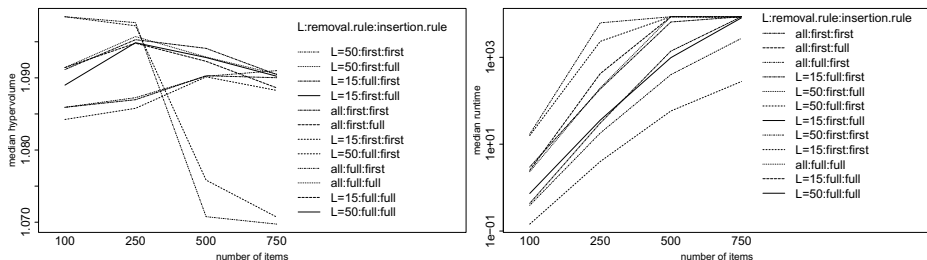


Fig. 3. Median hypervolume (left) and runtime (right) of different combinations of pivoting rule and candidate list sizes for PLS starting from greedy solutions and $r = 2$

and the ones that do not. Moreover, the best configurations for small instance sizes become much worse with larger instance sizes. The reason is that those configurations reach the CPU-time limit of 5 hours and were stopped before completion. In fact, the only configurations with runtime lower than 1 000 seconds are the ones that either (i) use a candidate list ($L \leq 50$) combined with *removal-first* and *insertion-first*, or; (ii) use a candidate list with $L = 15$ combined with *removal-first* and *insertion-full*.

6 Experiments with PLS as Post-optimization Method

To analyze PLS as a post-optimization method, we start by comparing PLS against the greedy procedure used for generating its input solutions. The motivation for this comparison is to understand whether PLS is actually significantly improving the input set or simply adding more nondominated solutions. The greedy procedure is run with η_1 and $2n$ weights, which is roughly the same number of solutions expected to be found by stand-alone PLS. The parameters used by PLS are: η_1 , *highest-profit*, $L = 50$, *removal-full*, *insertion-first*, and $r = 1$.

Fig. 4 shows that the difference between the greedy procedure and PLS (using $2n$ weights for greedy solutions) is quite strong. The approximation set identified by PLS dominates the output of the greedy procedure across the entire range of the front, which means PLS is able to substantially improve all initial solutions. In addition, PLS finds a much larger approximation set.

We also add PLS as a post-optimization procedure to AutoMOACO [2], a high-performing population-based algorithm for bBKP and run AutoMOACO using the same parameters and time limit as in the original paper. The resulting approximation set is given as input to PLS, which is then run until completion using the same parameters as above. Fig. 5 shows the EAF difference for ZTZ 750. Again, PLS is able to improve the approximation fronts over the entire objective space, while the runtimes of PLS remain low (see Table 3). Thus, PLS significantly improves the approximation obtained by AutoMOACO, incurring only a reasonable computational overhead.

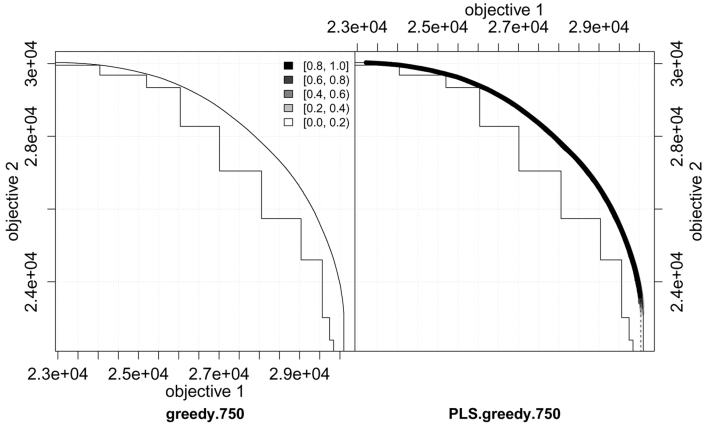


Fig. 4. EAF difference plot. Greedy solutions using $2n$ weights (Greedy) vs. PLS initialized with greedy solutions using $2n$ weights ($\text{PLS}_{\text{greedy}}$). Instance ZTZ 750.

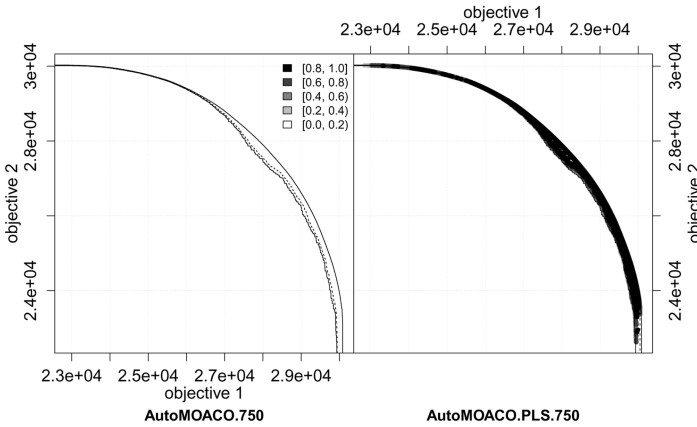
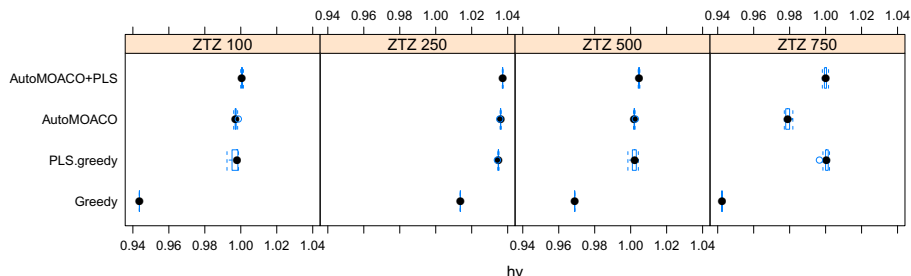


Fig. 5. EAF difference plot. AutoMOACO vs. AutoMOACO+PLS. Instance ZTZ 750.

To conclude this analysis, we compare all four algorithms considered in this section. Figure 6 shows the boxplot of the hypervolume for all four ZTZ instances. As expected, the greedy procedure presents the worst hypervolume values. Among the remaining algorithms, AutoMOACO and stand-alone PLS perform similarly on most instances. Interestingly, although the number of solutions is greatly increased when PLS is used as post-optimization for AutoMOACO, the differences in the hypervolume are relatively small. For the largest instance, the performance of AutoMOACO decreases considerably, but PLS is able to compensate such loss.

Table 3. Average number of solutions and runtime: ZTZ instances, 25 runs

	ZTZ 100	ZTZ 250	ZTZ 500	ZTZ 750
Greedy	13 (0.00s)	43 (0.01s)	69 (0.06s)	114 (0.12s)
PLS _{greedy}	98.81 (0.04s)	356.46 (1.08s)	742.38 (7.64s)	1502.2 (40.35s)
AutoMOACO	81.84 (1s)	287.84 (6.25s)	376 (26s)	273.08 (56.25s)
AutoMOACO+PLS	110.48 (1.03s)	382.76 (7.27s)	807.28 (33.48s)	1542.48 (114.6s)

**Fig. 6.** Boxplot of the hypervolume indicator for ZTZ instances

7 Conclusions and Future Work

In this paper, we have applied Pareto local search (PLS) to the biobjective bidimensional knapsack problem to analyze the impact of common local search components found in the literature, and to empirically investigate by how much it can improve over existing algorithms. Our results show that the performance of stand-alone PLS strongly depends on high-quality input solutions. However, such solutions can be generated without significant computational overhead using greedy (meta)heuristics. Large neighborhood sizes proved prohibitive w.r.t. computation time even when combined with a candidate list. Nevertheless, archiving mechanisms that constrain the size of the approximation set remain to be tested.

The insights from this research can be used to design better neighborhood operators for the bBKP. Additionally, a combination of more elaborate algorithms, such as iterated greedy with PLS could lead to state-of-the-art results.

Acknowledgments. The research leading to the results presented in this paper has received funding from the Meta-X project from the Scientific Research Directorate of the French Community of Belgium and from the FRFC project “Méthodes de recherche hybrides pour la résolution de problèmes complexes”. Leonardo C. T. Bezerra, Manuel López-Ibáñez and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a FRIA doctoral fellow, a postdoctoral researcher and a research associate, respectively.

References

1. Alsheddy, A., Tsang, E.: Guided Pareto local search and its application to the 0/1 multi-objective knapsack problems. In: Caserta, M., Voß, S. (eds.) MIC 2009. University of Hamburg, Hamburg (2010)
2. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: Automatic Generation of Multi-objective ACO Algorithms for the Bi-objective Knapsack. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 37–48. Springer, Heidelberg (2012)
3. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: An analysis of local search for the bi-objective bidimensional knapsack: Supplementary material (2012), <http://iridia.ulb.ac.be/supp/IridiaSupp2012-016/>
4. Drugan, M.M., Thierens, D.: Path-Guided Mutation for Stochastic Pareto Local Search Algorithms. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 485–495. Springer, Heidelberg (2010)
5. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research* 38(8), 1219–1236 (2011)
6. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Pareto Local Search Algorithms for Anytime Bi-objective Optimization. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 206–217. Springer, Heidelberg (2012)
7. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Combining Two Search Paradigms for Multi-objective Optimization: Two-Phase and Pareto Local Search. In: Talbi, E.-G. (ed.) Hybrid Metaheuristics. SCI, vol. 434, pp. 97–117. Springer, Heidelberg (2013)
8. Fonseca, C.M., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: CEC 2006, pp. 1157–1163. IEEE Press, Piscataway (2006)
9. Geiger, M.J.: Decision support for multi-objective flow shop scheduling by the Pareto iterated local search methodology. *Computers and Industrial Engineering* 61(3), 805–812 (2011)
10. Liefoghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., Talbi, E.G.: On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics* 18(2), 317–352 (2011)
11. López-Ibáñez, M., Paquete, L., Stützle, T.: Exploratory analysis of stochastic local search algorithms in biobjective optimization. In: Bartz-Beielstein, T., et al. (eds.) *Experimental Methods for the Analysis of Optimization Algorithms*, pp. 209–222. Springer, Berlin (2010)
12. Lust, T., Teghem, J.: The multiobjective multidimensional knapsack problem: a survey and a new approach. *Intern. Trans. in Oper. Res.* 19(4), 495–520 (2012)
13. Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the bi-objective traveling salesman problem: An experimental study. In: Gandibleux, et al. (eds.) *Metaheuristics for Multiobjective Optimisation*. LNEMS, pp. 177–200. Springer, Berlin (2004)
14. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto evolutionary algorithm. *IEEE Transactions on Evolutionary Computation* 3(4), 257–271 (1999)

An Artificial Immune System Based Approach for Solving the Nurse Re-rostering Problem

Broos Maenhout^{1,*} and Mario Vanhoucke^{1,2,3}

¹ Faculty of Economics and Business Administration, Ghent University,
Tweekerkenstraat 2, 9000 Gent, Belgium

{broos.maenhout,mario.vanhoucke}@ugent.be

² Operations and Technology Management Centre, Vlerick Leuven Gent
Management School, Reep 1, 9000 Gent, Belgium

³ University College London, Gower Street, London WC1E 6BT, United Kingdom

Abstract. Personnel resources can introduce uncertainty in the operational processes. Constructed personnel rosters can be disrupted and render infeasible rosters. Feasibility has to be restored by adapting the original announced personnel rosters. In this paper, an Artificial Immune System for the nurse re-rostering problem is presented. The proposed algorithm uses problem-specific and even roster-specific mechanisms which are inspired on the vertebrate immune system. We observe the performance of the different algorithmic components and compare the proposed procedure with the existing literature.

Keywords: Nurse re-rostering, Meta-heuristics.

1 Introduction

The personnel scheduler constructs a deterministic personnel roster that determines the line-of-work for each personnel member. When unexpected events lead to schedule disruptions and infeasibilities, rescheduling is necessary to update the activity schedule and restore its feasibility. A disruption in a personnel schedule is defined as an occurrence when an employee that is planned to work a specific task is unavailable. In that case, the original roster has to be modified as tasks cannot be operated below a minimum number of required staff.

In this paper we present a new reactive optimisation approach to cope with schedule disruptions for the nurse shift scheduling problem, which is NP-hard ([9]). A new roster should be constructed that resembles the original roster as much as possible and that is in line with the staffing requirements per shift and the imposed time-related constraints. We propose an artificial immune system (AIS) to revise and re-optimize a schedule for a set of heterogeneous nurses. The constructed meta-heuristic is population-based and the population elements go through a cycle of proliferation, hypermutation and an immune defense mechanism. Based on antigenic pattern recognition different response systems are set

* Corresponding author.

up to neutralise the pathogens and the infected cells. The key for success is the match between the antigen and the antibodies, i.e. the characteristics of nurse rosters are analysed and evaluated which invokes a specific immune response to improve the nurse roster. This metaphor inspired us to explore the abilities of AIS for the nurse re-rostering problem, which has not been done before.

The nurse re-rostering problem has received limited attention in the staff scheduling literature. [7,8] formulated the nurse re-rostering problem and proposed in both papers LP-based rounding heuristics. [9] and [10] developed an indirect genetic algorithm. The decoder is built upon a constructive heuristic that entails a sequential re-assignment of a list of all tasks to the nurses. [9] score the individuals only based on the similarity with the original roster, whereas [10] additionally incorporate fairness between nurses. [4] proposes a heuristic algorithm based on mathematical programming that tries to maximize the job satisfaction and the schedule similarity. [6] developed a direct genetic algorithm that operates on the set of pareto-optimal solution elements and introduced a variable neighbourhood search tailored to the nurse re-rostering problem.

The remainder of the paper is organised as follows. The problem under study is described in section 2. In section 3, we discuss the fundamentals of AIS and discuss the problem-specific implementation of these principles. In section 4, we discuss the algorithmic performance and compare the proposed optimisation procedure with the existing literature. In section 5, conclusions are drawn.

2 Problem Description

The nurse re-rostering problem assumes that the nurse rostering problem is solved. The constructed nurse rosters are the starting point. The re-rostering problem arises when a set of disruptions occur for which a nurse is unable to perform the originally assigned tasks on one or more future work days. In that case, these tasks must be performed by other nurses. Hence, the nurse re-rostering problem embodies a reactive approach in order to restore the feasibility of these duty timetables. The required re-construction of nurse rosters is undertaken along with the following scheduling constraints and requirements, i.e.

- The set of disrupted tasks must be performed by other nurses in a way that the *minimum staffing requirements* are met for each shift on each day.
- The new roster should be conform to all established *time-related requirements* as for the original nurse rostering problem, i.e. national legislation, institutional conditions and personal contract stipulations. These rules define (socially) acceptable schedules for the individual nurses.
- In the nurse re-rostering problem additional constraints are imposed restricting the *availability of nurses*, i.e. nurses may not be assigned to work tasks on the days they are absent due to e.g. vacation or schedule disruptions.

Apart from complying with all these constraints, the nurses should be assigned to shifts such that the quality of the reconstructed timetable is optimised. The schedule quality is measured by multiple objectives, i.e.

- The staffing requirements are imposed as soft constraints and we try to minimise the deficient number of nurses (*Objective 1*).
- Minimise additional labour costs as a result of overtime, inefficient over-staffing, the use of external or temporary nurses, etc (*Objective 2*).
- Minimise roster changes and satisfy the individual nurse preferences as best as possible (*Objective 3*). When re-rostering staff, the nurse preferences consist primarily of retaining the original nurses' individual shift assignments as much as possible. In some hospitals, the nurse scheduler additionally considers the original nurse preferences for re-rostering nursing staff.
- Minimise the number of assignment infeasibilities due to schedule disruptions (*Objective 4*).
- Distribute the workload as evenly as possible over the nurses such that the constructed individual nurse schedules are fair. In this way, fairness is maintained (or even improved) after reconstructing the nurse roster as nurses have to catch up with unperformed duties due to disruptions (*Objective 5*).

For a mathematical problem formulation we refer to [6].

3 The Artificial Immune System: Algorithmic Interpretation and Implementation

An Artificial Immune System (AIS) is an evolutionary algorithm proposed by [3] and inspired by the theory of immunology. The AIS for the nurse re-rostering problem is conceived as a population-based evolutionary algorithm (with population size n). After the initialisation of the population elements or immune cells (see section 3.1), the optimisation system first evaluates the immune cells by calculating the objective function value of the different objective function components (Objectives 1-5, see section 3.2). Then the system undergoes a cycle of selection, proliferation and hypermutation, immune response and receptor editing. The immune cells with an acceptable total objective function value are selected and proliferated, i.e. these immune cells divide themselves and are copied. There is no combination or crossover between these selected population elements. During this reproduction, the immune cells undergo a hypermutation process (see section 3.4), where parts of the cells (i.e. the nurse roster) are randomly changed. These newly introduced individuals are the antigens, which are labelled as self or nonself by evaluating their objective function value and identifying the nurse roster characteristics via the antibodies. For the nonself antigens the immune response is invoked (see section 3.5). Afterwards the population is updated according to the receptor editing process (see section 3.6). The proposed algorithm is schematically represented in figure 1. This conceptual metaphor is in line with the algorithmic interpretation of AIS by [1].

The algorithmic structure of an AIS resembles an evolutionary algorithm without crossover. Both, evolutionary algorithms and AIS use the concept of survival of the fittest as evolution strategy. In the AIS algorithm are the individuals that are not selected for cloning in a given generation replaced by new random individuals. This substantial injection of randomness in each evolution cycle helps

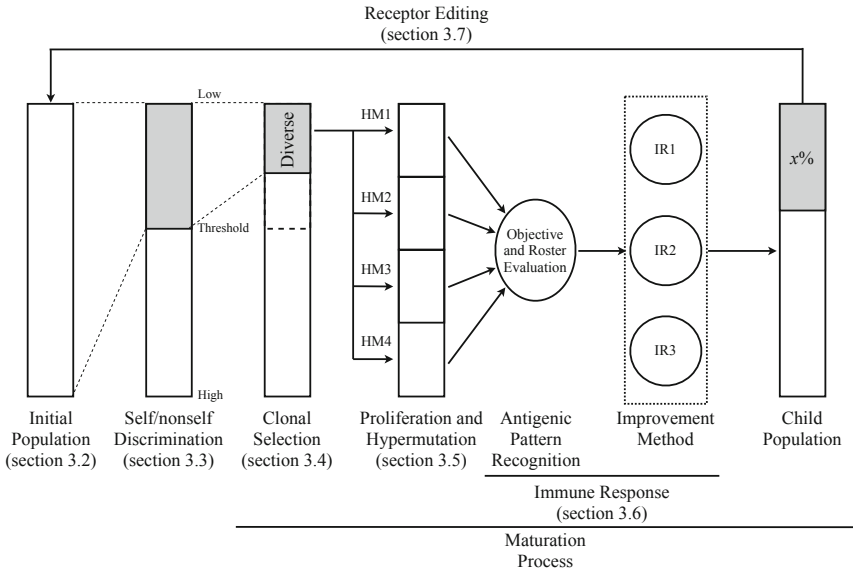


Fig. 1. A schematic representation of the AIS for the nurse re-rostering problem

the algorithm to explore the whole search space. Therefore, the concept of receptor editing avoids premature convergence.

3.1 Initialisation Procedure

In order to construct a diverse collection of elite solutions, the solution elements in the initial population are constructed in two different ways, i.e. completely random and with a constructive heuristic. The constructive heuristic is able to find high-quality solutions early in the search process. The heuristic schedules the set of nurses one by one in a random sequence taking the different objective function components into account. The individual roster line of each nurse is determined based on the partially constructed nurse roster (i.e. the roster lines for the nurses that are already scheduled). This scheduling of a particular nurse over the complete planning horizon is a resource constrained path problem.

3.2 Self/nonself Discrimination

The first step of the AIS procedure is pattern recognition where the objective function value of each component (objective 1-5) is calculated based on the number of violations (nonself cells) and the corresponding penalty. Whenever the total objective function value is lower than a fitness threshold value, we label the population element as a candidate for clonal selection.

3.3 Clonal Selection

Several population elements are selected to proliferate. The selection rate of each population element is proportional to its objective function value: the better the objective function value, the higher the selection probability. In this selection process diversification is stimulated as solutions that resemble the selected solution are ruled out for further selection based on a diversity threshold.

3.4 Proliferation and Hypermutation

Proliferation in the case of immune cells is asexual, a mitotic process where the cells divide themselves. In the context of an evolutionary algorithm, this implies that there is no combination or crossover with other population elements as the selected nurse rosters are integrally replicated. The degree of proliferation is directly proportional to the objective function value: the better the objective function value, the higher the number of duplicates generated. During this reproduction, the population elements undergo a hypermutation process according to which parts of the nurse roster are randomised. This random assignment is conform to all hard time-related constraints. In order to stimulate diversity among the different cloned cells, we apply four different mutation methods, i.e.

- Mutation of individual nurse schedules (HM1): This mutation method runs through the list of nurses. A nurse is possibly reassigned to a random nurse schedule.
- Mutation of day rosters (HM2): This mutation method runs through the days of the planning horizon. If selected, a random day roster is constructed for a particular day.
- Mutation of pairs of day rosters (HM3): This mutation method runs through the list of combinations of two days (e.g. day 1 and day 6). The assignments on a selected pair of days are swapped randomly between the nurses.
- Mutation of single assignments (HM4): This mutation method runs through the different assignments of the nurses over the days. A selected assignment is deleted and changed to another shift assignment.

We apply a single mutation method on each duplicated population element and each of these four methods has a 25% probability to be selected. In this way, a very diverse set of cells is constructed. An important parameter that is to be decided is the mutation rate, which indicates the percentage of a nurse roster that is subject to random changes. The mutation rate suffered by each immune cell during reproduction is inversely proportional to the affinity of the cell receptor with the antigen: the better the affinity, the smaller the mutation.

3.5 Immune Response

When required, we invoke an immune response or improvement method where different local search mechanisms are applied sequentially to improve the quality of a newly created population element. The quality of an AIS typically depends

on how well response mechanisms (i.e. the local search mechanisms) can be matched with the newly introduced nurse rosters (i.e. antigen). In this process is antigenic pattern recognition the first pre-requisite for the immune system to be activated and to mount an immune response. The recognition has to satisfy some criteria. First, the epitopes (i.e. nurse roster structure) and affinity (i.e. the objective function value) of the antigen are evaluated by the cell receptors of an immune cell (i.e. antibodies). An immune response is activated by an antibody depending on the characteristics and the affinity of the new individual.

Antigenic Pattern Recognition

The nature of an antigen is measured by the antigenic pattern recognition, which evaluates the roster structure and calculates the objective function value for each of the five objectives. For each objective function component, we apply one of the three improvement methods below if the specific threshold value τ_{obj} for the immune response is exceeded (i.e. nonself recognition). The order in which the different immune responses are invoked is based on the roster characteristics, i.e.

1. The evaluation of the nurse roster based on the different objective function components: The higher the objective function score on a particular component, the higher the priority to apply the corresponding immune response.
2. A factor var_{obj} which measures how the objective function value for a particular component is distributed over the different nurses and/or days. If $var_{obj} = 0$, the objective function value is evenly distributed over the nurses or days. If $var_{obj} = 1$, the objective function value is maximal for one or several days, and zero for all remaining days. In order to define the spread of the objective function violations, we use a general measure of variance that has been proposed by [11].

Improvement Method

We apply three local search algorithms that are proposed by [5] for the nurse scheduling problem, i.e.

- Immune response 1 (IR1): The pattern-based local search optimises the roster line of a particular nurse given the (fixed) roster lines of all other nurses.
- Immune response 2 (IR2): The day-based local search optimises a single day of the nurse roster given the (fixed) assignments of the nurses on all other days. This decomposition heuristic optimises the nurse roster day-by-day by optimally assigning a feasible shift assignment to each nurse. To that purpose, a linear assignment problem is defined and solved to optimality. In composing a linear assignment problem, we duplicate each shift column such that each shift has a number of columns that is equal to its coverage requirements. We add dummy nurses and dummy shift columns to allow over- and understaffing. The assignment costs are provided by the appropriate objective function coefficients. Infeasible assignments are given a very high cost such that these assignments will never be selected.

- Immune response 3 (IR3): The schedule-based local search focuses on the whole schedule for all nurses by swapping (sub-parts of) schedules between nurses. To that purpose, we define a linear assignment problem that optimally redistributes (sub-parts of) the shift patterns of the current population element among the nurses. Each (subpart of a) pattern can be assigned to each other personnel member at the account of an assignment cost that is composed out of the different objective function components. Infeasible assignments are given a very high assignment cost.

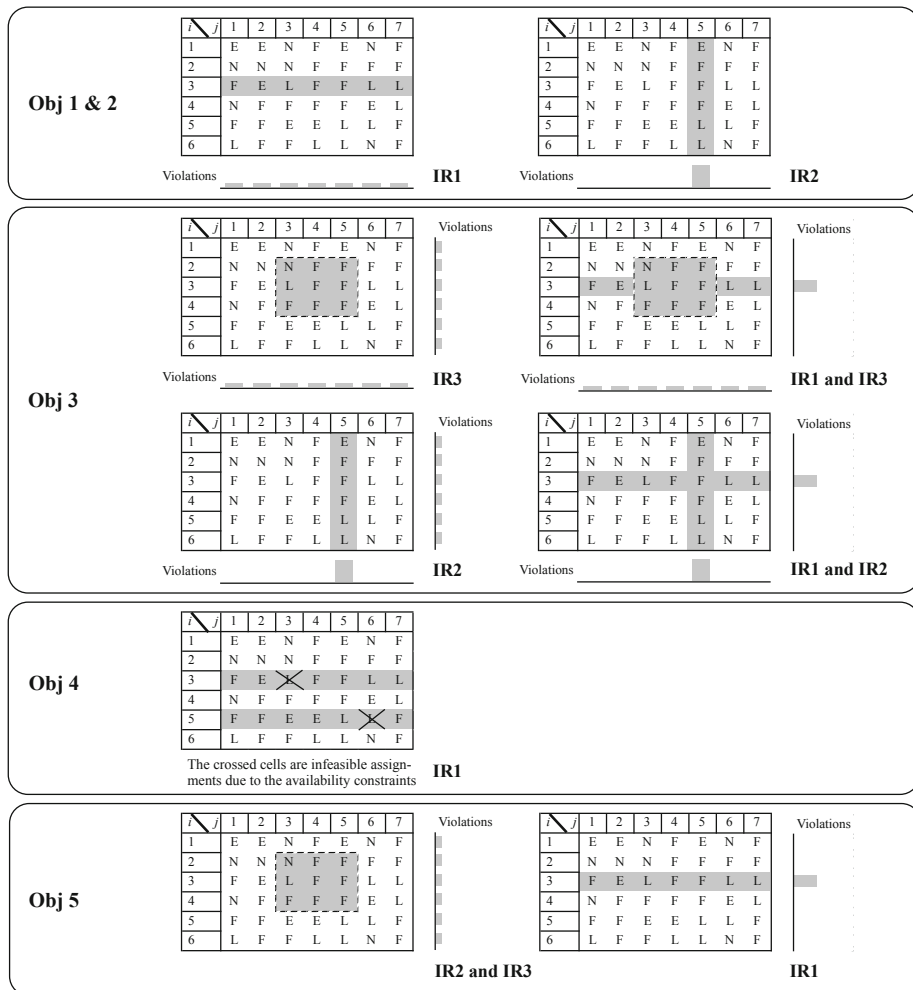


Fig. 2. Illustration of the match of the antigen and the immune response for the nurse re-rostering problem

The three local search algorithms decompose the original problem into smaller problems that are optimally solved and they can be perceived as complementary as each local search concentrates on a different part of the scheduling matrix, i.e. on the schedule of a single nurse, on a single day and on the complete matrix. The order in which the different parts of the nurse roster are optimised, depends on the characteristics of each roster area and the objective function component we want to improve. The roster area that shows the highest potential to improve, receives the highest priority and will be reoptimised first.

Match between immune response and antigen

As these local search methods in their purest form would require too much effort, the match between the antibody and antigen is very important in order to find the right balance between diversification and intensification. In figure 2 we give an overview of the objective function component and the corresponding immune response. We display an example nurse roster of 6 nurses and a period of 7 days. Each day the nurses are assigned to the early (E), late (L), night (N) or free (F) shift. The number of violations for each day (nurse) is displayed below (to the right of) each nurse roster. For illustrative purposes we display only the two extreme cases where all violations are evenly distributed over the days (nurses) and all violations coincide on one single day. If necessary, we calculate the factor var_{obj} based on the distribution of these violations and determine the appropriate immune response IR1, IR2 and/or IR3 using a breakpoint value. The shaded areas of the nurse roster are optimised by invoking the respective immune response, i.e. the shift-pattern of a single nurse (IR1), on a single day (IR2) and/or on the complete matrix (IR3). For objective 1, for example, we verify the number of times too few nurses are schedules and calculate the variance var_{obj1} of these violations over the days. If this variance is below the breakpoint value, IR1 is applied as the violations are evenly distributed over the planning horizon. If this variance is higher than the breakpoint value, IR2 is applied as the violations are concentrated on a couple of days.

3.6 Receptor Editing

After the application of the maturation process, the population is maintained and dynamically updated. $x\%$ of the original population elements is eliminated and replaced by new solutions with a better objective function value. In order to avoid the entrance of highly resembling solutions, the number of different assignments in the new solution should be higher than some threshold value.

4 Computational Experiments

In this section, we provide computational insights into the proposed AIS for the nurse re-rostering problem. In section 4.1, we describe the problem parameters and the test design. In section 4.2, we validate the beneficial performance of the proposed problem-specific operators. In section 4.3, we compare the performance

of the proposed procedure with the existing literature. All tests were carried out on a Dell Dual Core processor 2.8 Ghz and 2 Gb RAM.

4.1 Test Design

In order to test the performance of the proposed procedure rigorously, we utilise the test design and dataset constructed by [6]. The dataset comprises artificial nurse rosters with 30 full-time nurses, a planning horizon of 28 days and 3 working shifts. The schedule disruptions are generated by means of two input parameters, i.e. the number of disruptions and the spread of the disruptions over the days of the planning horizon. The dataset contains 864 problem instances. The nurse roster is re-constructed from the day of the first disruptions to the end of the planning horizon such that the different staffing constraints, time-related constraints and nurse availability constraints are respected. The following time-related constraints are imposed (see [2] and [10]), i.e.

- The nurses are assigned to only one working shift or to a day off for each day of the planning horizon.
- A minimum free timespan of 11 hours is imposed between working shifts (*forward rotation*).
- The number of working assignments is restricted (min 10, max 20).
- The number of consecutive working assignments is restricted (min 2, max 5).
- The number of assignments per shift type is restricted (min 0, max 20).
- The number of consecutive duties per shift type is restricted (min 1, max 5).

Table 1 provides the weights of the objective function components.

Table 1. Relative priorities and weights for the different objective function components

Objective	Description	Weight
1	Minimise the deficient number of nurses	10,000
2	Minimise additional labour costs	5,000
3	Minimise the roster changes	100
4	Minimise the number of assignment infeasibilities	100,000
5	Distribute the workload as evenly as possible	50

4.2 Algorithmic Performance

In this section, we analyse the effect of the implemented problem-specific principles. The results are displayed in table 2 under a stop criterion of 1,000 evaluated schedule solutions. For each alternative we display the average solution quality (Z), the percentage deviation from the best performing heuristic procedure ($\%Dev$) and the required CPU time (in seconds) (CPU). In order to test

Table 2. Computational results for different optimisation strategies (1,000 schedules)

Strategy		Overall	%Dev	CPU
Hypermutation	This procedure	27,631	0.00%	16.4
	HM1	36,354	31.57%	17.0
	HM2	30,593	10.72%	15.2
	HM3	31,518	14.07%	15.8
	HM4	37,260	34.85%	13.6
Immune response	This procedure	27,631	0.00%	16.4
	AIS_LS_Fixed	28,281	2.35%	19.8

the effect of the different strategies we start from the best performing heuristic procedure and implement a certain strategy or characteristic of the procedure.

Hypermutation - The results reveal that the proposed hypermutation method leads to significant better results compared to the four other approaches where the hypermutation process applies each time the same mutation method, i.e. HM1, HM2, HM3 or HM4. A detailed analysis revealed that this hypermutation method introduced a higher diversity in the search process as this method leads to a more diverse and higher number of unique new individuals.

Immune response - The results reveal that the local search method of [6] ('AIS_LS_Fixed'), which executes the three improvement heuristics in a fixed and sequential manner, performs 2.35% worse than the proposed improvement method, which selects the order and the type of the local search heuristics based on the solution quality and the characteristic of the roster at hand. The proposed approach needs a smaller number of steps to improve a nurse roster to an acceptable level as the right local search is invoked to improve a specific objective function component.

4.3 Benchmarking and Comparison with the Existing Literature

In this section, we compare our procedure with the indirect genetic algorithm of [10], the direct genetic algorithm of [6] and a multi-start heuristic. The multi-start heuristic constructs a random solution multiple times and applies the proposed immune response mechanism until the stop criterion is reached.

Table 3. Benchmark comparison with the existing literature (1,000 schedules)

Procedure	Overall	CPU
Pato and Moz (2008) (GA)	91,478	22.2
Maenhout and Vanhoucke (2011) (GA)	27,640	16.0
Multi-start heuristic	35,882	20.9
This procedure (AIS)	27,631	16.4

The results reveal that the proposed procedure outperforms the multi-start heuristic and the current literature. The comparison with the multi-start heuristic points out that introducing an evolutionary design and some intelligence in the diversification mechanism returns beneficial results. AIS is able to compete with the established technique of genetic algorithms as the procedure improves slightly the results of [6]. However, a detailed computational comparison reveals that the differences in the conceptual metaphor and algorithmic implementation lead to a distinct computational behaviour (see figure 3). We compare how both heuristics perform for the following problem characteristics, i.e.

- All: This category considers all instances.
- Spread: The categories (low, medium, high) are investigated (measured by the indicator SD with resp. values of 1, 0.5 and 0 (cfr. [6])).
- Disruptions: The categories (low, medium, high) are investigated (measured by the indicator TND with values of 0.01, 0.03 and 0.05 (cfr. [6])).
- Coverage: The categories (low, medium, high) are investigated (measured by the indicator TCC with values of 0.2, 0.35 and 0.5 (cfr. [12])).

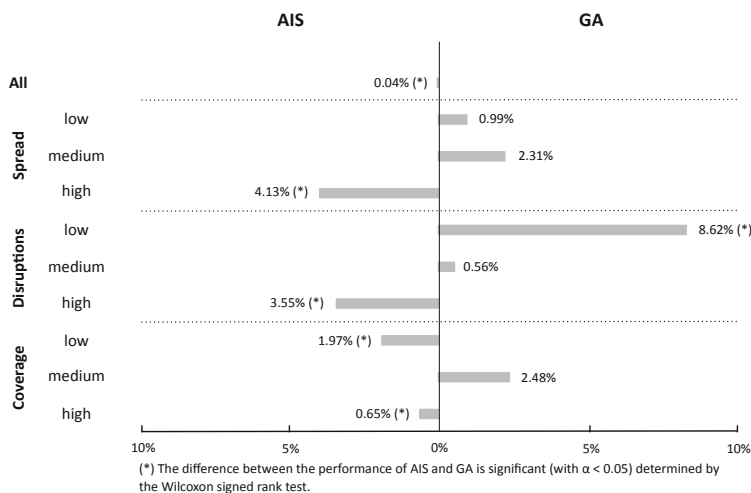


Fig. 3. A detailed computational comparison of AIS and GA of [6]

For these categories we measured the percentage difference in performance from the best performing heuristic and verified the significance using the Wilcoxon signed rank test. Overall, we observe that the proposed heuristic significantly outperforms the genetic algorithm of [6] with 0.04%. AIS is able to outperform the genetic algorithm significantly by 4.13% and 3.55% for resp. a high spread and a high number of disruptions. The same observations can be made for low and high staffing requirements. Hence, AIS shows a very different computational behaviour than the genetic algorithm of [6].

5 Conclusion

In this paper, we adapted the optimisation framework of AIS to solve the nurse re-rostering problem that revises and re-optimises a schedule of a set of heterogeneous nurses. The evolutionary algorithm is characterised by a hypermutation process that uses four different problem-specific mutation operators to insert diversification in the search process. The subsequent intensification process of the mutated rosters is and dependent on the characteristics of a specific roster. In this way, the meta-heuristic is able to make a trade-off between diversification and intensification. The computational results show the effectivity of the proposed approach and that the technique of artificial immune systems is competitive with more traditional optimisation frameworks like genetic algorithms.

References

1. Aickelin, U., Dasgupta, D.: Artificial immune systems Tutorial. In: *Introductory Tutorials in Optimisation*, pp. 1–29. Kluwer, New York (2005)
2. Burke, E., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The state of the art of nurse rostering. *Journal of Scheduling* 7, 441–499 (2004)
3. De Castro, L., Timmis, J.: Artificial immune systems: a novel paradigm for pattern recognition. In: *Artificial Neural Networks in Pattern Recognition*, pp. 67–84. University of Paisley, United Kingdom (2002)
4. Knighton, S.: *An Optimal Network-Based Approach to Scheduling and Rostering Continuous Heterogeneous Workforces*. PhD thesis, Arizona State University, Tempe, AZ (2005)
5. Maenhout, B., Vanhoucke, M.: An electromagnetic meta-heuristic for the nurse scheduling problem. *Journal of Heuristics* 13, 359–385 (2007)
6. Maenhout, B., Vanhoucke, M.: An evolutionary approach for the nurse re-rostering problem. *Computer & Operations Research* 38, 1400–1411 (2011)
7. Moz, M., Pato, M.: An integer multicommodity flow model applied to the re-rostering of nurse schedules. *Annals of Operations Research* 119, 285–301 (2003)
8. Moz, M., Pato, M.: Solving the problem of re-rostering nurse schedules with hard constraints: New multicommodity flow models. *Annals of Operations Research* 128, 179–197 (2004)
9. Moz, M., Pato, M.: A genetic algorithm approach to a nurse re-rostering problem. *Computers & Operations Research* 34, 667–691 (2007)
10. Pato, M., Moz, M.: Solving a bi-objective nurse re-rostering problem by using a utopic Pareto genetic heuristic. *Journal of Heuristics* 14, 359–374 (2008)
11. Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., Tavares, L.: An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research* 187, 511–524 (2008)
12. Vanhoucke, M., Maenhout, B.: On the characterization and generation of nurse scheduling problem instances. *European Journal of Operational Research* 196, 457–467 (2009)

Automatic Algorithm Selection for the Quadratic Assignment Problem Using Fitness Landscape Analysis

Erik Pitzer, Andreas Beham, and Michael Affenzeller

University of Applied Sciences Upper Austria
School of Informatics, Communications and Media
Softwarepark 11, 4232 Hagenberg, Austria
{erik.pitzer, andreas.beham, michael.affenzeller}@fh-hagenberg.at

Abstract. In the last few years, fitness landscape analysis has seen an increase in interest due to the availability of large problem collections and research groups focusing on the development of a wide array of different optimization algorithms for diverse tasks. Instead of being able to rely on a single trusted method that is tuned and tweaked to the application more and more, new problems are investigated, where little or no experience has been collected. In an attempt to provide a more general criterion for algorithm and parameter selection other than “it works better than something else we tried”, sophisticated problem analysis and classification schemes are employed. In this work, we combine several of these analysis methods and evaluate the suitability of fitness landscape analysis for the task of algorithm selection.

Keywords: Fitness Landscape Analysis, Problem Understanding, Quadratic Assignment Problem, Robust Taboo Search, Variable Neighborhood Search.

1 Introduction

While metaheuristic algorithms have been employed successfully in the past to solve many different problems belonging to different problem classes and different encodings, every new problem and every new problem class requires meticulous examination, algorithm selection and parameter tuning to ensure high quality results. Even with proper experience this can be a tedious task.

In the past, fitness landscape analysis methods have been developed to analyze the characteristic structure of problems that are typically solved with metaheuristic algorithms to mitigate these problems. Initially, many measures have been developed to directly correlate with problem hardness [1, 2]. However, simple and generally applicable fitness landscape measures do not have enough power to work efficiently in isolation to solve this case. Therefore, more measures were created to complement them [3].

In this work we try to simplify the process of algorithm selection by providing a method to quickly compare characteristic properties of problem instances

with each other. In a second step we use the underlying numeric data to create a feature vector composed of many fitness landscape characteristics that is then used to compare the problem at hand with other problems that have been subjected to optimization before. Together with previous optimization results several methods for algorithm selection have been tested.

2 Theoretical Foundations

2.1 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) was introduced in [4] and is a well-known problem in the field of operations research. As it is NP-hard, it is often solved with the help of metaheuristics such as tabu search [5, 6]. Informally, it can be described as finding the best assignment for a set of facilities to a set of locations so that each facility is assigned to exactly one location which in turn houses only this facility. An assignment is considered better than another when the flows between the assigned facilities have to be hauled over smaller distances.

More formally, the problem is given as an $N \times N$ matrix W with elements w_{ik} denoting the weights between facilities i and k and an $N \times N$ matrix D with elements d_{xy} denoting the distances between locations x and y . The goal is to find a permutation π with $\pi(i)$ denoting the element at position i to minimize the following equation:

$$\min \sum_{i=1}^N \sum_{k=1}^N w_{ik} \cdot d_{\pi(i)\pi(k)} \quad (1)$$

2.2 QAPLIB

The quadratic assignment problem library (QAPLIB)[7] is a collection of benchmark instances from different contributors and is freely accessible on the web¹. It includes the instance descriptions in a common format, as well as optimal and best-known solutions and consists of a total of 137 instances from 15 contributing sources which cover real-world as well as random instances. About 117 instances have been selected for this study with the exception of the `esc*` instances that sometimes do not have any flows and most of them are solved in only a few iterations.

2.3 Robust Taboo Search

Tabu search (TS) is a general metaheuristic that was proposed by Glover in [8]. It behaves like a local search, except that it will always make a move in each iteration, even if the current fitness deteriorates. The name results from the use of a memory that forbids to make certain moves and forces the exploration of new parts of the fitness landscape.

¹ <http://www.seas.upenn.edu/qaplib/>

The Robust Taboo Search (RTS) was proposed by Taillard in [5] and the code has since been further developed². The main difference lies in the stochastic choice of tabu tenures. For every move, the time to keep it tabu is a random number drawn from a strongly left-skewed distribution and lowers the possibility of repeatedly returning to the same solution. Another important aspect of the RTS is its aspiration tenure to perform moves which have not been seen for a number of iterations and increase diversification of the search.

2.4 Variable Neighborhood Search

Another metaheuristic that has gained popularity over the years is the variable neighborhood search (VNS) which is described in [9]. The core of the algorithm is similar to tabu search in that it uses a simple local search algorithm to find good solutions. However, the diversification is different. VNS introduces not only one, but several neighborhoods which it uses to diversify the current solution.

When solving the QAP, we configured the VNS to use the swap neighborhood for local search and for making the first small diversification. In total we used seven different neighborhoods that perturbed the permutation and allowed the algorithm to escape local optima. The neighborhood functions are implemented in the open source optimization environment HeuristicLab³ and are called: Swap2, Swap3, Scramble, Inversion, Insertion, Translocation, and TranslocationInversion. While these are not detailed in this work, they can be found in the source at the `Encodings.Permutation` name space. The impact of the changes made in these operators increases with the neighborhood index.

The difference between RTS's and VNS's diversification mechanisms leads to situations where one algorithm is able to outperform the other one on certain problem instances.

2.5 Fitness Landscape Analysis Fundamentals

The purpose of fitness landscape analysis is to gain more insight into the internal structure of a problem instance. Its methodology and power is similar to that of metaheuristic optimization processes, however, with a different aim. While metaheuristic optimization tries—by definition—to find a good solution in reasonable time, fitness landscape analysis (FLA) methods try to “find” good insights into the given problem's structure in reasonable time.

Typically, problem dependent measures have been developed to estimate problem hardness (e.g. flow dominance for the quadratic assignment problem [10]). Even earlier, generally applicable methods have been developed that tried to relate simple measures to problem hardness [2]. We are following a slightly different approach in this work: Instead of trying to establish a relationship between fitness landscape analysis measures (FLA measures) and problem hardness we

² <http://mistic.heig-vd.ch/taillard/>

³ <http://dev.heuristiclab.com/>

are taking the analysis results simply as a concise description of important landscape properties and use it to compare problems with each other. We postulate that similar problems (in terms of fitness landscape analysis) will have similar properties when subjected to optimization.

The formal definition of a fitness landscape includes not only the set of all possible solution candidates \mathcal{C} and the fitness function $f : \mathcal{C} \rightarrow \mathbb{R}$ but also a notion of connectedness \mathcal{X} . This connectedness can usually be defined by a distances between solution candidates $d : \mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}$. Then, we can define the *landscape* as the triple shown in Equation 2.

$$\mathcal{F} := (\mathcal{C}, f, d) \quad (2)$$

One of the first generally applicable measures for fitness landscapes, auto correlation, is described in [11]. The base for deriving the described ruggedness measures is to look at adjacent fitness values obtained by different trajectories as explained in Section 2.5. From this series of fitness values the auto correlation function $R(\tau)$ is calculated (see Equation 3) where \bar{f} and σ_f^2 are the mean and variance of the series of fitness values and $E[x]$ is the expected value of x . This function shows the correlation between the series of fitness values and itself, shifted by a certain number of steps. The first measure is the autocorrelation at a shift of one step. This shows how much a single step in the landscape modifies the fitness value on average. This value is often just called *the* autocorrelation but for completeness will be referred to as ‘autocorrelation 1’ throughout this paper. Another related measure is the *correlation length* which is the number of shift steps until the correlation is no longer statistically significant (i.e. is less in magnitude than $2/\sqrt{n}$, where n is the length of the analyzed trajectory).

$$R(\tau) := \frac{E[(f_i - \bar{f})(f_{i+\tau} - \bar{f})]}{\sigma_f^2} \quad (3)$$

Another interesting derivative from the series of fitness values obtained by various trajectories are several entropic information measures [12]. The underlying idea is to analyze fitness slope shapes between consecutive steps which are obtained by applying a relaxed sign function that gives either 1, -1 or 0 if the initial value is above, below or inside an ε -band around zero. Derived from these *symbols*, the frequency of consecutive *slopes* is now summarized in several entropy values. The first of which is the *information content* shown in Equation 4 where $P_{[pq]}$ is the frequency of consecutive slopes.

$$H(\varepsilon) := - \sum_{p \neq q} P_{[pq]} P_{[pq]} \quad (4)$$

While the information content calculates the entropy of consecutive different slopes, the *density basin information* does the same for consecutive equal slopes. These two measures can be perceived as the “interestingness of rugged and smooth areas”. Additionally, the *partial information content* which is simply the number of slope direction changes in the sequence and the *information stability* which is the maximum difference between consecutive fitness values are

analyzed. By varying the parameter ε one can focus on larger or smaller fitness differences and hence “zoom in and out” of the fitness landscape.

A very important characteristic of fitness landscapes is neutrality. It describes the parts of the landscape with adjacent solution candidates with (almost) equal fitness. This is especially relevant for trajectory-based optimization strategies that have no mechanism for intelligently escaping from a plateau or population-based algorithms that can take advantage of the greater potential boundary surface of the neutral area. One way to measure neutrality is to estimate the number and sizes of neutral areas by performing neutral walks as explained in Section 2.5.

A straightforward approach is to count the average number of steps that can be made without a fitness change in comparison to the total number of steps.

Usually the size of the problem determines the solution space size and, therefore, is *the* major driving factor in increasing problem complexity and hence problem hardness. Many fitness landscape measures are, unsurprisingly, strongly correlated or even dominated by problem size. This means, that the extent of a measurement is predominantly determined by the problem size. As the problem size itself is one of the factors considered in the further investigations anyway, we have tried to filter out the effects of the problem size on the fitness landscape measures. Especially auto correlation and correlation length have been normalized by dividing through the problem size. Moreover, the auto correlation coefficient $\xi = 1/(1 - r(1))$ (where $r(1)$ is the auto correlation at step 1) has been included and normalized as otherwise it correlates perfectly with problem size itself.

Trajectories. As stated earlier, fitness landscape analysis can be compared to metaheuristics on a technical level. One of the basic tools for exploring a problem’s landscape are sampling techniques. The most popular trajectory used for fitness landscape analysis is the *random walk*. In this case, a random neighbor is generated by applying a manipulation operator to a random starting point and following along. This trajectory exhibits very high diversity of the examined solution candidates while still elucidating the neighborhood’s properties.

Another trajectory is the *adaptive walk* that resembles an optimization process. Here from a sample’s neighbors the best is chosen to reach higher fitness levels and hence emphasize the exploration of higher quality solution candidates which are more interesting for typical optimization algorithms. An important variation, that was used in this work, is the *up-down walk*, where fitness is not improved indefinitely but stops at a local optimum and reverses its direction. This resembles a climber that climbs up a hill, then down again then up again, and so on. This strategy emphasizes the exploratory nature of fitness landscape analysis that is not directly interested in finding good solutions. We have also collected some additional numbers during up-down walks as described in Section 3.1.

Finally, areas with similar or equal fitness in the solution space can also be explored with a so-called *neutral walk*. Here the strategy is to continue onwards at the same fitness level to explore plateaus and saddle areas. Usually a larger

sample of neighbors has to be used to ensure that neighboring solution candidates with equal fitness are found. While producing neutral walks we ensure that the distance to the starting point is continuously increased to prevent “getting lost” inside the flat neutral area as strongly bent or wound neutral areas could exhibit large differences between distances and number of steps. The actual measurements taken are listed in Section 3.1.

3 Methodology

3.1 Fitness Landscape Investigations

After the initial optimization experiments we have hand picked several instances of the QAPLIB and performed a thorough analysis which can be found online⁴. We have then created several experiments for every instance of the QAPLIB with a best know solution based on the analysis results of these few instances.

We have only measured values induced by the ‘swap-2’ neighborhood, which is one of the smoothest landscape variants for the QAP. Moreover, for all three walk types we measured auto correlation, correlation length, information content, partial information content, density basin information, information stability and regularity. In addition, for up-down walks we measured average lengths and variances of the up and down portions as well as average level and variance of the top and bottom turning points. Similarly we measured average lengths and variances of the flat portions of neutral walks. Moreover, we also measured the distances between entry and exit points of the neutral areas. In summary, we derived 34 different fitness landscape characteristics including the problem size itself. In Table 1a we have summarized the runtime of the analysis algorithms as well as the run times of optimization trials in Table 1b using two different algorithms with hand-tuned parameter settings. All of these analysis experiments were performed using HeuristicLab⁵ [13] with custom plug-ins for fitness landscape analysis. The runtimes are still rather high in comparison to algorithm execution, but we did not yet optimize the performance of the feature extraction phase. Faster methods are available in parts, e.g. [14] shows how to calculate the autocorrelation coefficient much quicker and without sampling.

3.2 Structural Problem Comparison

The underlying idea is to select a set of n fitness landscape analysis values and compare the resulting vectors for different problem instances. The resulting *Euclidian distance* in features space can then be used as a measure of problem similarity and helps select appropriate methods and algorithm parameters for optimization.

In our case we have used as many complementary fitness landscape measures as possible initially. Moreover, we have tested different subsets with different

⁴ <http://dev.heuristiclab.com/AdditionalMaterial>

⁵ <http://dev.heuristiclab.com>

Table 1. Run Times and Run Repetitions: While the optimization algorithms had to be repeated several times for each problem, the fitness landscape analysis needs to be performed just a single time for each instance

(a) Fitness Landscape Analysis					(b) Optimization Algorithms				
Trajectory	Runtime		Runs	Evaluated Solutions	Alg	Runtime		Runs	Evaluated Solutions
	Avg	Total				Avg	Total		
Random	0:59	1:55:03	117	100,000	RTS	1:22	79:45:50	3510	1,000,000
Up/Down	3:10	6:60:30	117	1,000,000	VNS	1:15	72:53:58	3510	1,000,000
Neutral	2:20	4:33:00	117	100,000					

variable selection techniques. A first filtering step was variable selection by correlation. All variables with significant correlation to other variables were then excluded at three different levels: 0.9, 0.99 and 1. Next these values are all standardized to equalize the influences each variable has on the overall distance.

The first successful use case for the obtained distance values was a clustering of problem instances. Figure 1 shows the results of multi dimensional downscaling of the obtained distances. The map shows that some instances come from sources that form their own clusters while other clusters consist of instances from multiple sources.

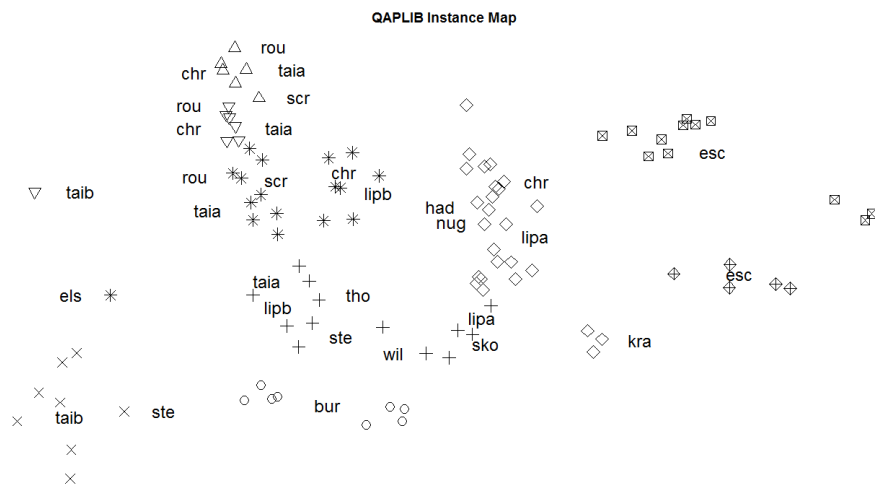


Fig. 1. Multi-Dimensional Downscaling and Projection of Fitness Landscape Distances: The instances were clustered using k-means ($k = 9$) and projected using Neighborhood Components Analysis [15]

3.3 Algorithm Selection: Dominance

As a first attempt towards algorithm selection we have reformulated a simple binary decision problem. For the quadratic assignment problem two algorithms have proved particularly successful, namely the robust taboo search and the variable neighborhood search as introduced in Sections 2.3 and 2.4.

Following several previous studies [16–18] of this dataset we have concluded that average parameter settings can provide a decent baseline and we have used only a single parameter setting for each algorithm.

On the other hand, we have taken a step back from parameter selection or hardness prediction and gone to a more general problem of algorithm selection. To come up with a good overview of algorithm performance two aspects have to be considered: On the one hand, algorithm convergence speed and, on the other hand, the final achieved quality. While one algorithm might arrive at a high quality solution very quickly, another one could, after some more time, come up with a significantly superior solution. Therefore, it is not straightforward to decide on one algorithm in general.

For this purpose, we have created repeated snapshots of the algorithm progress after approximately the following numbers of evaluations: $\{1, 2.5, 5, 7.5\} \cdot 10^6$, $i \in \{1, \dots, 6\}$. For VNS these numbers are only approximate since we did not measure performance inside the inner local search. For this reason, VNS seems to start out much better than RTS. Table 1b summarizes the run times of 30 repetitions over 117 different problem instances for the two algorithm configurations. As can be seen, not only the number of evaluations but also the overall run time of the two algorithms are quite comparable. Therefore, for this particular problem class it is a difficult problem to predict which algorithm will be better, as both are equally good on average.

For all algorithms and snapshots we calculated the mean and standard deviation over 30 runs as shown in Figure 2. While for some problem instance, a significant domination of one algorithm over another can be seen (Figures 2a and 2b), in other cases the domination depends on the allowed computational effort: Figures 2c and 2d, where at first one algorithm dominates, but the other one converges earlier.

To come up with a reasonable distinction between algorithms we have analyzed the relative performance over time and performed a significance analysis using a t-test to determine whether an algorithm is significantly better at a certain snapshot, which we call *snapshot dominance*. It became clear that a better alternative would be to include the convergence history as well, especially in cases where no algorithm was significantly better at a certain time. In these cases, the previously dominating algorithm would be selected as still superior as it arrived and stayed at a competitive level first. Only after the other algorithm becomes significantly better again, the dominances are exchanged. An important exception was finding the global optimum or best known solution which was always counted as significantly better if the other algorithm had not found it yet.

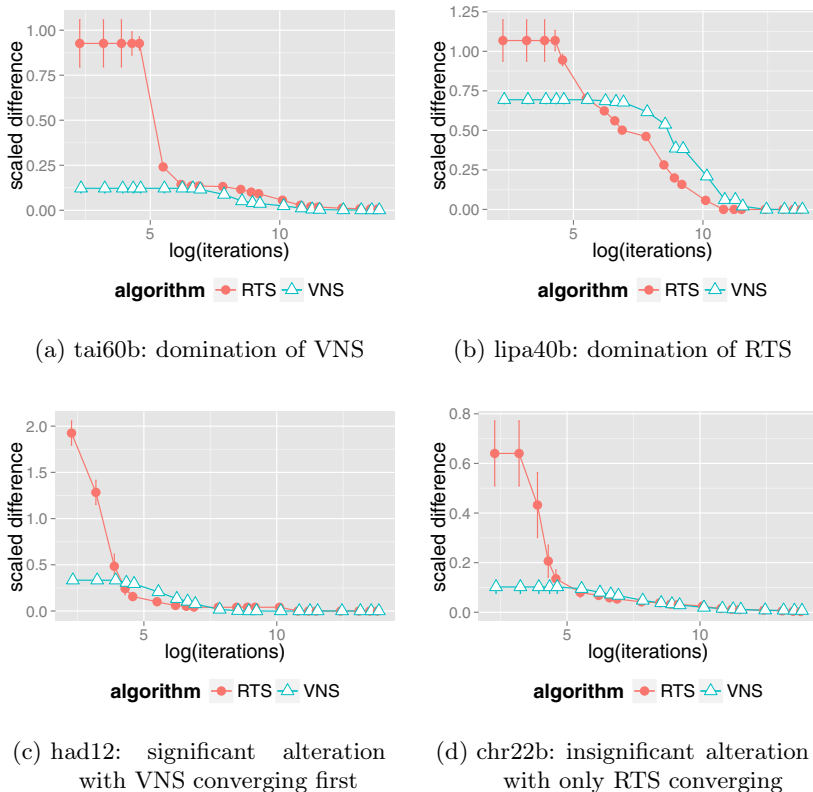


Fig. 2. Convergence Function Families - the y axis shows the quality linearly scaled between the best-known quality value and an instance-specific average

We took these series of dominances and performed a run-length encoding yielding series of alternating algorithms which are either currently significantly better or are now competitive and have been dominating before. We have performed this analysis at different significance levels. Our final choice was to only look at very significant performance differences

Table 2 shows some (extreme) examples of different significance levels. VNS comes out first every time, since the first few snapshots (at logarithmic) scale reflect the first measured value that was taken much later. However, when the significance is increased enough, we end up with three possibilities, either RTS or VNS dominates in the end or there was a significant alteration.

4 Algorithm Dominance Prediction

As the fitness landscape analysis measures show a meaningful separation between problem generators we tested whether they could also be used for predicting

Table 2. Dominance Series Compression: Increasing the required significance level only a few scenarios remain

instance	significance level			instance	significance level		
	0.9	0.99	0.9999		0.9	0.99	0.9999
bur26a	VR	VR	VR	bur26b	V	V	V
kra30b	VRVR	VR	VR	chr22b	VRVR	VR	V
lipa90b	VRV	VRV	VR	lipa70a	VRVRV	VRV	V
lipa80a	VRV	VRV	VRV	had12	VRV	VRV	V

algorithm choice. For this purpose, we have prepared several datasets consisting of different variable selections of FLA values together with different variants of class assignments of the previous section: We used three different thresholds for variable cross-correlation elimination: 1, 0.99 and 0.9, where all variables that a simple Pearson’s and a Spearman’s rank correlation higher than these values were dropped. With the first threshold, redundant values such as information stability, partial information content which are repeated for all walks and one direction of up-down walk statistics were removed. In the 90% correlation case, many ruggedness related characteristics of the up-down walks were removed as they always characterize smooth adaptive walks. The data sets had 34, 30 and 18 variables and can be examined at material⁶.

Three different types of classification were performed: A direct, three class classification, yielding either one of the algorithms or undecided, a two-class classification, where we removed the four undecided instances, as for these cases the choice would not matter anyway, and a re-sampled binary classification problem to ensure equal numbers of class members.

These nine datasets were tried using 4-fold cross validation for all forty applicable classification algorithms within Weka 3.6 [19] using only default settings. The aim of this work is not to suggest a certain classification scheme but to evaluate the overall viability of fitness landscape analysis as predictor of algorithm selection.

5 Results

Table 3 shows the results of algorithm selection using different classifiers. While all results were quite promising we have selected those that showed another significant improvement over a simple rule learner with one variable (OneR). The best results were achieved using a variant of support vector machines using sequential minimal optimization [20].

The results show quite satisfactory predictive power, being able to correctly select the dominating algorithm. While, the binary problem is, naturally, slightly easier to predict as the most difficult, but also most unimportant instance have been removed. With an average percentage of 80% correctly classified instances

⁶ <http://dev.heuristiclab.com/AdditionalMaterial>

Table 3. Top Classifier Results: Percent of correctly classified problem instances for one variable rule learner (OneR), sequential minimal optimization (SMO) of a support vector machine and Gaussian processes (GProc), and linear regression (LR)

r	selection	SMO	LR	GProc	OneR
1.00	all	80.57	77.55	76.61	71.11
0.99	all	80.40	77.55	77.15	71.08
0.90	all	78.60	78.72	78.77	68.97
1.00	binary	84.33	81.30	79.18	73.87
0.99	binary	84.21	80.27	79.95	73.81
0.90	binary	82.74	82.56	81.13	71.90
1.00	re-sampled	81.96	79.00	78.37	73.07
0.99	re-sampled	82.44	78.41	79.11	72.85
0.90	re-sampled	81.36	79.42	78.78	70.81
	average	81.85	79.42	78.78	71.94

and a maximum of $\tilde{2}$ % difference between the folds, this scheme should prove as a real help in the selection of an appropriate algorithm.

6 Conclusion

In the exploration of fitness landscape analysis, its general applicability and the suitability for algorithm and parameter selection, we have picked the topic of a binary selection problem between two exceptionally performing algorithms. In this case, it would be difficult, even for experienced researchers, to make an up-front decision between any of the two. Using a combination of several fitness landscape analysis methods, a grid of tests with different problem instances and simple convergence speed analysis of robust taboo search and variable neighborhood search, we have shown that the insights obtained by fitness landscape analysis can provide valuable help for this decision.

In the future we plan to evaluate other schemes of inference to come up with more methods to help decide which algorithms to choose and even which parameters to select for different problem instance. As an ultimate goal, we want to be able to make these predictions even for problem classes for which hardly any experience is available by using the possibility of comparing fitness landscape analysis results of different problem classes. In a certain way, we are trying get around the free lunch theorem [21] and snatch a free appetizer by providing a reasonable first choice of algorithm and parameters.

It is however necessary to admit that we did not yet extend these results to further problems and algorithms. Also, it should be clear that presently we compared two algorithms in two specific parameterizations that were chosen to be suitable in preceding studies. The inclusion of algorithm parameters in the comparison is also an open issue for future work.

References

1. Weinberger, E.D.: Local properties of kauffman's n-k model, a tuneably rugged energy landscape. *Physical Review A* 44(10), 6399–6413 (1991)
2. Jones, T.: *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, New Mexico (1995)
3. Pitzer, E., Affenzeller, M.: A Comprehensive Survey on Fitness Landscape Analysis. In: Fodor, J., Klempous, R., Suárez Araujo, C.P. (eds.) *Recent Advances in Intelligent Engineering Systems*. SCI, vol. 378, pp. 161–191. Springer, Heidelberg (2012)
4. Koopmans, T.C., Beckmann, M.: Assignment problems and the location of economic activities. *Econometrica* 25(1), 53–76 (1957)
5. Taillard, E.D.: Robust taboo search for the quadratic assignment problem. *Parallel Computing* 17, 443–455 (1991)
6. James, T., Rego, C., Glover, F.: Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 39(3), 579–596 (2009)
7. Burkard, R.E., Karisch, S.E., Rendl, F.: QAPLIB - A quadratic assignment problem library. *Journal of Global Optimization* 10(4), 391–403 (1997)
8. Glover, F.: Tabu search – part I. *ORSA Journal on Computing* 1(3), 190–206 (1989)
9. Hansen, P., Mladenovic, N., Perez, J.: Variable neighbourhood search: methods and applications. *Annals of Operations Research* 175, 367–407 (2010)
10. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation* 4(4), 337–352 (2000)
11. Weinberger, E.: Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics* 63(5), 325–336 (1990)
12. Vassilev, V.K., Fogarty, T.C., Miller, J.F.: Information characteristics and the structure of landscapes. *Evol. Comput.* 8(1), 31–60 (2000)
13. Wagner, S.: *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Johannes Kepler University, Linz, Austria (2009)
14. Chicano, J.F., Luque, G., Alba, E.: Autocorrelation measures for the quadratic assignment problem. *Applied Mathematics Letters* 25(4), 698–705 (2012)
15. Goldberger, J., Roweis, S., Hinton, G., Salakhutdinov, R.: Neighbourhood components analysis. In: Saul, L.K., Weiss, Y., Bottou, L. (eds.) *Advances in Neural Information Processing Systems* 17, pp. 513–520. MIT Press, Cambridge (2005)
16. Pitzer, E., Vonolfen, S., Beham, A., Affenzeller, M., Bolshakov, V., Merkurjeva, G.: Structural analysis of vehicle routing problems using general fitness landscape analysis and problem specific measures. In: *14th International Asia Pacific Conference on Computer Aided System Theory*, pp. 36–38 (2012)
17. Pitzer, E., Beham, A., Affenzeller, M.: Generic hardness estimation using fitness and parameter landscapes applied to robust taboo search and the quadratic assignment problem. In: *GECCO 2012 Companion*, pp. 393–400 (2012)
18. Pitzer, E., Beham, A., Affenzeller, M.: Correlation of Problem Hardness and Fitness Landscapes in the Quadratic Assignment Problem. In: *Advanced Method and Applications in Computational Intelligence*, pp. 163–192. Springer (in press, 2013)
19. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. *SIGKDD Explorations* 11(1), 10–18 (2009)
20. Platt, J.C.: Fast Training of Support Vector Machines using Sequential Minimal Optimization. In: *Advances in Kernel Methods: Support Vector Learning*, pp. 185–208. MIT Press (1998)
21. Macready, W.G., Wolpert, D.H.: What makes an optimization problem hard? *Complexity* 5, 40–46 (1996)

Balancing Bicycle Sharing Systems: A Variable Neighborhood Search Approach*

Marian Rainer-Harbach, Petrina Papazek, Bin Hu, and Günther R. Raidl**

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{rainer-harbach,papazek,hu,raidl}@ads.tuwien.ac.at

Abstract. We consider the necessary redistribution of bicycles in public bicycle sharing systems in order to avoid rental stations to run empty or entirely full. For this purpose we propose a general Variable Neighborhood Search (VNS) with an embedded Variable Neighborhood Descent (VND) that exploits a series of neighborhood structures. While this metaheuristic generates candidate routes for vehicles to visit unbalanced rental stations, the numbers of bikes to be loaded or unloaded at each stop are efficiently derived by one of three alternative methods based on a greedy heuristic, a maximum flow calculation, and linear programming, respectively. Tests are performed on instances derived from real-world data and indicate that the VNS based on a greedy heuristic represents the best compromise for practice. In general the VNS yields good solutions and scales much better to larger instances than two mixed integer programming approaches.

1 Introduction

A large number of public Bicycle Sharing Systems (BSSs) has been introduced in many cities around the world in the last decade. Such systems augment public transport well and frequently present attractive “green” alternatives to individual motorized traffic. A BSS consists of a number of stations where users can rent and return bikes in an automated way. Operators face an important challenge with regard to customer satisfaction: Due to different factors such as the topographical height, the numbers of bikes rented and returned, respectively, differ significantly among the stations. Running such a system without any maintenance would therefore soon result in many completely empty or, equally worse, completely full stations. Thus the operator needs to actively rebalance the system by moving bicycles between stations with a fleet of vehicles, e.g. cars with trailers. In the *Balancing Bicycle Sharing System* (BBSS) problem we aim at finding efficient vehicle routes with corresponding bicycle-loading instructions at the visited stations in order to bring the system in balance as far as possible.

In this work we address this problem by a Variable Neighborhood Search (VNS) with an embedded Variable Neighborhood Descent (VND), which exploit various specifically designed neighborhood structures. While tours are searched within the VNS/VND,

* This work is supported by the Austrian Research Promotion Agency (FFG), contract 831740.

** The authors thank Matthias Prandtstetter, Andrea Rendl and Markus Straub from the Austrian Institute of Technology (AIT) for the collaboration in this project.

corresponding loading instructions are efficiently derived by either a greedy approach, a maximum flow calculation, or linear programming. Experiments are performed on benchmark instances derived from Citybike, the BSS in Vienna, Austria. They indicate that high quality solutions can be found with this approach and that the maximum flow based calculating of loading instruction performs best in practice.

2 The Balancing Bicycle Sharing System Problem

In this section we formalize the BBSS problem. Currently, we only consider a static problem variant in which user activities during the rebalancing process are neglected. The BSS is represented by a complete directed graph $G_0 = (V_0, A_0)$. Node set $V_0 = V \cup O$ consists of nodes for the rental stations V as well as start and end points O for vehicles (garages or overnight parking places). Each arc $(u, v) \in A_0$ has associated a travel time $t_{u,v} > 0$ that includes an expected time for parking near v and loading/unloading bikes. Let the subgraph induced by the bike stations V only be $G = (V, A)$, $A \subset A_0$.

Each station $v \in V$ has associated a capacity $C_v \geq 0$, i.e., the number of available parking positions, the number of available bikes at the beginning of the rebalancing process $p_v \geq 0$, and a target number of available bikes after rebalancing $q_v \geq 0$. A fleet of vehicles $L = \{1, \dots, |L|\}$ is available for transporting bikes. Each vehicle $l \in L$ has a capacity of $Z_l > 0$ bikes, a total time budget \hat{t}_l within which it has to finish a route (i.e., the worker's shift length), as well as specific start and destination nodes $s_l, d_l \in O$, respectively. We assume that all vehicles start and finish rebalancing empty. A solution consists of two parts. The first part is the route for each vehicle $l \in L$ specified by an ordered sequence of visited stations $r_l = (r_l^1, \dots, r_l^{\rho_l})$ with $r_l^i \in V$, $i = 1, \dots, \rho_l$ and ρ_l representing the number of stops. Note that stations may be visited multiple times by the same or different vehicles. Start and end points s_l and d_l are fixed for each vehicle and are prepended and appended, respectively, to each route in order to obtain complete tours. The second part consists of loading and unloading instructions $y_{l,v}^{+,i}, y_{l,v}^{-,i} \in \{0, \dots, Z_l\}$ with $l \in L$, $v \in V$, and $i = 1, \dots, \rho_l$, specifying how many bikes are picked up or delivered, respectively, at station v at the i -th stop of vehicle l . Note that $\forall v \neq r_l^i : y_{l,v}^{+,i} = y_{l,v}^{-,i} = 0$.

The following conditions must hold in a feasible solution: The number of bikes available at each station $v \in V$ needs to be within $\{0, \dots, C_v\}$. For any vehicle $l \in L$ capacities Z_l may never be exceeded, and the tour time t_l

$$t_l = \begin{cases} t_{s_l, r_l^1} + \sum_{i=2}^{\rho_l} t_{r_l^{i-1}, r_l^i} + t_{r_l^{\rho_l}, d_l} & \text{for } \rho_l > 0 \\ t_{s_l, d_l} & \text{for } \rho_l = 0, \end{cases} \quad (1)$$

is restricted by the time budget \hat{t}_l , $\forall l \in L$. Let a_v be the final number of bikes after rebalancing at each station $v \in V$,

$$a_v = p_v + \sum_{l \in L} \sum_{i=1}^{\rho_l} (y_{l,v}^{-,i} - y_{l,v}^{+,i}). \quad (2)$$

The objective is to find a feasible solution that primarily minimizes the deviation from the target number of bikes $\delta_v = |a_v - q_v|$ at each station $v \in V$ and secondarily the number of loading/unloading activities plus the overall time required for all routes, i.e.,

$$\min \alpha^{\text{bal}} \sum_{v \in V} \delta_v + \alpha^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} \left(y_{l,r_l^+}^{+,i} + y_{l,r_l^-}^{-,i} \right) + \alpha^{\text{work}} \sum_{l \in L} t_l, \quad (3)$$

where $\alpha^{\text{bal}}, \alpha^{\text{load}}, \alpha^{\text{work}} \geq 0$ are scaling factors controlling the relative importance of the respective terms.

A simplification that may be exploited in different approaches is to consider *monotonicity* regarding fill levels of stations. Let $V_{\text{pic}} = \{v \in V \mid p_v \geq q_v\}$ denote pickup stations and $V_{\text{del}} = \{v \in V \mid p_v < q_v\}$ denote delivery stations. A vehicle is only allowed to load bicycles at pickup stations and unload them at delivery stations. In this way the number of bikes decreases or increases monotonically, therefore the order in which different vehicles visit a single station does not matter. On the downside, forcing monotonicity might exclude some better solutions that would have been feasible without this restriction.

3 Related Work

It was not until recently that the BBSS problem has been recognized as a combinatorial optimization problem and the operations research community described a few systematic solution approaches. However, they address significantly different problem variants and a direct comparison between existing approaches is difficult. The majority of existing works uses mixed integer programming (MIP), which in principle is able to find proven optimal solutions but in practice is restricted to very small instances.

Chemla et al. [1] address the static case with only one vehicle and achieving perfect balance as a hard constraint. They describe a branch-and-cut approach utilizing an embedded tabu search for locally improving incumbent solutions. To the best of our knowledge, their tabu search is the only metaheuristic approach applied to the rebalancing problem until now. One of the key concepts is to only consider the visiting order of the rebalancing vehicle in the solution representation and to obtain the loading instructions by an auxiliary algorithm based on a maximum flow computation. With this technique the search space can be reduced significantly. In our work we extend this idea towards our more general problem definition. Raviv et al. [2] propose four MIP models. In their objective function they model user dissatisfaction and tour lengths but ignore the number of loading operations. The models were tested on real-world data obtained from Vélib (Paris) with up to 60 stations. Results show that the most basic arc indexed model produces the best lower bounds in a given time limit, but more complex models offer more flexibility with respect to the requirements. Benchimol et al. [3] again assume balancing as hard constraint, only consider the total tour length as objective, and focus on approximation algorithms for selected special situations. Finally, Contardo et al. [4] investigate the more complex dynamic scenario where rebalancing is done while the bike sharing system is in use. They propose an arc-flow formulation and a pattern-based formulation for a space-time network model. The latter is solved heuristically by a hybrid approach using column generation and Benders decomposition. On randomly created instances, this approach was able to handle instances with up to 100 stations and 60 time periods, however significant gaps between lower and upper bounds still remain.

There are other works in the literature which focus on the strategic planning aspects of bike sharing systems (i.e., location and network design depending on demands). However, these aspects are not within the scope of this work. More generally, BBSS is closely related to diverse variants of the classical vehicle routing problem (VRP). However, it differs in substantial ways: Most importantly, stations may be visited multiple times, even by different vehicles. Consequently, BBSS can be described as a capacitated single commodity split pickup and delivery VRP.

4 Greedy Construction Heuristic

To efficiently generate a meaningful initial solution, we employ a construction heuristic based on greedy principles. This procedure assumes monotonicity as described in Section 2. A solution is built by iteratively creating a tour for each vehicle following a local best successor strategy. From the last station of a partial tour, we first determine the *set* $F \subseteq V$ of *feasible successor stations*. These are all stations that are not yet balanced and can be reached without exceeding the shift length, i.e., there is enough time left to visit the station and to go to the destination node afterwards.

For each such candidate station $v \in F$, we calculate the maximum amount of bicycles that can be picked up or delivered by

$$\gamma_v = \begin{cases} \min(a_v - q_v, Z_l - b_l) & \text{for } v \in F \cap V_{\text{pic}}, \\ \min(q_v - a_v, b_l) & \text{for } v \in F \cap V_{\text{del}}, \end{cases} \quad (4)$$

where b_l represents the final load of vehicle l and a_v the final number of bikes at station v in the currently considered partial tour. For $\rho_l = 0$ they are initialized with $b_l = 0$ and $a_v = p_v$.

We assume that no bikes are allowed to remain on a vehicle when returning to the depot. Therefore, an additional correction is necessary for pickup stations: We determine for each $v \in F \cap V_{\text{pic}}$ if after visiting v the remaining time budget allows the vehicle to deliver at least $b_l + 1$ bicycles to other stations, i.e., all bikes the vehicle currently has loaded plus at least one that would be picked up from v . If this is not the case, visiting v is useless as no bike may finally be picked up there.

For this purpose, we estimate the number of deliverable bikes b_v^{del} after visiting v by iteratively applying the exact same greedy heuristic restricted to delivery stations only. We stop extending this delivery-only route when either $b_v^{\text{del}} \geq b_l + \gamma_v$ (i.e., we have shown that all bicycles picked up at v can be delivered later) or the time budget \hat{t}_l is exceeded. Then, pickup stations v with $b_v^{\text{del}} < b_l + 1$ are removed from set F , while the number of bikes to be picked up is possibly reduced for the others:

$$\gamma_v \leftarrow \min(\gamma_v, b_v^{\text{del}} - b_l), \quad \forall v \in F \cap V_{\text{pic}}. \quad (5)$$

Now, all candidate stations $v \in F$ are evaluated using the ratio $\gamma_v/t_{u,v}$, where $t_{u,v}$ is the traveling time from the vehicle's last location u to station v ; thus we consider the balance gain per time unit. The node $v \in F$ with the highest ratio is then appended to the tour of vehicle l ; ties are broken randomly. Loading instructions are set as follows:

$$y_{l,v}^{+,\rho_l} = \gamma \text{ and } y_{l,v}^{-,\rho_l} = 0 \quad \text{if } v \in V_{\text{pic}}, \quad (6)$$

$$y_{l,v}^{+,\rho_l} = 0 \text{ and } y_{l,v}^{-,\rho_l} = \gamma \quad \text{if } v \in V_{\text{del}}. \quad (7)$$

Next, b_l and a_v are updated accordingly and the procedure continues with the next extension, evaluating stations in F from scratch, until no feasible extension remains.

5 Variable Neighborhood Search

In this section we describe our VNS approach. It uses the general VNS scheme with an embedded VND for local improvement as described in [5].

5.1 Solution Representation and Derivation of Loading Instructions

Concerning the VNS we use an incomplete solution representation by storing for each vehicle $l \in L$ its route $r_l = (r_l^1, \dots, r_l^{\rho_l})$ only. Corresponding loading instructions $y_{l,v}^{+,i}, y_{l,v}^{-,i}, l \in L, v \in V, i = 1, \dots, \rho_l$ are derived for each created set of tours by one of the following, alternative procedures, which have different assets and drawbacks.

Greedy Heuristic (GH): This simplest and fastest approach follows the pure greedy strategy from the construction heuristic and assumes monotonicity. For each tour, the stations are considered in the order as they are visited and loading instructions are computed as described in Section 4. Even under the restriction of monotonicity, GH is not guaranteed to find optimal loading instructions. For example, it can be beneficial to retain bikes in the vehicle at a first stop at some station v in order to satisfy a following delivery station as v will be visited a second time and can also be satisfied then.

Maximum Flow Approach (MF): When assuming monotonicity, we are able to derive optimal loading instructions via an efficient maximum flow computation on a specifically defined flow network. This approach is inspired by [1] but extends their method towards multiple vehicles and the consideration of balance in the objective function. We define graph $G_{\text{fm}} = (V_{\text{fm}}, A_{\text{fm}})$ with node set $V_{\text{fm}} = \{\sigma, \tau\} \cup V_{\text{pic}} \cup V_{\text{del}} \cup V_L$ where σ and τ are the source and target nodes of the flow, respectively, and $V_L = \bigcup_{l \in L} V_l$ with $V_l = \{v_l^i \mid l \in L, i = 1 \dots, \rho_l\}$ represents the stops of all routes.

Arc set $A_{\text{fm}} = A_\sigma \cup A_L \cup A_{\text{pic}} \cup A_{\text{del}} \cup A_\tau$ consists of:

- $A_\sigma = \{(\sigma, v) \mid v \in V_{\text{pic}}\}$ with capacities $p_v - q_v$.
- $A_\tau = \{(v, \tau) \mid v \in V_{\text{del}}\}$ with capacities $q_v - p_v$.
- $A_{\text{pic}} = \{(v, v_l^i) \mid v_l^i \in V_L, v = r_l^i, v \in V_{\text{pic}}\}$, i.e., each pickup node in V_{pic} is connected with every node representing a stop at this station in any route $l \in L$. These arcs' capacities are not limited.
- $A_{\text{del}} = \{(v_l^i, v) \mid v_l^i \in V_L, v = r_l^i, v \in V_{\text{del}}\}$, i.e., each node representing a stop at a delivery station is connected to the corresponding delivery node in V_{del} . These arcs' capacities are also not limited.
- $A_L = \{(v_l^{i-1}, v_l^i) \mid v_l^i \in V_L, i > 1\}$, i.e., the nodes representing the stops in each tour are connected according to the tour. Arc capacities are Z_l .

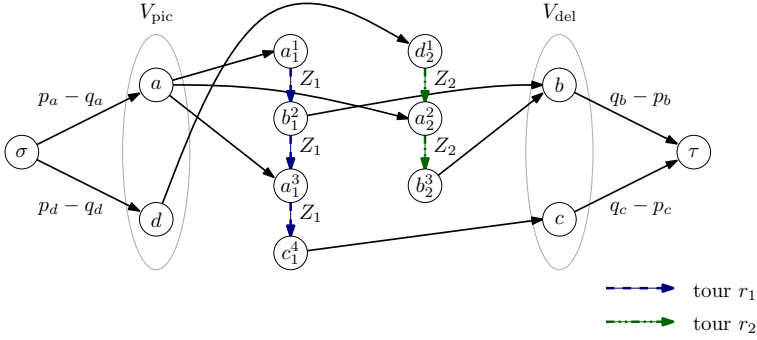


Fig. 1. Exemplary flow network when considering monotonicity for the tours $r_1 = (a, b, a, c)$ and $r_2 = (d, a, b)$ with $V_{pic} = \{a, d\}$ and $V_{del} = \{b, c\}$

An exemplary network is shown in Figure 1. Calculating a maximum (σ, τ) -flow on it directly yields optimal loading instructions $y_{l,v}^{+,i}, y_{l,v}^{-,i}$ via the flows on the corresponding arcs A_{pic} and A_{del} , respectively. In our implementation, we used the efficient push-relabel method from Cherkassky and Goldberg [6] for the flow computation.

Linear Programming Approach (LP): Finally, we are able to determine optimal loading instructions even for the general, not necessarily monotonic case by solving a minimum cost flow problem on a differently defined network by linear programming. The main difference is that the order in which vehicles make their stops (at possibly the same stations) is considered. Bikes can be buffered at stations or even be directly transferred from one vehicle to another when they meet.

Let $t(r_l^i)$ denote the absolute time when vehicle l makes its i -th stop at station r_l^i . We define the multi-graph $G_f = (V_f, A_f)$ with node set $V_f = \{\sigma, \tau\} \cup V_t$ where $V_t = \{v^j \mid \exists v_l^i \in V_l : t(r_l^i) = j\}$, i.e., besides source and target nodes σ and τ we have a node v^j for each station v and time j when a vehicle arrives at v . Furthermore $V^{first} = \{v^{j_{min}} \in V_t \mid j_{min} = \min\{j \mid v^j \in V_t\}\}$, i.e., these nodes represent the first visits of all stations among all routes, and $V^{last} = \{v^{j_{max}} \in V_t \mid j_{max} = \max\{j \mid v^j \in V_t\}\}$, i.e., these nodes represent the last visits of all stations.

Arc set $A_f = A_\sigma \cup A_\tau \cup A_R \cup A_V$ consists of:

- $A_\sigma = \{(\sigma, v^j) \mid v^j \in V^{first}\}$ with capacities p_v .
- $A_\tau = \{(v^j, \tau) \mid v^j \in V^{last}\}$ with capacities q_v .
- $A_R = \bigcup_{l \in L} A_{R,l}$ with $A_{R,l} = \{(u^j, v^j) \mid u = r_l^{i-1}, v = r_l^i, j' = t(r_l^{i-1}), j = t(r_l^i), i = 2, \dots, \rho_l\}$, $\forall l \in L$, i.e., the arcs representing the flow induced by the vehicles. Capacities are Z_l . Note that multiple arcs exist between two nodes if two (or more) vehicles leave and arrive at the same stations exactly at the same times.
- $A_V = \bigcup_{v \in V} A_v$ with $A_v = \{(v^{j_1}, v^{j_2}), \dots, (v^{j_{max-1}}, v^{j_{max}})\}$, $(v^{j_1}, \dots, v^{j_{max}})$ is the sequence of nodes $\{v^j \in V_t\}$ sorted according to increasing j . Capacities are C_v .

An example of this network is given in Figure 2. Now, a simple maximum flow calculation would in general not yield optimal or even feasible loading instructions. Instead,

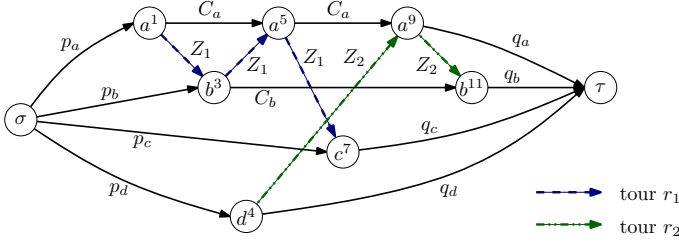


Fig. 2. Exemplary flow network for the general case with tours $r_1 = (a, b, a, c)$ and $r_2 = (d, a, b)$

we have to solve a minimum cost flow problem via the following LP, which uses flow variables $f_{u,v}$, $\forall (u, v) \in A_f$. By $pred_l(v^j) \in V_t$ we denote the predecessor of the node v^j on the route of vehicle l , i.e., $pred_l(v^j) = u^{j'}$ with $u = v_l^{j-1}$, $j' = t(r_l^{j-1})$, and by $succ_l(v^j) \in V_t$ the successor, i.e., $succ_l(v^j) = w^{j''}$ with $w = v_l^{j+1}$, $j'' = t(r_l^{j+1})$.

$$\min \alpha^{bal} \sum_{v \in V^{last}} \delta_v + \alpha^{load} \sum_{l \in L} \sum_{i=1}^{\rho_l} \left(y_{l,r_l^i}^{+,i} + y_{l,r_l^i}^{-,i} \right) \quad (8)$$

subject to

$$\sum_{(u,v^j) \in A_{\sigma} \cup A_V} f_{u,v^j} + \sum_{l \in L} \sum_{(u,v^j) \in A_{R,l}} f_{u,v^j} = \sum_{(v^j,w) \in A_{\tau} \cup A_V} f_{v^j,w} + \sum_{l \in L} \sum_{(v^j,w) \in A_{R,l}} f_{v^j,w} \quad \forall v^j \in V_t \quad (9)$$

$$y_{l,v^i}^{+,i} - y_{l,v^i}^{-,i} = \begin{cases} f_{v^i, succ_l(v^i)} & \forall l \in L, i=1, v=r_l^i, j=t(r_l^i) \\ f_{v^i, succ_l(v^i)} - f_{pred_l(v^i), v^i} & \forall l \in L, i=2, \dots, \rho_l-1, v=r_l^i, j=t(r_l^i) \\ -f_{pred_l(v^i), v^i} & \forall l \in L, i=\rho_l, v=r_l^i, j=t(r_l^i) \end{cases} \quad (10)$$

$$f_{\sigma, v^j} = p_v \quad \forall (\sigma, v^j) \in A_{\sigma} \quad (11)$$

$$f_{v^j, \tau} - q_v \leq \delta_v \quad \forall (v^j, \tau) \in A_{\tau} \quad (12)$$

$$q_v - f_{v^j, \tau} \leq \delta_v \quad \forall (v^j, \tau) \in A_{\tau} \quad (13)$$

$$0 \leq f_{v^j, \tau} \leq C_v \quad \forall (v^j, \tau) \in A_{\tau} \quad (14)$$

$$0 \leq f_{u^j, v^j} \leq Z_l \quad \forall l \in L, (u^j, v^j) \in A_{R,l} \quad (15)$$

$$0 \leq f_{v^j, v^j} \leq C_v \quad \forall (v^j, v^j) \in A_V \quad (16)$$

$$\delta_v \geq 0 \quad \forall (v^j, \tau) \in A_{\tau} \quad (17)$$

$$y_{l,v^i}^{+,i} \in \{0, \dots, Z_l\} \quad \forall l \in L, v \in V, i=1, \dots, \rho_l \quad (18)$$

$$y_{l,v^i}^{-,i} \in \{0, \dots, Z_l\} \quad \forall l \in L, v \in V, i=1, \dots, \rho_l \quad (19)$$

The objective function (8) is directly derived from our main objective (3). Equations (9) are the flow conservation equalities, while equations (10) link the loading instruction variables with the flows. The flows at arcs $(\sigma, v^j) \in A_{\sigma}$ are fixed to the station's initial number of bikes p_v in (11).

As we have a capacitated but unrestricted flow network with all capacities being integer, the LP is totally unimodular and the corresponding polytope's extreme points are all integer. Therefore by solving this LP with a common LP solver (or more specifically a network simplex algorithm), we obtain optimal integral values for the loading instructions.

5.2 VND and VNS Neighborhood Structures

We use several classical neighborhood structures that were successfully applied in various VRPs together with new structures exploiting specifics of BBSS. Concerning the classical neighborhood structures, we based our design on the experience from [7].

VND Neighborhoods: The following neighborhoods are all searched in a best improvement fashion and applied in the given, static order. Preliminary experiments with a dynamic reordering strategy brought no significant advantages. All created candidate tours are incrementally checked for feasibility with respect to time budgets and infeasible solutions are discarded. For a feasible solution we derive loading instructions by one of the methods from Section 5.1 and remove obsolete nodes without any loading actions.

Remove station (REM-VND): This neighborhood considers all single station removals to avoid unnecessary visits.

Insert unbalanced station (INS-U): This neighborhood includes all feasible solutions where a yet unbalanced station is inserted at any possible position.

Intra-route 2-opt (2-OPT): This is the classical 2-opt neighborhood for the traveling salesman problem, applied individually to each route.

Replace station (REPL): Here, any solution in which one station is replaced by a different, yet unbalanced station is included.

Intra or-opt (OR-OPT): This neighborhood considers all solutions in which sequences of one, two, or three consecutive stations are moved to another place within the same route.

2-opt* inter-route exchange (2-OPT*): This classical neighborhood considers all feasible exchanges of arbitrarily long end segments of two routes.

Intra-route 3-opt (3-OPT): This neighborhood resembles a restricted form of the well-known 3-opt neighborhood, individually applied to each route. For any partitioning of a route into three nonempty subsequences $r_l = (a,b,c)$, the routes (b,a,c) and (a,c,b) are considered. An effective enumeration scheme excludes all solutions of the previous neighborhoods.

VNS Neighborhoods: For diversification, the shaking procedure selects solutions randomly from the following types of VNS neighborhoods, which are all parameterized by δ , yielding a total of 24 individual neighborhoods. During this process, created routes that violate the time budget are repaired by removing stations from the end.

Move sequence (MV-SEQ): Select a sequence of one to $\min(\delta, \rho_l)$ stations at random, delete it, and reinsert it at a random position of a different route. If the original route contains less than δ stations, the whole route is inserted at the target route. Both source and target routes are selected randomly. $\delta \in \{1, \dots, 5, \rho_l\}$.

Exchange sequence (EX-SEQ): Exchange two randomly selected segments of length one to $\min(\delta, \rho_l)$ between two randomly chosen routes. $\delta \in \{1, \dots, 5, \rho_l\}$.

Remove stations (REM-VNS): Consider all stations of all routes and remove each station with probability $\delta \in \{10\%, 14\%, 18\%, 22\%, 26\%, 30\%\}$.

Destroy and recreate (D&R): Select a random position in a randomly chosen route, remove all nodes from this position to the end, and recreate a new end segment by applying a randomized version of the greedy construction heuristic. The randomization is done in the typical GRASP-like way [8] with the threshold parameter set to $\delta \in \{0\%, 4\%, 8\%, 12\%, 16\%, 20\%\}$.

6 Computational Results

We tested our VNS algorithm on a set of instances based on real-world data provided by Citybike Vienna¹ which runs a bike-sharing system with 92 stations. They are generated as follows:

- Travel times $t_{u,v}$, $(u, v) \in A_0$ are real average driving times plus an estimation for parking the vehicle and loading/unloading bikes based on the experience of the drivers.
- The number of currently available bikes p_v at station $v \in V$ is taken from a snapshot of the system.
- The target value q_v is assumed to be 50% of the station's capacity.
- In order to make *perfect balance* at least theoretically possible when having enough time, $\sum_{v \in V} p_v = \sum_{v \in V} q_v$ must hold. This is established by applying small changes to p_v for some randomly chosen stations.
- We derived instances with $|V| \in \{10, 20, 30, 60, 90\}$ stations by choosing them randomly from the pool of 92 stations. In addition we consider one common depot (one of the remaining stations) to be the start and end point for all vehicles.
- We assume a homogeneous fleet of $|L| \in \{1, 2, 3, 5\}$ vehicles with capacity $Z_l = 20$, $\forall l \in L$.
- The total time budget for each vehicle is set to $\hat{t}_l \in \{2h, 4h, 8h\}$.
- Each instance set uses a unique combination of $|V|, |L|, \hat{t}_l$ and contains 30 instances, resulting in a total of 1800 instances².

The scaling factors in the objective function were set to $\alpha^{\text{bal}} = 1$, $\alpha^{\text{load}} = \alpha^{\text{work}} = \frac{1}{10000}$. Using these factors, improving the system balance always has a greater impact on the objective value than reducing the tour lengths or the number of loading operations. The algorithm has been implemented in C++ using GCC 4.6 and each test run was performed on a single core of an Intel Xeon E5540 machine with 2.53 GHz and 3 GB RAM per core. Each run was terminated when no improvement could be achieved within the last 5000 VNS iterations or after a CPU time of one hour. For solving the LP-based approach to determine loading instructions CPLEX 12.4 was used with default settings.

In addition to the VNS algorithm, we implemented a mixed integer programming (MIP) model similar to the sequence-indexed formulation from [2] but adapted to our problem formulation. This model is not able to consider dependencies among vehicles and is therefore restricted to the monotonic case. CPLEX 12.4 with default settings and a CPU-time limit of one hour was used for trying to solve the instances with this model.

¹ <http://www.citybikewien.at/>

² Benchmark instances: https://www.ads.tuwien.ac.at/w/Research/Problem_Instances

Table 1. Results of the MIP approach and the VNS considering the three variants of deriving loading instructions. Each instance set contains 30 instances. All runtimes are in seconds.

Instance s			MIP			VNS with GH			VNS with MF			VNS with LP		
V	L	\hat{t}	ub	lb	\bar{t}_{tot}	obj	sd	\bar{t}_{tot}	obj	sd	\bar{t}_{tot}	obj	sd	\bar{t}_{tot}
10	1	120	28.3477	28.3477	4	28.3477	9.9111	1	28.3477	9.9111	2	28.3477	9.9111	212
10	1	240	4.2942	0.0424	3600	4.2941	3.5524	5	4.2941	3.5524	10	4.2941	3.5524	1332
10	1	480	0.0320	0.0276	3600	0.0317	0.0033	8	0.0317	0.0033	17	0.0317	0.0033	2042
10	2	120	9.8269	9.4768	911	10.0266	6.3028	2	9.9601	6.2475	3	9.9600	6.2475	459
10	2	240	0.0340	0.0322	856	0.0339	0.0043	5	0.0339	0.0043	10	0.0339	0.0043	1441
10	2	480	0.0317	0.0313	1245	0.0317	0.0033	7	0.0317	0.0033	15	0.0317	0.0033	1797
20	2	120	55.8294	26.9012	3600	55.0962	13.2321	4	55.3628	13.3731	8	55.3628	13.3731	1097
20	2	240	19.7884	0.0383	3600	4.3908	3.7546	29	4.2575	3.7276	58	4.2576	3.7275	3600
20	2	480	1.8906	0.0403	3600	0.0614	0.0061	51	0.0615	0.0061	142	0.0614	0.0061	3600
20	3	120	37.3759	1.4777	3600	31.9096	11.9065	7	31.7763	11.8112	13	31.8430	11.8650	1727
20	3	240	6.2083	0.0401	3600	0.0651	0.0060	31	0.0650	0.0060	65	0.0652	0.0060	3600
20	3	480	13.4191	0.0316	3600	0.0616	0.0060	55	0.0614	0.0061	114	0.0614	0.0061	3600
30	2	120	106.9631	56.3908	3600	104.7633	17.7686	6	104.7633	17.7686	12	104.7633	17.7142	1539
30	2	240	74.9886	0.0487	3600	34.7941	10.8729	48	34.6608	10.4812	109	35.1940	10.9637	3600
30	2	480	69.8069	0.0432	3600	0.0926	0.0062	186	0.0925	0.0061	491	0.0928	0.0061	3600
30	3	120	90.4419	16.6454	3600	78.0441	17.2764	10	78.1773	17.0832	21	78.5771	17.2677	2521
30	3	240	61.6715	0.0461	3600	7.1526	4.7495	86	7.1523	4.2272	191	7.6186	4.3543	3600
30	3	480	175.4000	0.0015	3600	0.0925	0.0061	156	0.0925	0.0061	399	0.0928	0.0062	3600
60	3	120	274.3101	157.7350	3600	253.9795	27.8187	20	253.8462	27.6739	45	254.3794	27.3265	3600
60	3	240	370.2000	0.0000	3600	126.7616	20.5332	260	126.8282	20.9660	521	129.2945	20.1347	3600
60	3	480	—	—	—	6.1766	4.1036	1835	6.7758	4.1422	3600	10.1071	5.0800	3601
60	5	120	289.3111	34.9784	3600	197.7411	28.0192	54	196.6749	29.4401	99	197.0747	28.7557	3600
60	5	240	370.2000	0.0000	3600	41.1497	12.6579	725	41.6161	13.3489	1556	47.2145	13.0440	3600
60	5	480	—	—	—	0.1901	0.0090	2006	0.1902	0.0087	3600	0.1938	0.0087	3601
90	3	120	492.2319	290.8990	3600	441.5141	21.0737	35	441.6473	20.8266	82	441.4474	20.8250	3600
90	3	240	566.2667	0.0000	3600	295.1644	15.6493	425	294.5646	16.1776	985	297.3642	15.4610	3601
90	3	480	—	—	—	100.5887	9.6476	3600	101.1221	9.9480	3600	110.5868	9.4745	3601
90	5	120	566.2667	0.0000	3600	375.7435	19.5815	83	376.1432	20.6335	169	376.2767	20.5456	3600
90	5	240	—	—	—	174.9566	13.5297	1411	174.3566	12.7181	3304	184.8218	12.6962	3601
90	5	480	—	—	—	1.2863	1.5549	3600	1.6855	1.6746	3600	9.0772	3.5834	3601

In addition, we also investigated a second MIP model based on a time-indexed formulation [2] for the general case. Experiments indicated that this approach unfortunately led to even worse results due to the higher complexity of the model and a required discretization of station visit times. We therefore omit these results here. Besides documenting the general suitability of the VNS and comparing it to the MIP approach, we aim at analyzing the impacts of the three alternative procedures to derive loading instructions. Table 1 lists average results for 30 instance sets (out of the 60) that appear most relevant for practice. Complete results are available for download with the benchmark instances.

For the MIP approach the table shows mean upper bounds \overline{ub} , mean lower bounds \overline{lb} , and median total run times $\overline{t}_{\text{tot}}$ for the cases where upper or lower bounds could be obtained within the time limit. The other column groups in the table show the results of the three VNS variants with GH, MF and LP applied to obtain loading instructions, respectively. For each variant mean objective values of the final solutions \overline{obj} , their

standard deviations sd , and median total run times $\overline{t_{tot}}$ are listed. In each row best mean results are printed bold.

In general we can clearly observe that the pure MIP approach is only able to solve very small instances to optimality within the time limit. Very large gaps between lower and upper bounds show that it scales badly with increasing numbers of vehicles and especially with longer time budgets. For large instances CPLEX often only found trivial solutions where all vehicles stay at the depot, or even no solutions at all.

Among the three VNS variants, the one applying GH clearly was fastest. MF increased the running time on average by about 120%. The VNS with LP even took about 110 times longer than the VNS with MF on average for those runs that were not terminated by the time limit. Concerning solution quality, we observed that GH is able to obtain results very similar to those of MF. Both variants found better final solutions with lower objective values than the respective other variant in about 21% of runs. In the remaining 58% both approaches obtained equally good results. Objective values are on average slightly better for the MF-variant. In general, however, absolute quality differences are rather small. Also, a Wilcoxon signed-rank test does not show a significant difference regarding solution quality of GH and MF. MF runs were terminated by the time limit for the largest 8% of instances. When only comparing runs not terminated by the time limit, average objective values are more favorable for MF. However, also in this comparison the improvement over GH cannot be said to be statistically significant.

In principle the VNS with LP is sometimes able to obtain better results than the other variants since it may take advantage of not being restricted to monotonicity. Due to the substantially higher computational overhead, however, about 60% of all runs were terminated before a reasonable convergence had been achieved due to exceeding the time limit of one hour. Therefore, the LP-approach typically led to significantly worse results, particularly for larger instances. The LP-variant obtained better solutions in only 10%, while the MF-variant outperformed the LP-variant in 36% of all runs. A Wilcoxon signed-rank test confirms the assumption that the VNS with MF performs better w.r.t. solution quality with a very low error probability of less than 0.01%.

Figure 3 shows typical relative success rates for the VND neighborhoods on a large instance. In the VNS, all shaking neighborhoods have similar relative success rates, therefore we omit the corresponding chart. These results show that all neighborhood structures contribute well to the overall performance.

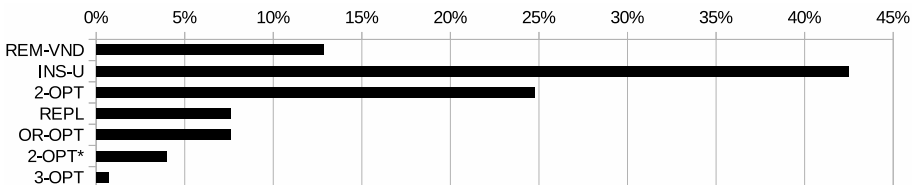


Fig. 3. Relative success rates of VND neighborhoods for an instance with $|V| = 90$, $|L| = 5$, $\hat{t} = 480$ using the MF-variant

7 Conclusions and Future Work

We presented a VNS metaheuristic with an embedded VND for solving the balancing bicycle sharing system problem. Main ingredients are a meaningful greedy construction heuristic for generating initial solutions, neighborhood structures derived from VRPs, new problem-specific neighborhood structures, as well as three alternatives for deriving optimized loading instructions for created candidate tours. Experimental results on instances derived from real-world data show that the VNS in general performs well and scales much better than two MIP approaches. Concerning the derivation of loading instructions, the greedy method is fastest and delivers solutions similar in quality to those of the more complex maximum flow based approach. The LP-based method has the advantage of being able to find optimal loading instructions even for the general, not necessarily monotonic case, but unfortunately the added flexibility cannot compensate the typically much larger computational effort when considering reasonable runtime limits. Thus, the fast greedy method is the best compromise for practice.

In future work, we intend to model the times needed for loading bikes at a station more accurately by taking the number of loading actions into account instead of assuming average dwell times. Another practically relevant extension is to allow vehicles to start and return nonempty. Finally, we also want to turn towards the dynamic scenario, where the fill levels at stations change during the balancing process. Stochastic aspects then also need to be considered. Last but not least, hybridizing the VNS with the MIP approaches, e.g., by including some MIP-based large neighborhood search, appears to be promising.

References

1. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. To appear in *Discrete Optimization* (2012)
2. Raviv, T., Tzur, M., Forma, I.A.: Static Repositioning in a Bike-Sharing System: Models and Solution Approaches. To appear in *EURO Journal on Transportation and Logistics* (2012)
3. Benchimol, M., Benchimol, P., Chappert, B., De la Taille, A., Laroche, F., Meunier, F., Robinet, L.: Balancing the stations of a self service bike hire system. *RAIRO – Operations Research* 45(1), 37–61 (2011)
4. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a Dynamic Public Bike-Sharing System. Technical Report CIRRELT-2012-09, CIRRELT, Montreal, Canada (2012), submitted to *Transportation Science*
5. Mladenović, N., Hansen, P.: Variable neighborhood search. *Computers and Operations Research* 24(11), 1097–1100 (1997)
6. Cherkassky, B.V., Goldberg, A.V.: On implementing the push-relabel method for the maximum flow problem. *Algorithmica* 19(4), 390–410 (1997)
7. Pirkwieser, S., Raidl, G.R.: A variable neighborhood search for the periodic vehicle routing problem with time windows. In: Prodhon, C., et al. (eds.) *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing*, Troyes, France (2008)
8. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 219–249. Kluwer Academic Publishers (2003)

Combinatorial Neighborhood Topology Particle Swarm Optimization Algorithm for the Vehicle Routing Problem

Yannis Marinakis¹ and Magdalene Marinaki²

¹ Decision Support Systems Laboratory, Department of Production Engineering and Management, Technical University of Crete, Chania, Greece

marinakis@ergasya.tuc.gr

² Industrial Systems Control Laboratory, Department of Production Engineering and Management, Technical University of Crete, Chania, Greece

magda@dssl.tuc.gr

Abstract. One of the main problems in the application of a Particle Swarm Optimization in combinatorial optimization problems, especially in routing type problems like the Traveling Salesman Problem, the Vehicle Routing Problem, etc., is the fact that the basic equation of the Particle Swarm Optimization algorithm is suitable for continuous optimization problems and the transformation of this equation in the discrete space may cause loose of information and may simultaneously need a large number of iterations and the addition of a powerful local search algorithm in order to find an optimum solution. In this paper, we propose a different way to calculate the position of each particle which will not lead to any loose of information and will speed up the whole procedure. This was achieved by replacing the equation of positions with a novel procedure that includes a Path Relinking Strategy and a different correspondence of the velocities with the path that will follow each particle. The algorithm is used for the solution of the Capacitated Vehicle Routing Problem and is tested in the two classic set of benchmark instances from the literature with very good results.

Keywords: Particle Swarm Optimization, Variable Neighborhood Search, Path Relinking, Vehicle Routing Problem.

1 Introduction

Particle Swarm Optimization (PSO) is a population-based swarm intelligence algorithm that was originally proposed by Kennedy and Eberhart [9]. PSO simulates the social behavior of social organisms by using the physical movements of the individuals in the swarm. Its mechanism enhances and adapts to the global and local exploration. Most applications of PSO have concentrated on the optimization in continuous space but in the last years the PSO algorithm is used also in discrete optimization problems. The Particle Swarm Optimization

is a very popular optimization method and its wide use, mainly during the last years, is due to the number of advantages that this method has, compared to other optimization methods. Some of the key advantages are that this method does not need the calculation of derivatives, that the knowledge of good solutions is retained by all particles and that particles in the swarm share information between them. PSO is less sensitive to the nature of the objective function, can be used for stochastic objective functions and can easily escape from local minima. Concerning its implementation, PSO has few parameters to regulate and the assessment of the optimum is independent of the initial solution.

There is a number of different representations of the solution of a particle in order to take advantage of the equations of Particle Swarm Optimization. For example, the most commonly used transformation is the relative position indexing. Initially, the tour is represented via path representation, which is not suitable for the PSO. Then, a transformation into a floating point in the interval $(0, 1]$ is performed. Afterwards, the velocities of all particles are calculated and, then, the solution of each particle is transformed back to the integer domain and the cost is calculated. This implementation, although includes the risk of losing information, has given very good results in various application of routing problems [14,15]. An almost similar approach has been proposed in [1]. A very efficient encoding of the position of each particle by using an adjacency matrix has been proposed in [12] in order to be able to apply the binary version of the Particle Swarm Optimization. A similar approach using matrix based particle representation has been presented in [10]. In [5] a new position update rule is used where the authors propose to update the position of each particle using mutation and crossover operators developed for permutation encoding in the literature and more precisely using swap and partially mapping crossover (PMX) operators.

In this paper, we propose a different way to calculate the position of each particle which will not lead to any losing of information and will speed up the whole procedure. An initial version of this idea was presented by Marinakis and Marinaki in [11,13,15] and, then, was improved by Rosendo and Pozo in [22]. The equation of positions is replaced by the use of a Path Relinking procedure. The path relinking generates new solutions by exploring trajectories that connect high-quality solutions by starting from one of these solutions, called the *starting solution* and generating a path in the neighborhood space that leads towards the other solution, called the *target solution* [4]. The reason that the Path Relinking algorithm is used to replace the position equation of the PSO is that there is a correspondence between the idea behind the Path Relinking and the idea of movement of a particle in the Particle Swarm Optimization algorithm. More precisely, a particle in Particle Swarm Optimization can either follow its own path, or go towards to its previous optimal solution, or go towards to the global optimal solution (to the best particle in the swarm). When the particle decides to follow either the path to its previous optimal solution or the path to the global optimal solution, a path relinking strategy is applied where the current solution plays the role of the starting solution and the best particle of the swarm or the current best solution of the particle plays the role of the target solution.

The trajectories between the two solutions are explored by simple swapping of two nodes of the starting solution until the starting solution becomes equal to the target solution. The paths are generated by choosing moves in order to introduce attributes in the starting solution that are present in the guiding target solution. If in some step of the path relinking strategy a new best solution, either of the particle or of the whole swarm, is found, then, the current best (particle or swarm) solution is replaced with the new one and the algorithm continues.

In this paper, we propose an improvement of this idea where two different Path Relinking strategies are used. In the first one, the path relinking strategy is depended from the average values of the equations of velocities, meaning that for each particle the average value of the equation of velocities is calculated and, then, depended on this value, the particle chooses which one of the three movements will perform. In the second one, a new path relinking strategy is proposed. In this strategy, the three possible movements of a particle are performed simultaneously in the solution, thus, depending on the velocities' equation either no path relinking is performed, or a path relinking is performed using the previous best solution of the particle or a path relinking is performed using the global best solution. Afterwards, a Variable Neighborhood Search (VNS) [8] is used to improve the solution of each particle.

The algorithm is used for solving the Capacitated Vehicle Routing Problem. The **Vehicle Routing Problem (VRP)** or the **capacitated vehicle routing problem (CVRP)** is often described as the problem in which vehicles based on a central depot are required to visit geographically dispersed customers in order to fulfill known customer demands. The problem is to construct a low cost, feasible set of routes - one for each vehicle. A route is a sequence of locations that a vehicle must visit along with the indication of the service it provides. The vehicle must start and finish its tour at the depot. As the Vehicle Routing Problem is an NP-hard problem, a large number of approximation techniques were proposed. These techniques are classified into two main categories: Classical heuristics that were developed mostly between 1960 and 1990 and metaheuristics that were developed in the last twenty five years. In the last ten years a number of evolutionary and nature inspired algorithms have been applied for the solution of the Vehicle Routing Problem. The reader can find more detailed descriptions of these algorithms in the books [7,24].

With the new version of the combined PSO and Path Relinking Algorithm, the Combinatorial Neighborhood Topology Particle Swarm Optimization, the continuous values of the velocities' equation are not used at all in the solution of a particle and no transformation between integer and continuous values is needed in order to apply the Particle Swarm Optimization in a combinatorial optimization problem, and, thus, no information about good solutions is loosed between two consecutive iterations. We apply the proposed algorithm in a number of benchmark instances from the literature. Then, we compare the results of the proposed algorithm with the results of other PSO implementations from the literature and, finally, we compare the results with the results of the other algorithms from the literature. The rest of the paper is organized as follows:

In the next section the proposed algorithm, the Combinatorial Neighborhood Topology Particle Swarm Optimization (CNTPSO), is presented and analyzed in detail. Computational results are presented and analyzed in the third section while in the last section conclusions and future research are given.

2 Combinatorial Neighborhood Topology Particle Swarm Optimization Algorithm

In this paper, a new topology for Particle Swarm Optimization (PSO) for suitably solving combinatorial optimization problems is presented. In a PSO algorithm, initially, a set of particles is created randomly where each particle corresponds to a possible solution. Each particle has a position in the space of solutions and moves with a given velocity. One of the key issues in designing a successful PSO for the Vehicle Routing Problem is to find a suitable mapping between Vehicle Routing Problem solutions and particles in PSO. Each particle is recorded via the path representation of the tour, that is, via the specific sequence of the nodes.

Usually in a PSO implementation the calculation of the new position is given by Equation 2 (see below) and, thus, the movement of a particle between two positions is depending directly of the calculation of the velocities performed by Equation (1) (see below). Thus, the above mentioned representation should be transformed appropriately. However, in this paper the calculation of the velocities using a novel Path Relinking procedure lead the algorithm to keep in each iteration the path representation of each solution without needing to transform the solutions in the continuous space.

In a classic Particle Swarm Optimization algorithm, the position of each individual (called particle) is represented by a d -dimensional vector in problem space $x_i = (x_{i1}, x_{i2}, \dots, x_{id})$, $i = 1, 2, \dots, N$ (N is the population size and d is the number of the vector's dimension), and its performance is evaluated on the predefined fitness function. The velocity v_{ij} represents the changes that will be made to move the particle from one position to another. Where the particle will move depends on the dynamic interaction of its own experience and the experience of the whole swarm. There are three possible directions that a particle can follow: to follow its own path, to move towards the best position it had during the iterations ($pbest_{ij}$) or to move to the best particle's position ($gbest_j$). The velocity and position equations are updated as follows (constriction PSO) [3]:

$$v_{ij}(t+1) = \chi(v_{ij}(t) + c_1 rand_1(pbest_{ij} - x_{ij}(t)) + c_2 rand_2(gbest_j - x_{ij}(t))) \quad (1)$$

and

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (2)$$

where

$$\chi = \frac{2}{|2 - c - \sqrt{c^2 - 4c}|} \text{ and } c = c_1 + c_2, c > 4 \quad (3)$$

t is the iterations counter, c_1 and c_2 are the acceleration coefficients, $rand_1$ and $rand_2$ are two random variables in the interval $(0, 1)$. In this paper, Equation 2 is not used at all. From the other hand the role of the velocities' equation is very important. As it was mentioned previously the equation of positions of each particle has been replaced by a Path Relinking Strategy. The Path Relinking [4] is an intensification strategy that is used as a way of exploring trajectories between elite solutions. Thus, the idea behind the Path Relinking is almost the same as the idea behind the movement of a particle in the swarm. A particle in Particle Swarm Optimization can either follow its own path, or go towards to its previous optimal solution, or go towards to the global optimal solution (to the best particle in the swarm). Thus, in the CNTPSO when the particle decides to follow either the path to its previous optimal solution or the path to the global optimal solution, a path relinking strategy is applied where the current solution plays the role of the starting solution and the best particle of the swarm or the current best solution of the particle plays the role of the target solution. The trajectories between the two solutions are explored by simple swapping of two nodes of the starting solution until the starting solution becomes equal to the target solution.

The most important part of this algorithm is how the particle decides to follows the previous best or the global best of the whole swarm. Inspired by the equation of the inertia PSO we would like to give to the particles more exploration abilities in the initial iterations of the algorithm and as the iterations proceed to reduce the ability of the particle to search in a different solution space and to increase the possibility to search around its previous best solution and around the global optimum. Thus, in the first version of the path relinking, initially, we calculate the average value of the velocity equation of each particle.

$$\text{average}_v = \frac{\sum_{j=1}^d v_{ij}(t+1)}{d} \quad (4)$$

If this value is less than a number L_1 , then, no path relinking is performed (NOPR), meaning that the particle follows its own path (using a local search procedure as it will be described later), if it is between the number L_1 and L_2 the particle performs a path relinking with its previous best solution (PRPB), and if the value is greater than L_2 the particle performs a path relinking with the global optimum solution (PRGB). In the beginning of the algorithm, L_1 and L_2 have large values and the values are decreased during the iterations in order to succeed the increase of the possibility of a Path Relinking with the global or the personal best and the decrease of the possibility of searching by its own, using a local search, the solution space. Thus, the values of L_1 and L_2 are

$$L_1 = (u_{bound} - l_{bound}) \times (w_1 - \frac{w_1}{iter_{max}} \times t) + l_{bound} \quad (5)$$

and

$$L_2 = (u_{bound} - l_{bound}) \times (w_2 - \frac{w_2}{2 * iter_{max}} \times t) + l_{bound} \quad (6)$$

where, t is the current iteration and $iter_{max}$ is the maximum number of iterations, u_{bound} and l_{bound} are the upper and lower bounds for the velocities of each particle. Usually in the literature the values for the upper and lower bounds are $+4$, -4 , respectively. If in some iterations in an element of the particle the value of the velocity violates these bounds, then, this element is initialized with a new value inside the bounds. The parameters w_1 and w_2 should have large values as it is desired the value of L_1 to be as large as possible in the beginning of the algorithm and to be reduced during the iterations. Also, the value of L_2 should be larger than the value of L_1 and, thus, the value of w_2 should be larger than the value of w_1 . Thus, it is selected $w_1 = 0.8$ and as it is decided to give to the two other strategies (PRPG and PRGB) almost equal selection possibility the value of w_2 is selected equal to 0.9.

In the second version of Path Relinking the same procedure are followed but instead of using the average value of the velocities it is examined for each element of the velocities' equation of the particle which part of the condition with the L_1 and L_2 values holds. Thus, in the end of this procedure each particle is divided in three parts. For the elements of the solution that belong to the first part no path relinking is applied and in these elements a local search is applied, for the elements of the solution that belong to the second part a path relinking procedure is applied with the previous personal best and for the last part a path relinking is applied using the global best solution. As it is possible with the use of the path relinking procedure the solutions to converge near to a single solution in a short number of iterations, a solution is not selected if it differs from the optimum solution less than 30% except if this solution improves the optimum solution. A local search strategy based on the Variable Neighborhood Search (VNS) algorithm [8] is applied in each particle in the swarm in order to improve the solutions produced from the particle swarm optimization algorithm. In this paper, the VNS algorithm is used with the following way. Initially, the number of local search algorithms is selected. The local search strategies for the Vehicle Routing Problem are distinguished between local search strategies for a single route and local search strategies for multiple routes. The local search strategies that are chosen and belong to the category of the single route interchange are the well known methods for the TSP, the 2-opt and the 3-opt. In the single route interchange all the routes have been created in the initial phase of the algorithm. The Local Search Strategies for Single Route Interchange try to improve the routing decisions. The Local Search Strategies for Multiple Route Interchange try to improve the assignment decisions. This, of course, increases the complexity of the algorithms but gives the possibility to improve even more the solution. The multiple route interchange local search strategies that are used are the 1-0 relocate, 2-0 relocate, 1-1 exchange and 2-2 exchange.

As we do not want to increase the complexity of the algorithm, it is decided to apply in each particle one local search combination of algorithms per iteration. For this reason, a VNS operator C_{VNS} is selected that controls which local search algorithm is applied. The C_{VNS} value is compared with the output of a random number generator, $rand_i(0, 1)$. If the random number is less or equal to the

C_{VNS} , then, the first local search algorithm is used. Then, if the random number is less or equal to the $2 * C_{VNS}$, then, the second local search algorithm is used, and so on. As we would like to have not only simple local search algorithms but also their combinations we select ten local search algorithms, the six previously mentioned methods (2-opt, 3-opt, 1-0 relocate, 2-0 relocate, 1-1 exchange and 2-2 exchange) and four combinations (2-opt with 1-0 relocate, 2-opt with 1-1 exchange, 2-opt with 3-opt and 1-1 exchange and 2-opt with 3-opt, 1-0 relocate and 1-1 exchange). Thus, the C_{VNS} operator is set equal to 0.1 and only for 10% of the cases a time consuming local search procedure is applied in the problem. In each iteration of the algorithm the optimal solution of the whole swarm and the optimal solution of each particle are kept. The algorithm stops when a maximum number of iterations has been reached.

3 Results and Discussion

The algorithm was implemented in Fortran 90 and was compiled using the Lahey f95 compiler on a Intel Core 2 DUO CPU T9550 at 2.66 GHz, running Suse Linux 9.1. The algorithm was tested on two sets of benchmark problems. The 14 benchmark problems proposed by Christofides [2] and the 20 large scale vehicle routing problems proposed by Golden [6]. Each instance of the first set contains between 51 and 200 nodes including the depot. Each problem includes capacity constraints while the problems 6-10, 13 and 14 have, also, maximum route length restrictions (m.t.l.) and non zero service times (s.t.). For the first ten problems, nodes are randomly located over a square, while for the remaining ones, nodes are distributed in clusters and the depot is not centered. The second set of instances contains between 200 and 483 nodes including the depot. Each problem instance includes capacity constraints while the first eight have, also, maximum route length restrictions but with zero service times. The efficiency of the two versions of CNTPSO algorithm (CNTPSO1 and CNTPSO2, where in the CNTPSO1 the first version of Path Relinking is used while in the CNTPSO2 the second version is used) is measured by the quality of the produced solutions. The quality is given in terms of the relative deviation from the best known solution, $\omega = \frac{(c_{CNTPSO} - c_{BKS})}{c_{BKS}}$ and is denoted in Tables 1 and 2 with ω_1 and ω_2 for CNTPSO1 and CNTPSO2, respectively. In these Tables in columns 1-4 the most important characteristics (number of nodes (n), Capacity of Vehicles (Q), maximum route length restrictions (mtl) and service times (st)) of each of the data sets are presented. The parameters of the proposed algorithm are selected after thorough testing. A number of different alternative values were tested and the ones selected are those that gave the best computational results concerning both the quality of the solution and the computational time needed to achieve this solution. The selected parameters are: number of particles equal to 50, number of iterations equal to 1000, $c_1 = c_2 = 2.05$ and $w_1 = 0.8, w_2 = 0.9$. After the selection of the final parameters, 10 different runs with the selected parameters were performed for each of the benchmark instances.

As it was mentioned previously a maximum number of iterations was selected. There are two issues that someone should take into account when he/she designs

an algorithm: the first one is the number of iterations with no recorded improvement and the other one is the maximum running time. In order to cope with the first issue a number of iterations without improvement was selected (50 in all instances) and, then, the algorithm was terminated. For the second issue a maximum computational time was selected for each instance. As the increase of the number of customers needs more computational time to find good solutions we could not use the same maximum computational time for all instances. Thus, the maximum allowed time when we solved the first set of benchmark instances was set equal to 200 seconds while when we solved the second set of benchmark instances the maximum allowed time was set equal to 500 seconds. However, in most instances the maximum allowed time was not reached and the algorithm was terminated with the use of the maximum iterations or the iteration without improvement. In the last five columns of Tables 1 and 2, the best known solution (BKS - column 5), the results of the proposed algorithm (column 5), the quality of the solution of the proposed algorithm (ω_1 - column 6) for the CNTPSO1 and the results of the proposed algorithm (column 7), the quality of the solution of the proposed algorithm (ω_2 - column 8) for the CNTPSO2 are presented, respectively. It can be seen from Table 1 that the CNTPSO1 algorithm in nine out of the fourteen instances of the first set has reached the best known solution. For the other five instances the quality of the solutions is between 0.01% and 0.15% and the average quality for the fourteen instances is 0.033%. On the other hand, the CNTPSO2 in eleven out of the fourteen instances of the first set has reached the best known solution. For the other three instances the quality of the solutions is between 0.05% and 0.14% and the average quality for the fourteen instances is 0.019%. For the 20 large scale vehicle routing problems (Table 2) both algorithms have found the best known solution in one of them, for the rest the quality is between 0.10% and 1.08% and the average quality of the solutions is 0.44% for the CNTPSO1, while for the CNTPSO2 the quality is between 0.02% and 0.81% and the average quality of the solutions is 0.37%. In general both algorithms gave very good results in all instances in short computational time but the second version with the combined Path Relinking procedure performs slightly better than the other method.

In Tables 3 and 4 comparisons of the proposed algorithm with other algorithms from the literature are presented. The first algorithm is a hybrid Particle Swarm Optimization (HybPSO) algorithm for the solution of the Vehicle Routing Problem [15]. The second algorithm used for the comparisons is a Hybridization version of Particle Swarm Optimization with a Genetic Algorithm (HybGENPSO) for the solution of the Vehicle Routing Problem [13]. The next two algorithms, the PSOSR1 and the PSOSR2, are presented and analyzed in [1] and the last one (PSOMAT) is presented in [10]. For the last three algorithms the authors did not present results for the large scale VRPs and, thus, comparisons could be performed only for the small instances. In the first set of instances the CNTPSO2 found the best solution in eleven instances, the HybGENPSO in ten instances, the CNTPSO1 in 9 instances, the HybPSO in seven instances, the PSOMAT in six instances, the PSOSR2 in four instances and the PSOSR1 in

Table 1. Results of CNTPSO algorithm in Christofides benchmark instances

n	Q	m.t.l.	s.t.	BKS	CNTPSO1	ω_1 (%)	CNTPSO2	ω_2 (%)
51	160	∞	0	524.61 [21]	524.61	0.00	524.61	0.00
76	140	∞	0	835.26 [21]	835.26	0.00	835.26	0.00
101	200	∞	0	826.14 [21]	826.14	0.00	826.14	0.00
151	200	∞	0	1028.42 [21]	1028.42	0.00	1028.42	0.00
200	200	∞	0	1291.45 [21]	1293.18	0.15	1292.35	0.08
51	160	200	10	555.43 [21]	555.43	0.00	555.43	0.00
76	140	160	10	909.68 [21]	909.68	0.00	909.68	0.00
101	200	230	10	865.94 [21]	865.94	0.00	865.94	0.00
151	200	200	10	1162.55 [21]	1163.28	0.06	1163.15	0.05
200	200	200	10	1395.85 [21]	1396.05	0.01	1395.85	0.00
121	200	∞	0	1042.11 [21]	1043.15	0.10	1042.11	0.00
101	200	∞	0	819.56 [21]	819.56	0.00	819.56	0.00
121	200	720	50	1541.14 [21]	1543.35	0.14	1543.25	0.14
101	200	1040	90	866.37 [21]	866.37	0.00	866.37	0.00

Table 2. Results of CNTPSO in the 20 benchmark Golden instances

n	Q	m.t.l.	s.t.	BKS	CNTPSO1	ω_1 (%)	CNTPSO2	ω_2 (%)
240	550	650	0	5627.54 [16]	5688.25	1.08	5665.35	0.67
320	700	900	0	8444.50 [19]	8458.45	0.17	8455.15	0.13
400	900	1200	0	11036.22 [20]	11098.15	0.56	11094.15	0.52
480	1000	1600	0	13624.52 [18]	13695.22	0.52	13688.75	0.47
200	900	1800	0	6460.98 [23]	6460.98	0.00	6460.98	0.00
280	900	1500	0	8412.8 [18]	8445.55	0.39	8435.28	0.27
360	900	1300	0	10181.75 [17]	10201.24	0.19	10198.15	0.16
440	900	1200	0	11643.90 [19]	11715.35	0.61	11712.41	0.59
255	1000	∞	0	583.39 [16]	586.15	0.47	586.15	0.47
323	1000	∞	0	741.56 [16]	743.18	0.22	743.25	0.23
399	1000	∞	0	918.45 [16]	924.59	0.67	924.15	0.62
483	1000	∞	0	1107.19 [16]	1117.45	0.93	1116.12	0.81
252	1000	∞	0	859.11 [16]	862.35	0.38	862.35	0.38
320	1000	∞	0	1081.31 [16]	1086.24	0.46	1086.07	0.44
396	1000	∞	0	1345.23 [16]	1352.39	0.53	1352.21	0.52
480	1000	∞	0	1622.69 [16]	1632.35	0.60	1632.21	0.59
240	200	∞	0	707.79 [16]	709.45	0.23	708.76	0.14
300	200	∞	0	997.52 [16]	1001.15	0.36	998.83	0.13
360	200	∞	0	1366.86 [16]	1368.24	0.10	1367.20	0.02
420	200	∞	0	1820.09 [16]	1825.35	0.29	1822.94	0.16

one instance. The average quality in all instances for the CNTPSO2 is 0.019%, for the CNTPSO1 is 0.033%, for the HybGENPSO is 0.046%, for the HybPSO is 0.084%, for the PSOMAT is 0.71%, for the PSOSR2 is 0.88% and for the PSOSR1 is 1.93%. It can be observed that the results of the CNTPSO2 are better than the results of the other six algorithms, while the results of CNTPSO1 are almost equally good with the results of the HybGENPSO and HybPSO. The other three algorithms used in the comparisons did not give as good results as the proposed algorithms. For the second set of instances the average quality of the CNTPSO2 is 0.37%, for the CNTPSO1 is 0.44%, for the HybGENPSO is 0.60%, for the HybPSO is 0.52%. Thus, the CNTPSO2, also, performs better in the second set of instances. In this set of instances the CNTPSO1 is the second

best algorithm, while the HybPSO is the third algorithm. The HybGENPSO which was the second best algorithm in the previous set of instances now gives slightly inferior results from the other algorithms. As we can see from these comparisons the proposed algorithm is the most efficient implementation of Particle Swarm Optimization for the solution of the Capacitated Vehicle Routing Problem. This is due to the very efficient representation of the solutions and to the fact that there is no loose of any information by using transformations to continuous space.

Table 3. Comparison of the proposed algorithm with implementations of PSO from the literature in the 14 Christofides benchmark instances

CNTPSO1		CNTPSO2		HybPSO		HybGENPSO		PSOSR1		PSOSR2		PSOMAT	
cost	ω (%)	cost	ω (%)	cost	ω (%)	cost	ω (%)	cost	ω (%)	cost	ω (%)	cost	ω (%)
524.61	0.00	524.61	0.00	524.61	0.00	524.61	0.00	524.61	0.00	524.61	0.00	524.61	0.00
835.26	0.00	835.26	0.00	835.26	0.00	835.26	0.00	849.58	1.71	844.42	1.10	835.26	0.00
826.14	0.00	826.14	0.00	826.14	0.00	826.14	0.00	835.80	1.17	829.40	0.39	830.26	0.50
1028.42	0.00	1028.42	0.00	1029.54	0.11	1028.42	0.00	1067.57	3.81	1048.89	1.99	1047.72	1.88
1293.18	0.15	1292.35	0.08	1294.13	0.22	1294.21	0.23	1345.84	4.22	1323.89	2.52	1329.59	2.97
555.43	0.00	555.43	0.00	555.43	0.00	555.43	0.00	556.68	0.23	555.43	0.00	555.43	0.00
909.68	0.00	909.68	0.00	909.68	0.00	909.68	0.00	952.77	4.74	917.68	0.88	913.24	0.39
865.94	0.00	865.94	0.00	868.45	0.29	865.94	0.00	877.84	1.37	867.01	0.12	865.94	0.00
1163.28	0.06	1163.15	0.05	1164.35	0.15	1163.41	0.07	Infeasible		1181.14	1.60	1178.49	1.37
1396.05	0.01	1395.85	0.00	1396.18	0.02	1397.51	0.12	1465.66	5.00	1428.46	2.34	1431.04	2.52
1043.15	0.10	1042.11	0.00	1044.03	0.18	1042.11	0.00	1051.87	0.94	1052.34	0.98	1042.97	0.08
819.56	0.00	819.56	0.00	819.56	0.00	819.56	0.00	820.62	0.13	819.56	0.00	819.56	0.00
1543.35	0.14	1543.25	0.14	1544.18	0.20	1544.57	0.22	1566.32	1.63	1546.20	0.33	1545.56	0.29
866.37	0.00	866.37	0.00	866.37	0.00	866.37	0.00	867.13	0.09	866.37	0.00	866.37	0.00

Table 4. Comparison of the proposed algorithm with implementations of PSO from the literature in the 20 Golden instances

CNTPSO1		CNTPSO2		HybPSO		HybGENPSO	
cost	ω (%)	cost	ω (%)	cost	ω (%)	cost	ω (%)
5688.25	1.08	5665.35	0.67	5695.14	1.20	5670.38	0.76
8458.45	0.17	8455.15	0.13	8461.32	0.20	8459.73	0.18
11098.15	0.56	11094.15	0.52	11098.35	0.56	11101.12	0.59
13695.22	0.52	13688.75	0.47	13695.51	0.52	13698.17	0.54
6460.98	0.00	6460.98	0.00	6462.35	0.02	6460.98	0.00
8445.55	0.39	8435.28	0.27	8461.18	0.58	8470.64	0.69
10201.24	0.19	10198.15	0.16	10202.41	0.20	10215.14	0.33
11715.35	0.61	11712.41	0.59	11715.35	0.61	11750.38	0.91
586.15	0.47	586.15	0.47	586.29	0.50	586.87	0.60
743.18	0.22	743.25	0.23	743.57	0.27	746.56	0.67
924.59	0.67	924.15	0.62	928.49	1.09	925.52	0.77
1117.45	0.93	1116.12	0.81	1118.57	1.03	1114.31	0.64
862.35	0.38	862.35	0.38	862.35	0.38	865.19	0.71
1086.24	0.46	1086.07	0.44	1088.37	0.65	1089.21	0.73
1352.39	0.53	1352.21	0.52	1352.21	0.52	1355.28	0.75
1632.35	0.60	1632.21	0.59	1632.28	0.59	1632.21	0.59
709.45	0.23	708.76	0.14	710.87	0.44	712.18	0.62
1001.15	0.36	998.83	0.13	1002.59	0.51	1006.31	0.88
1368.24	0.10	1367.20	0.02	1368.57	0.13	1373.24	0.47
1825.35	0.29	1822.94	0.16	1826.74	0.37	1831.17	0.61

4 Conclusions

In this paper, a new algorithm based on the Particle Swarm Optimization for the solution of the Vehicle Routing Problem is presented. This algorithm is a hybridization of the Particle Swarm Optimization algorithm with the Variable Neighborhood Search algorithm. As a number of different variants of the Particle Swarm Optimization algorithm have been published, mainly using a different equation for the calculation of the velocities, we used the constriction Particle Swarm Optimization. The most important issue that we have to deal with was the fact that the PSO algorithm is suitable for continuous optimization problems. Thus, it was a challenge to find an effective transformation of the solutions of PSO in discrete values without losing information from this procedure. This was achieved by replacing the equation of positions with a novel procedure that includes a Path Relinking Strategy and a different correspondence of the velocities with the path that will follow each particle. The algorithm was tested in the two set of benchmark instances that are usually used in the literature with very good results. Our future research will be focused in the application of this idea in other more difficult routing problems.

References

1. Ai, T.J., Kachitvichyanukul, V.: Particle swarm optimization and two solution representations for solving the capacitated vehicle routing problem. *Computers and Industrial Engineering* 56, 380–387 (2009)
2. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (eds.) *Combinatorial Optimization*. Wiley, Chichester (1979)
3. Clerc, M., Kennedy, J.: The particle swarm: explosion, stability and convergence in a multi-dimensional complex space. *IEEE Transactions on Evolutionary Computation* 6, 58–73 (2002)
4. Glover, F., Laguna, M., Marti, R.: Scatter search and path relinking: Advances and applications. In: Glover, F., Kochenberger, G.A. (eds.) *Handbook of Metaheuristics*, pp. 1–36. Kluwer Academic Publishers, Boston (2003)
5. Goksal, F.P., Karaoglan, I., Altiparmak, F.: A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. *Computers and Industrial Engineering* (2012), doi:10.1016/j.cie.2012.01.005
6. Golden, B.L., Wasil, E.A., Kelly, J.P., Chao, I.M.: The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Crainic, T.G., Laporte, G. (eds.) *Fleet Management and Logistics*, pp. 33–56. Kluwer Academic Publishers, Boston (1998)
7. Golden, B., Raghavan, S., Wasil, E.: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer LLC (2008)
8. Hansen, P., Mladenovic, N.: Variable neighborhood search: Principles and applications. *European Journal of Operational Research* 130, 449–467 (2001)
9. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: *Proceedings of 1995 IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948 (1995)
10. Kim, B.I., Song, S.J.: A probability matrix based particle swarm optimization for the capacitated vehicle routing problem. *Journal Intelligence Manufacturing* 23, 1119–1126 (2012)

11. Marinakis, Y., Marinaki, M.: A Particle Swarm Optimization Algorithm with Path Relinking for the Location Routing Problem. *Journal of Mathematical Modelling and Algorithms* 7(1), 59–78 (2008)
12. Marinakis, Y., Marinaki, M.: A Hybrid Multi-Swarm Particle Swarm Optimization Algorithm for the Probabilistic Traveling Salesman Problem. *Computers and Operations Research* 37, 432–442 (2010)
13. Marinakis, Y., Marinaki, M.: A Hybrid Genetic - Particle Swarm Optimization Algorithm for the Vehicle Routing Problem. *Expert Systems with Applications* 37, 1446–1455 (2010)
14. Marinakis, Y., Marinaki, M.: A Hybrid Particle Swarm Optimization Algorithm for the Open Vehicle Routing Problem. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 180–187. Springer, Heidelberg (2012)
15. Marinakis, Y., Marinaki, M., Dounias, G.: A Hybrid Particle Swarm Optimization Algorithm for the Vehicle Routing Problem. *Engineering Applications of Artificial Intelligence* 23, 463–472 (2010)
16. Mester, D., Braysy, O.: Active guided evolution strategies for large scale capacitated vehicle routing problems. *Computers and Operations Research* 34, 2964–2975 (2007)
17. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. *Computers and Operations Research* 34, 2403–2435 (2007)
18. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research* 31, 1985–2002 (2004)
19. Prins, C.: A GRASP \times Evolutionary Local Search Hybrid for the Vehicle Routing Problem. In: Pereira, F.B., Tavares, J. (eds.) *Bio-inspired Algorithms for the Vehicle Routing Problem*. SCI, vol. 161, pp. 35–53. Springer, Heideberg (2009)
20. Reimann, M., Doerner, K., Hartl, R.F.: D-Ants: savings based ants divide and conquer the vehicle routing problem. *Computers and Operations Research* 31(4), 563–591 (2004)
21. Rochat, Y., Taillard, E.D.: Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics* 1, 147–167 (1995)
22. Rosendo, M., Pozo, A.: A hybrid Particle Swarm Optimization algorithm for combinatorial optimization problems. In: 2010 IEEE Congress on Evolutionary Computation, CEC (2010), doi:10.1109/CEC.2010.5586178
23. Tarantilis, C.D., Kiranoudis, C.T.: BoneRoute: an adaptive memory-based method for effective fleet management. *Annals of Operations Research* 115(1), 227–241 (2002)
24. Toth, P., Vigo, D.: *The vehicle routing problem*. Monographs on Discrete Mathematics and Applications. SIAM (2002)

Dynamic Evolutionary Membrane Algorithm in Dynamic Environments

Chuang Liu and Min Han*

Faculty of Electronic Information and Electrical Engineering, Dalian University of
Technology, Dalian Liaoning, 116023, China
chuang.liu.cn@gmail.com, minhan@dlut.edu.cn

Abstract. Several problems that we face in real word are dynamic in nature. For solving these problems, a novel dynamic evolutionary algorithm based on membrane computing is proposed. In this paper, the partitioning strategy is employed to divide the search space to improve the search efficiency of the algorithm. Furthermore, the four kinds of evolutionary rules are introduced to maintain the diversity of solutions found by the proposed algorithm. The performance of the proposed algorithm has been evaluated over the standard moving peaks benchmark. The simulation results indicate that the proposed algorithm is feasible and effective for solving dynamic optimization problems.

Keywords: dynamic evolutionary membrane algorithm, membrane computing, diversity, dynamic optimization problem.

1 Introduction

Many real-worlds optimization problems, especially in scientific research and engineering practice, are dynamic. The characteristic of these problems is that the optimal solution is changed over time [1, 2]. More specifically, the objective function and the constraints condition will be changed when a new environment is reached. These optimization problems over time are called dynamic optimization problems (DOPs)[3, 4]. For solving DOPs, the goal of the solving optimization methods is no longer a fixed global optimal solution, but tracking the changing process of the solutions as the environment changes, which brings greater challenges in terms of the methods finding the effective solutions[5]. Recently, the study of solving DOPs has attracted extensive attention in evolutionary computation community. A variety of effective strategies to solve DOPs is proposed in order to improve the ability to track the optimal solutions, and the detailed overview can be found in [6–8].

The evolutionary membrane algorithm (EMA) is a novel optimization method based on the membrane computing for solving the optimization problems [12]. Due to the membrane computing has a multiple layer and nested structure which

* This work was supported by the project (61074096) of the National Nature Science Foundation of China.

a skin membrane contains several membranes [9–11], EMA also inherits the character of the structure of the membrane computing. The structure is conducive to the algorithm in the parallel exploration of the search space, where can accelerate the ability for finding the global optimal solution and enhance the diversity of the candidate solutions. Hence, EMA is suitable for solving the complex optimization problems. EMA has been successfully used to solve static optimization problems, such as the global numerical optimization problems [12], and the multi-objective optimization problems [13], but it has not ability to solve the DOPs.

For solving DOPs, a dynamic evolutionary membrane algorithm based on EMA is proposed, named as dynamic EMA (DEMA). In order to find the global optimal solution of the DOPs, two problems need to be solved: first, how to effectively maintain the diversity of solutions; second, how to exactly predict the new environmental reaching. For solving the first problem, a partitioning strategy and four reaction rules are employed. The partitioning strategy is designed to divide the search space for each membrane having a corresponding search space in DEMA. Four evolutionary rules are used to update the information of the symbol-objects. These rules not only increase the search efficiency of the proposed algorithm, but also improve the diversity of candidate solutions. They enhance the adaptability of the proposed algorithm to solve DOPs. For solving the second problem, a change detection mechanism is designed to update the evolutionary information in the memory so that the new optimal solution is recorded for the current environment. In the experiment, the moving peaks benchmark test function is constructed to verify the ability of the proposed algorithm for solving DOPs. Based on the results of the above experiment, the performance of the proposed DEMA is analyzed.

2 Membrane Computing

Membrane computing is a distributed and parallel computing theory, which is proposed by Păun who takes inspiration of the structure and the function of biological cells [11]. The membrane computing of the degree n is listed in the following Eq. (1).

$$\Pi = (V, T, \mu, w_1, \dots, w_n, R) \quad (1)$$

Where,

1. V is the alphabet. Its elements are called the symbol-objects. The symbol-object is the abstract representation of atomic, molecular or the other chemical substances.
2. $T \subseteq V$, where T is the output alphabet.
3. μ is a membrane structure of the degree n .
4. $w_i \in V^*$, w_i represent the multiset in the i -th region in the membrane structure μ . V^* is a set consisted by the multiple symbol-objects.
5. R represents the reaction rules which can handle the membrane and the symbol-objects.

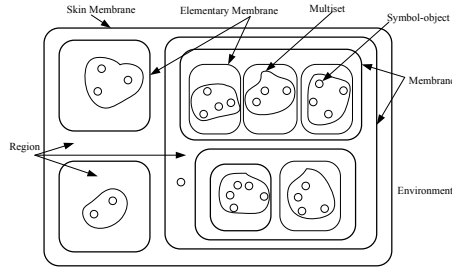


Fig. 1. The generic structure of membrane computing

Fig. 1 shows the generic structure of the membrane computing. The description of some concepts in Fig. 1 is elaborated in the following sections.

The external membrane is usually called a skin membrane. Each membrane determines a compartment, also called a region. And there are several internal membranes in the region of skin membrane, which are corresponded to the membranes and the elementary membrane. If a membrane dose not include any other membrane in its region, it is said to be an elementary membrane. In the basic membrane computing, each region may contain a multiset consisted by the multiple symbol-objects. The symbol-objects are evolved by the evolution rules which are associated with the regions of the membrane.

3 Evolutionary Membrane Algorithm

In EMA, a symbol-object represents a candidate solution of the optimization problems, and a multiset represents candidate solutions of the optimization problems. The symbol-object can be evolved by the reaction rules in the region of the elementary membrane. Each symbol-object has a fitness value related to the optimization problems. Next, a brief description is given to the execution process of EMA: Several multisets are firstly constructed by the symbol-objects in the region of the skin membrane. Then, these multisets are sent into the different elementary membranes, and they are evolved in the region of the elementary membrane. Next, the evolved multisets from the elementary membrane are released into the skin membrane where implement the exchange of the information of the multiset. Finally, after a number of iterations, the symbol-object in the multiset is the global approximate optimal solution in the region of the skin membrane.

3.1 The Structure of EMA

EMA is based on the membrane computing, so it has a multilayer nested structure which a skin membrane contains several membrane. However, in our algorithm DEMA, the structure is simplified to a special two layer structure which a skin membrane contains several elementary membrane. This structure of the

proposed algorithm can be expressed as $[0, [1]_1, [2]_2, \dots, [m]_m]_0$, where, the subscript 0 represents the label of skin membrane; the subscript $1 \dots m$ is the label of the elementary membrane; m is the maximum number of the elementary membrane. For the further understanding of this special structure of the proposed algorithm, Fig. 2 gives a representation of the structure.

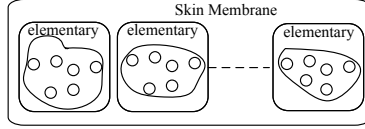


Fig. 2. The simplified membrane structure

In the proposed DEMA, the information of the symbol-objects from the different elementary membrane is shared in the region of the skin membrane, which can maintain the diversity of solutions during the search process. The region of the elementary membrane contains the multiset and the evolutionary rules. The symbol-objects in the mutiset are evolved by the evolutionary rules, which are updated their location and generated their new locations.

3.2 Partition Strategy

The proposed DEMA is composed by the multiple elementary membranes which can be used as the evolving unit to find some optimal solutions of the optimization problems. To speed up the search efficiency and improve the diversity of candidate solutions, the region of each elementary membrane has a corresponding multiset. Hence, a strategy for the partition the search space is proposed to divide the search space of the optimization problem. In the search space, the symbol-object represents a solution, and a multiset denotes a divided search space. All multisets represent the solutions covering the whole search space as much as possible. The strategy is implemented as follows: first, the symbol-objects of the multiset are evaluated using the objective function, and their fitness are calculated. Next, the symbol-objects of the multiset are sorted according to their fitness. Finally, the sorted multiset is divided into the sub-multiset with the same size. The specific process of the partition strategy is described in Eq. (2).

$$\begin{aligned}
 W' &= \text{sort}(W); \\
 W' &= \{w_1, w_2, \dots, w_m\} \\
 w_i &= W'((i - 1) * n + 1 : n : i * n) \\
 n &= \text{sizeof}(W)/m, 1 \leq i \leq m
 \end{aligned} \tag{2}$$

Where, m is the number of the elementary membrane; W is the multiset in the region of the skin membrane; $\text{sizeof}(W)$ represent the number of the symbol-objects in the multiset. W' is the sorted multiset according to the fitness of its

symbol-objects. w_i is the multiset in the i -th elementary membrane. $W'((i-1)*n+1 : n : i*n)$ is a sub-multiset which is constructed by taken n symbol-objects from the sorted multiset with the starting position $(i-1)*n+1$.

3.3 Symbol-Objects and Multiset

In the membrane computing, a symbol-object is the abstract representations of the chemical substances, such as atoms or molecules of the liquid; a multiset is consisted by the multiple symbol-objects. In the proposed DEMA, the symbol-object represents a solution of the optimization problem. And it is encoded in the decimal coding format under the constraint conditions. In other words, the symbol-object can be understood as a decision variable of the optimization problems. The multiset is the solution set of the optimization problems. Its detailed formula is given in Eq.(3).

$$x_{i,j} = \min_j + (\max_j - \min_j) \times \text{rand}() \tag{3}$$

Where, $1 \leq i \leq N$, N is the number of the symbol-objects. $1 \leq j \leq D$, D is the number of dimension of the symbol-objects. $x_{i,j}$ represents the j -th dimension of the i -th symbol-object. \min_j and \max_j denote the minimum value and the maximum value of the j -th dimension, respectively. rand is a uniform function in $(0,1)$.

3.4 Communication Rule

Communication rule implement the exchange information of the multiset from the region of the skin membrane to the elementary membrane. In the region of the skin membrane, the multiple multisets are sent into the different elementary membrane using communication rule after executing the partition strategy. This rule makes the algorithm to find the approximate optimal solutions in the appointed search space. It is described in Eq.(4).

$$[w_1, w_2, \dots, w_m]_0 \rightarrow [[w_1]_1, [w_2]_2, \dots, [w_m]_m]_0 \tag{4}$$

Where, m is the number of the elementary membrane; $[]_0$ represents the skin membrane; $[]_i$ denotes the i -th elementary membrane; w_i is the multiset in the region of the i -th elementary membrane.

4 Dynamic Evolutionary Membrane Algorithm

DEMA is a novel evolutionary algorithm based on EMA for solving the DOPs. Unlike EMA, DEMA has two improvements: redesigning the evolutionary rule and adding the detection of the environment change.

4.1 Evolutionary Rules

In the evolutionary algorithm based on the membrane computing, the position of the symbol-objects is updated in order to allow the algorithm to find the candidate solutions of the optimization problem. The symbol-object (molecular particle) in the region of the membrane does irregular Brownian motion, and Gaussian function can be expressed the Brownian motion process. So the Gaussian distribution function is introduced in the evolutionary rules. Then, four evolutionary rules are designed to describe the process which the symbol-object in the region of membrane is randomly affected from the different direction force. Fig. 3 describes the pseudo-code of invoking reaction rules.

Invoking evolutionary rule
Switch(rand)/[1,6]
Case 1: invoking rule 1; break;
Case 2: invoking rule 2; break;
Case 3: invoking rule 3; break;
Case 4: invoking rule 4; break;
End

Fig. 3. The pseudo-code of invoking reaction rules

Evolutionary rule 1 may implement the exchange information of the current symbol-objects with the local best one in the elementary membrane. The rule may accelerate the local convergence speed of the proposed algorithm.

$$x_{i,j} = x_{i,j} + (x_{i,j} - x_j^l) \cdot N(0, 0.2) \quad (5)$$

Where, $x_{i,j}$ is the j -th dimension of the i -th symbol-object in the region of the elementary membrane. x_j^l is the j -th dimension of the local best symbol-object in the region of the elementary membrane. $N(0, 0.2)$ represents a Gaussian function with mean of 0 and variance of 0.2.

Taken inspiration by arithmetic crossover in GA [15], evolutionary rules 2 and 3 are designed to implement the exchange information of the current symbol-objects with the global best one in the elementary membrane. The rule 2 enhances the diversity of solutions during the search process.

$$x_{i,j} = r \cdot x_{i,j} + (1 - r) \cdot x_j^g \quad (6)$$

Where, $x_{i,j}$ is the j -th dimension of the i -th symbol-object in the region of the elementary membrane. x_j^g is the j -th dimension of the global best symbol-object in the region of the skin membrane. r denotes a uniform function in (0,1).

Evolutionary rule 3 may implement the exchange information of the current symbol-objects with the global best one in the elementary membrane. The rule increases the diversity of solutions during the search process.

$$x_{i,j} = (1 - r) \cdot x_{i,j} + r \cdot x_j^g \quad (7)$$

Where, $x_{i,j}$ is the j -th dimension of the i -th symbol-object in the region of the elementary membrane. x_j^g is the j -th dimension of the global best symbol-object in the region of the skin membrane. r denotes a uniform function in $(0,1)$.

Evolutionary rule 4 may the exchange information of the current symbol-objects with the global best one in the elementary membrane. The rule may accelerate the global convergence performance.

$$x_{i,j} = x_{i,j} + (x_{i,j} - x_j^g) \cdot N(0, 0.2) \tag{8}$$

Where, $x_{i,j}$ is the j -th dimension of the i -th symbol-object in the region of the elementary membrane. x_j^g is the j -th dimension of the global best symbol-object in the region of the skin membrane. $N(0,0.2)$ represents a Gaussian function with mean of 0 and variance of 0.2.

4.2 Detection of the Environmental Change

The environment changes over time, which causes the objective function, the decision variables and the constraint functions to change. The information in the memory may be no longer the optimal solutions of the current environmental problems. In other words, the information can not correctly guide the algorithm to find the optimal solutions. To solve this problem, the change detection mechanism is needed. Once the environment changes, the information need to be immediately updated; otherwise, this outdated information will affect the implementation efficiency of the algorithm. Fig. 4 describes the pseudo-code of detecting environment change. The environmental change is detected by re-evaluate the fitness value of the symbol-objects. If the environment changes over time, the symbol-objects in the multiset need to be re-evaluated and updated the historical information in the memory. At last, the global symbol-object needs to be found according to their fitness.

```

Detecting the environmental change
If Detect the environmental change== TRUE
    fit=evaluate(SymbolObject);
End
If SymbolObject.fit != fit
    evaluate(multiset);
    find(SymbolObjectgbest);
End
    
```

Fig. 4. The detection of the environmental change

5 Experiments

To verify the performance and effectiveness of the proposed DEMA, the dynamic moving peak benchmark (MPB) test functions are employed [14]. This test function in a multidimensional space is defined by periodically changing the

Table 1. The parameter setting of MPB

Parameter	values
Number of Peaks	10
Change frequency	5000
High Severity	7
Width Severity	1
Height	50
Moving length	1.0
dimension	30, 50
Search Range	[0,100]
Height Range	[30,70]
Width Range	[1,12]

peak height, width and position. The default parameters of the MPB are shown in Table 1.

Due to the optimal solution changes over time in the DOPs, so the offline indicator is designed to evaluate the performance of the dynamic evolutionary algorithms. Its formula is described in Eq.(9).

$$of\ flineError = \frac{1}{FEs/CF} \sum_{t=1}^{FEs} (f(bestSolution(t)) - f(globalOptimum(t))) \quad (9)$$

Where, FEs is the total number of the fitness evaluation. CF is the change frequency. $bestSolution(t)$ is the best position found by the solving method in the t -th iteration. $globalOptimum(t)$ represents the global optimal value in the t -th iteration.

The values of $mean$, std and $performance_k$ are calculated according to formula (10), (11) and (12). The sum of all marks $performance_k$ gives a score, denoted by $performance$, which corresponds to the overall performance of the tested algorithm.

$$mean = \sum_{i=1}^{runs} \sum_{j=1}^{num_change} E_{i,j}^{last}(t) / (runs \cdot num_change) \quad (10)$$

$$std = \sqrt{\frac{1}{runs \cdot num_change} \sum_{i=1}^{runs} \sum_{j=1}^{num_change} (E_{i,j}^{last}(t) - mean)^2} \quad (11)$$

$$performance_k = \sum_{i=1}^{runs} \sum_{j=1}^{num_change} r_{i,j} / (num_change \cdot runs) \quad (12)$$

$$performance = \sum_{k=1}^N performance_k \quad (13)$$

$$E^{last}(t) = |f(x^b(t)) - f(x^*(t))| \quad (14)$$

5.1 Experimental Environment

The experiment will be run on the hardware environment of the Intel Pentium dual-core 2.93 GHz and 2G memory, and on the operating system of Windows XP. Based on the EALib package¹, the proposed DEMA is implemented in c++ language. It is initialized according to Eq.(15).

$$\Pi = \left\{ \begin{array}{l} \{x_{i,j}, 1 \leq i \leq 5, 1 \leq j \leq 10\}, \\ [0[1]_1, [2]_2, \dots, [5]_5]_0, \\ w_1, \dots, w_5, \\ rule_1, rule_2, rule_3, rule_4, \\ [w_i]_0 \rightarrow [[w_i]_i]_0, \\ [[w_i]_i]_0 \rightarrow [w_i]_0, 1 \leq i \leq 5 \end{array} \right\} \tag{15}$$

Where, Π denotes a membrane system; $x_{i,j}$ denotes the j -th symbol-object in the i -th elementary membrane; $[0[1]_1, [2]_2, \dots, [5]_5]_0$ denotes the skin membrane which is marked as zero and contains five elementary membranes; w_1, \dots, w_5 denotes the multiset labeled from 1 to 5, which are combined of ten symbol-objects. The $rule_1, rule_2, rule_3, rule_4$ denote four evolutionary rules. The $[w_i]_0 \rightarrow [[w_i]_i]_0$ denotes a communication rule from the skin membrane to the elementary membrane. The $[[w_i]_i]_0 \rightarrow [w_i]_0$ is also a communication rule from the elementary membrane to the skin membrane.

5.2 Experimental Results

For solving MPB with the peak of 10, CPSO [16], CDER [17], ESCA [18] and the proposed algorithm are employed, respectively. The simulation experiments independently run 100 times, and the *mean* and the *std* are recorded, respectively. Fig. 5 and Fig. 6 depict the relationship between the number of iterations and the offline error in MPB with the dimensions of 30 and 50. Table 2 and Table 3 give the statistical results of the different optimization algorithms after 100 independent run.

Fig. 5 and Table2 show the experimental results of the proposed algorithm in comparison with other algorithms in the peak of 10 and the dimension of 30. As is seen in Fig. 5, the proposed algorithm is superior to ESCA, CDER and CPSO. Compared with ESCA and CDER, CPSO has a good result.

The *Mean*, *Std* are calculated according to Eq.(10 and 11). And, the *Best* and the *Worst* represent respectively the best absolute error and the worst absolute error in 100 independent run in Table 2 and Table 3. The *Performance* is calculated according to Eq.(12). The statical results obtained by DEMA, CPSO, CDER, and ESCA for under the same conditions are given in Table 2. DEMA produced the best results in comparison with CPSO, CDER and ESCA. Even though CDER find the best value of the MPB, the *Mean* and *Std* indicate the proposed DEMA is robust and stable. The performance value also proved further the effective of EMA.

¹ <http://cs.cug.edu.cn/teacherweb/lichanghe/pages/EALib.html>

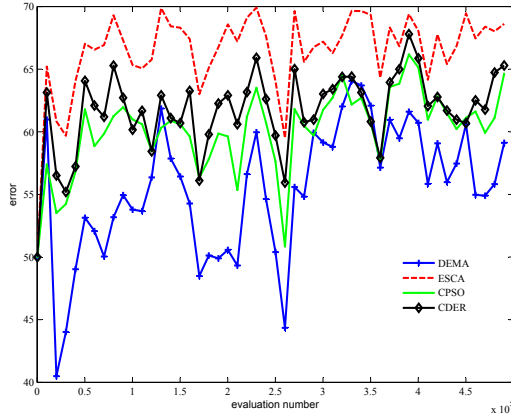


Fig. 5. The experimental results of MPB on 30 dimensions

Table 2. The experimental results of MPB on 30 dimensions

	Mean	Std	Best	Worst	Performance
DEMA	2.476e+01	8.224e+00	8.348e-01	5.755e+01	4.610e-01
CPSO [16]	4.761e+01	4.942e+00	1.716e+01	6.594e+01	1.981e-01
CDER [17]	3.809e+01	1.960e+01	2.137e-01	6.891e+01	2.963e-01
ESCA [18]	6.618e+01	2.449e-03	4.998e+01	6.991e+01	7.618e-03

Fig. 6 and Table 3 show the experimental results of the proposed algorithm in comparison with other algorithms in the peak of 10 and the dimension of 50. As is seen in Fig. 6, the proposed DEMA has obvious advantages in comparison with ESCA, CDER and CPSO. The results of ESCA, CDER and CPSO are similar.

The results of Table 3 on 50 dimension show the DEMA has the best performance in comparison with CPSO, CDER and ESCA. The mean and the STD indicate the proposed DEMA is robust and stable. The performance value shows the proposed DEMA is better than other competitors. Compared with CPSO and ESCA, CDER has a good result.

From the above simulation experiments, the overall performance of DEMA is better than the other compared algorithms. This indicates that DEMA has some advantage in terms of solving moving peak problems. These advantages are mainly reflected in the proposed DEMA using the membrane structure and the effectiveness of evolutionary rules, which can avoid the algorithm converge to a certain local extreme points and maintain good diversity of the candidate solutions during the search process. The experimental results indicate the proposed algorithm is effective in moving peak problems.

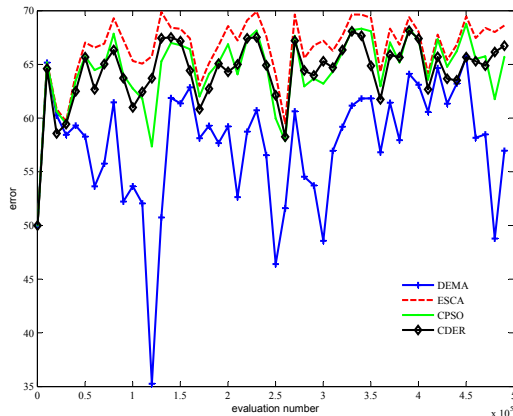


Fig. 6. The experimental results of MPB on 50 dimensions

Table 3. The experimental results of MPB on 50 dimensions

	Mean	STD	Best	Worst	Performance
DEMA	4.483e+01	3.388e+00	1.002e+01	6.687e+01	2.481e-01
CPSO [16]	6.263e+01	9.717e-01	4.281e+01	6.869e+01	4.920e-02
CDER [17]	5.894e+01	1.086e+01	2.561e+00	6.991e+01	8.301e-02
ESCA [18]	6.618e+01	6.146e-04	4.999e+01	6.983e+01	7.563e-03

6 Conclusions

Based on the membrane computing, a novel dynamic evolutionary membrane algorithm is proposed for solving the dynamic optimization problem. In DEMA, the partitioning strategy is employed to divide the search space to improve the search efficiency of the algorithm. In addition, four evolutionary rules are designed to maintain the diversity of the candidate solutions. Simulation experiments give the performance of the proposed DEMA in comparison with the existing algorithms. From the experimental results, the proposed algorithm achieved the desired results.

In the future, there are several interesting areas to pursue. The parameters of DEMA are the key to affect the performance of the algorithm, while the precisely determining, such as the number of elementary membrane, and the number of the symbol-objects, is very difficult. In addition, the evolutionary rules need be improved to balance between explorative and exploitative.

References

1. Liu, L., Yang, S., Wang, D.: Particle swarm optimization with composite particles in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 40, 1634–1648 (2010)

2. Wang, H., Yang, S., Ip, W.H., Wang, D.: A memetic particle swarm optimisation algorithm for dynamic multi-modal optimisation problems. *International Journal of Systems Science* 43, 1268–1283 (2012)
3. Brest, J., Korošec, P., Šilc, J., Zamuda, A., Boškovic, B., Maučec, M.S.: Differential evolution and differential ant-stigmergy on dynamic optimisation problems. *International Journal of Systems Science*, 1–17 (2011)
4. Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation* 12, 542–561 (2008)
5. Korosec, P., Silc, J.: The Continuous Differential Ant-Stigmergy Algorithm Applied to Dynamic Optimization Problems. In: *Proceedings of the 2012 Congress on Evolutionary Computation*, pp. 1317–1324. IEEE Press, New York (2012)
6. Nguyen, T.T., Yang, S., Branke, J.: Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation* 6, 1–24 (2012)
7. Cruz, C., Gonzalez, J.R., Pelta, D.A.: Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing-A Fusion of Foundations, Methodologies and Applications* 15, 1427–1448 (2011)
8. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation* 9, 303–317 (2005)
9. Păun, G., Rozenberg, G.: A guide to membrane computing. *Theoretical Computer Science* 287, 73–100 (2002)
10. Păun, G.: Computing with Membranes. *Journal of Computer and System Sciences* 61, 108–143 (2000)
11. Păun, G., Rozenberg, G., Salomaa, A.: *The Oxford handbook of membrane computing*. Oxford University Press (2010)
12. Liu, C., Han, M., Wang, X.: A novel evolutionary membrane algorithm for global numerical optimization. In: *2012 Third International Conference on Intelligent Control and Information Processing (ICICIP)*, pp. 727–732 (2012)
13. Liu, C., Han, M., Wang, X.: A Multi-Objective Evolutionary Algorithm based on Membrane Systems. In: *2011 Fourth International Workshop on Advanced Computational Intelligence (IWACI)*, pp. 103–109 (2011)
14. Blackwell, T., Branke, J.: Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation* 10, 459–472 (2006)
15. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd edn. Springer, Berlin (1994)
16. Yang, S., Li, C.: A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Transactions on Evolutionary Computation* 14, 959–974 (2010)
17. Li, C., Yang, S.: A general framework of multi-population methods with clustering in undetectable dynamic environments. *IEEE Transactions on Evolutionary Computation* 16, 556–577 (2012)
18. Lung, R.I., Dumitrescu, D.: Evolutionary swarm cooperative optimization in dynamic environments. *Natural Computing* 9, 83–94 (2010)
19. Aragón, V.S., Esquivel, S.C., Coello Coello, C.A.: A T-cell algorithm for solving dynamic optimization problems. *Information Sciences* 181, 3614–3637 (2011)

From Sequential to Parallel Local Search for SAT

Alejandro Arbelaez^{1,*} and Philippe Codognet²

¹ JFLI / University of Tokyo

² JFLI - CNRS / UPMC / University of Tokyo
University of Tokyo, Dept. of Computer Science,
7-3-1, Hongo, Bunkyo-ku, 113-0033 Tokyo, Japan
{arbelaez,codognet}@is.s.u-tokyo.ac.jp

Abstract. In the domain of propositional Satisfiability Problem (SAT), parallel portfolio-based algorithms have become a standard methodology for both complete and incomplete solvers. In this methodology several algorithms explore the search space in parallel, either independently or cooperatively with some communication between the solvers. We conducted a study of the scalability of several SAT solvers in different application domains (crafted, verification, quasigroups and random instances) when drastically increasing the number of cores in the portfolio, up to 512 cores. Our experiments show that on different problem families the behaviors of different solvers vary greatly. We present an empirical study that suggests that the best sequential solver is not necessarily the one with the overall best parallel speedup.

1 Introduction

The propositional Satisfiability Problem (SAT) is the first known NP-complete problem [1] and consists in determining whether a Boolean formula \mathcal{F} is satisfiable or not. \mathcal{F} is represented by a pair $\langle \mathcal{X}, \mathcal{C} \rangle$, where \mathcal{X} is a set of Boolean variables and \mathcal{C} is a set of clauses in Conjunctive Normal Form (CNF). Each clause is a disjunction of literals (a variable x or its negation $-x$). Additionally, an assignment is a mapping from the variables in the problem to truth values, i.e. 0 (false) or 1 (true).

SAT solvers are able to tackle instances from a wide variety of domains ranging from software verification to computational biology and automated planning. As these are very hard and computationally intensive problems, the use of parallelism to speed up SAT solvers has attracted the attention of a growing number of researchers in the last decade. Currently, there are two well-established techniques to develop parallel SAT solvers: divide-and-conquer (e.g. [2]) and parallel portfolios (e.g. [3] and [4]). The former, divides the search space into several sub-spaces being explored in parallel. The second one consists in parallel executions of different algorithms, either independently or cooperatively with some communication between the parallel solvers.

* Corresponding author.

The performance of parallel SAT solvers is classically measured by means of the overall number of solved instances, as it is for sequential solvers. That is, increasing the number of cores might also increase the total number of solved instances within the same wall-clock time. This is what is usually called *capacity solving*. For instance, in the annual SAT competition solvers are ranked based on the total number of solved instances within a given time limit (breaking ties using the wall-clock time). However, another interesting feature is the *scalability* of the parallel solvers, that is, the *speedup* obtained when using several cores with respect to sequential execution. But the notion of *speedup* obtained for parallel solvers has received up to now limited attention in the classical SAT literature, which was mostly focused on *capacity solving*.

The question we would like to investigate in this paper is: *Is the best sequential solver also the best one in a massively parallel context?* Indeed the best sequential solver might not scale up so well and its performance on n cores (for some large n) might not be better than that of another solver whose sequential performance is maybe less good but with a better parallel scalability.

In order to study this issue, we conducted a study of the scalability (w.r.t. the speedup of the algorithms) of SAT solvers in different application domains when drastically increasing the number of cores in the portfolio. We focused our attention in this paper on local search algorithms for SAT. In SAT local search there exist a large number of heuristics [5] with different performances for different problem families. For this reason, unlike sequential settings, where it is necessary to select only one algorithm to solve a given problem instance, the parallel portfolio needs to identify n solvers. Moreover, in addition of selecting solvers, the parallel portfolio might also consider their scalability when increasing the number of cores.

The rest of the paper is organized as follows. Section 2 presents background material including a description of local search algorithms for SAT and related work in the area. Section 3 describes all the benchmark families used in this paper, Section 4 presents extensive experimental results on parallel portfolios, and Section 5 presents general conclusions and future work.

2 Background

2.1 Local Search

A local search algorithm to tackle SAT instances starts with an initial random assignment for the variables (i.e. random values for the variables), then iteratively flips the truth value of one of the variables. The flipped variable usually minimizes the number of unsatisfied clauses, however, from time to time random selections are performed in order to avoid search stagnation.

In the following, we describe five well-known variable selection algorithms in local search for SAT. These algorithms have shown great performances in the annual SAT competitions. It is important to notice, that most of SAT local search algorithms have been inspired by the WalkSAT architecture [6], which selects an unsatisfied clause and from that clause identifies the most appropriated variable

to flip using some heuristic function. In the following, we describe a set of state-of-the-art local search algorithms to solve SAT instances.

Novelty [7] employs a function $score(x) = make + break$ to select a variable at each iteration of the local search procedure. Intuitively, *make* indicates the number of clauses that are satisfied under the current assignment but become unsatisfiable when flipping x , and *break* represents the number of clauses that are unsatisfiable under the current assignment and will be satisfied when flipping x . Then, *Novelty* selects, uniformly at random, an unsatisfied clause c , and from c identifies v_{best} and v_{2best} , the best and the second best variables in c according to the *score* function. v_{best} is flipped if this variable is not the most recently flipped one in c , otherwise v_{2best} is flipped with a given probability p and v_{best} with a probability $1-p$.

Pure Additive Weighting Scheme (PAWS) [8] adds a weight clause penalty to each clause and selects the variable that provides the highest reduction in the sum of all unsatisfied clause penalties. All weights are initialized to 1 and updated during the search process, i.e. increased 1 unit when search stagnation is observed and decreased after a given number of weight increases.

Variable Weighting (VW) [9] maintains a counter for each variable, indicating the number of flips of the variable. Then, VW (known as VW1 in [9]) selects an unsatisfied clause c and if no variable in c reports $break=0$, VW selects with a probability p a random variable in c . Otherwise, the variable with the smallest *break* value is selected, breaking ties by minimizing the number of flips of the variables.

Adaptive G2WSAT (AG2) [10]: introduces the concept of decreasing variables. Broadly speaking, a variable is decreasing if flipping it reduces the over all number of failed clauses. Taking this into account, the algorithm maintains a lists of promising decreasing variables L and selects the variable with minimal score in L . If L is empty, AG2 selects, with a probability dp the most recently flipped variable from a violated clause. Otherwise, with a probability $1-dp$ *Novelty* is used as a backup heuristic.

Sparrow [11] exploits features of the previously mentioned local search algorithms. First, similar to PAWS a weight penalty is added to all clauses. Second, similar to AG2 a list of promising variables L is maintained during the search. Whenever L is empty the penalty for unsatisfied clauses is increased 1 unit with a probability ps and decreased with probability $1-ps$. Sparrow selects the best variable from L , and if L is empty the algorithm selects, uniformly at random, an unsatisfied clause c and from c selects a variable using a probabilistic function which considers two criteria: the sum of all unsatisfied clauses and the last time the variable was last flipped.

2.2 Parallel SAT

A straightforward approach to parallelize local search algorithms consists in the parallel portfolio-based approach (so-called multi-start or multiple-walk). In this

approach, several algorithms (or different copies of the same one with different random seeds) are executed in parallel until a solution is found or a given timeout is reached.

The parallel portfolio has two important properties. First, no load balancing is required to parallelize the sequential algorithm. Second, in theory, it is possible to reach linear and super-linear speedups [12]. Indeed, in Section 4 we will observe that in practice some scenarios report super-linear speedups.

The portfolio approach without cooperation has been previously used in the gNovelty+ solver [13]. This portfolio executes independent copies of the gNovelty+ heuristic. Other parallel local search solvers for SAT comprehend PGSAT [14] and MiniWalk [15]. PGSAT divides the entire set of variables into independent subsets which are then allocated to different processors, then iteratively performs multiples flips in parallel (one for each subset). MiniWalk combines a complete solver (MiniSAT) and an incomplete one (WalkSAT). Broadly speaking, both solvers are launched in parallel and MiniSAT is used to guide WalkSAT by suggesting values for the selected variables.

Other work in the area includes [4], where the authors use cooperation to improve the performance of the parallel portfolio. In this work, each algorithm exchanges the best assignment for the variables found so far in order to properly craft a new assignment for the variables to restart from. These strategies range from a voting mechanism where each algorithm in the portfolio suggests a value for each variable to probabilistic constructions. However, as pointed out in [16] the performance of the cooperative portfolio considerably degrades as the number of cores increases.

Regarding complete parallel SAT solvers, several multicore algorithms (see [17] for a recent survey) have been proposed. Most of these algorithms are also based on the parallel portfolio architecture. However, in this case, the portfolio executes different and complementary backtracking search algorithms based on the DPLL method. Moreover, algorithms exchange learned clauses in order to improve performance.

3 Experimental Settings

This section describes the set of benchmark families used to test the performance of the algorithms. That is, crafted, quasigroups, verification, and random instances.

All the experiments were performed on the Grid'5000 platform, the French national grid for research, in particular we used a 44-node cluster with 24 cores (2 AMD Opteron 6164 HE processors at 1.7 Ghz) and 44 GB of RAM per node.

In addition, we used openMPI to build our parallel solver on top of UBCSAT [18]. When running on n cores, each parallel portfolio executes n independent copies of a given algorithm. Moreover, all algorithms were executed with their default parameters, except for Sparrow where we use the parameter configuration reported for the international SAT'11 competition.

3.1 Crafted Instances

This problem family corresponds to a selection of instances designed to be difficult for SAT solvers. In this paper, we used a set of 149 known SAT instances from the 2011 SAT competition (crafted category) and filtered out too easy and hard instances by running a portfolio of 16 copies of Sparrow with a timeout of 5 minutes. We use all instances whose median runtime across 10 runs was greater than 100 seconds and lower than 300 seconds. The final set consists in the following 9 instances (denoted crafted-[1 to 9] in this paper): srhd-sgi-m37-q505.75-n35-p15-s48276711 (crafted-1); srhd-sgi-m42-q585-n40-p15-s54275047 (crafted-2); srhd-sgi-m42-q663-n40-p15-s72490337 (crafted-3); srhd-sgi-m47-q742.5-n45-p15-s28972035 (crafted-4); em_8.4_5_fbc (crafted-5); rbsat-v1150c84314g7 (crafted-6); rbsat-v1375c111739g4 (crafted-7); sgen3-n240-s78945233-sat (crafted-8); sgen3-n260-s62321009-sat (crafted-9).

3.2 Quasigroup Instances

The Quasigroup with holes problem (qwh) consists in completing a pre-filled $N \times N$ matrix with the numbers $[1, 2, \dots, N]$ such that for each column (resp. row) of the matrix, each element occurs exactly once. Instances were generated using the *lsencode* instance generator [19]. It is also worth to notice that these instances have been widely used to test the performance of SAT and CSP solvers. The final set consists in the following 8 instances (denoted qwh-[1 to 8] in this paper): qwh.order.35.holes.405 (qwh-1); qwh.order.40.holes.528 (qwh-2); qwh.order.40.holes.544 (qwh-3); qwh.order.40.holes.560 (qwh-4); qwh.order.60.holes.1440 (qwh-5); qwh.order.60.holes.1620 (qwh-6); qwh.order.70.holes.2450 (qwh-7); qwh.order.70.holes.2940 (qwh-8).

3.3 Verification Instances

This problem family corresponds to a collection of SAT encoded CBMC (Bounded Model Checking) instances generated using [20]. In this paper, we used a set of 40 instances also used to test SAT solvers, such as [21]. We filtered out too easy and hard instances by running a portfolio of 16 copies of Sparrow with a timeout of 5 min. and selected all the instances whose median runtime across 10 runs was greater than 100 sec. and lower than 5 min. The final set of instances is the following (as named in [21] and denoted cbmc-[1 to 9] in this paper): 23 (cbmc-1); 25 (cbmc-2); 26 (cbmc-3); 28 (cbmc-4); 31 (cbmc-5); 32 (cbmc-6); 33 (cbmc-7); 35 (cbmc-8); 36 (cbmc-9).

3.4 Random Instances

Random instances (also known as Uniform Random k -SAT) are frequently used to test the performance of SAT solvers. These instances are automatically generated using three parameters: number of clauses (m), number of variables (n),

and the number of literals per clause (k). Clauses for a given instance are generated by iteratively selecting, uniformly at random, a variable id i and then with a probability $0.5 x_i$ is included into the clause, otherwise $-x_i$ is added to the clause (literals of different polarity are not accepted in the same clause). It is worth pointing out that random k -SAT instances around the phase transition (i.e. $m/n=4.2$) are known to be difficult [22].

In this paper, we consider a collection of 369 known satisfiable instances from the international SAT'11 competition. From this set we filtered out too easy and too hard instances by running a portfolio of 16 copies of Sparrow and removed instances whose median runtime were greater than 100 sec. and lower than 300 sec.. The final set consists in the following 8 instances (denoted rand-[1 to 8] in this paper), where Seed indicates the unique seed number used to generate the instance, v represents the number of variables, and r represents the ratio variables/clauses: Seed: 1854039067 - v: 30000 - c: 126000 - r: 4.2 (rand-1); Seed: 970100151 - v: 35000 - c: 147000 - r: 4.2 (rand-2); Seed: 1184456903 - v: 40000 - c: 168000 - r: 4.2 (rand-3); Seed: 1170024351 - v: 50000 - c: 210000 - r: 4.2 (rand-4); Seed: 537193780 - v: 50000 - c: 210000 - r: 4.2 (rand-5); Seed: 957916968 - v: 50000 - c: 210000 - r: 4.2 (rand-6); Seed: 969405384 - v: 1500 - c: 30000 - r: 20 (rand-7); Seed: 922811046 - v: 2000 - c: 30000 - r: 20 (rand-8).

4 Experiments

In this section, we present experiments of parallel portfolios when drastically increasing the number of cores. For the sake of clarity, we use the following notation: [Solver Name]- N , where N represents the number of cores, for example: Sparrow-128, AG2-512, and VW-32 represent respectively a portfolio of Sparrow on 128 cores, AG2 on 512 cores, and VW on 32 cores. Each core executing one copy of the indicated algorithm.

In addition, the speedup is reported against a portfolio of 16 cores and computed as follows: $Speedup = \frac{median-time([Solver]-N)}{median-time([Solver]-16)}$, where *median-time* reports the median time across 50 independent executions of a given portfolio strategy.

Moreover, we also study the runtime distribution (RTD) for each benchmark family (see chapter 4 in [5]). The RTD is a probability function $P(timeout < t)$ which assigns probabilities to a given random variable, i.e. the runtime needed until completion, and can be seen as the probability of solving a given instance within a given time limit t .

4.1 Crafted Instances

Let us start our analysis with Figure 1(a), where we observe the overall performance improvement when increasing the number of cores from 16 to 512. This figure shows the RTD for Sparrow-16, AG2-16, Sparrow-512, and AG2-512. As one might have expected, increasing the number cores also increases the chances of solving a given instance for this problem family. For instance, the probability of solving an instance with a time limit of 10 seconds for Sparrow increases from

Table 1. Performance summary for crafted instances. Each cell indicates the median runtime (top) and the percentage of solved instances (bottom) for each instance.

Instance	Alg	Number of Cores					
		16	32	64	128	256	512
crafted-1	SP	145.3 100%	90.7 100%	68.2 100%	23.8 100%	10.7 100%	5.9 100%
	AG2	27.0 100%	13.8 100%	4.5 100%	2.2 100%	0.9 100%	0.8 100%
crafted-2	SP	122.6 100%	73.6 100%	47.5 100%	16.2 100%	9.6 100%	5.4 100%
	AG2	48.7 100%	15.0 100%	7.3 100%	4.2 100%	1.6 100%	1.1 100%
crafted-3	SP	146.6 100%	95.6 100%	60.9 100%	29.9 100%	11.3 100%	6.7 100%
	AG2	51.9 100%	23.3 100%	6.0 100%	2.9 100%	1.9 100%	1.2 100%
crafted-4	SP	150.3 100%	63.8 100%	47.7 100%	22.7 100%	10.9 100%	4.6 100%
	AG2	30.7 100%	17.9 100%	5.9 100%	3.5 100%	2.3 100%	1.9 100%
crafted-5	SP	129.0 100%	54.7 100%	27.2 100%	14.0 100%	8.8 100%	5.8 100%
	AG2	742.8 96%	428.6 100%	224.5 100%	105.4 100%	50.3 100%	38.5 100%

Instance	Alg	Number of Cores					
		16	32	64	128	256	512
crafted-6	SP	278.2 100%	141.3 100%	41.7 100%	30.0 100%	14.8 100%	11.5 100%
	AG2	123.6 100%	71.3 100%	30.9 100%	18.2 100%	11.1 100%	6.5 100%
crafted-7	SP	167.3 100%	89.4 100%	63.7 100%	32.1 100%	16.4 100%	7.3 100%
	AG2	70.8 100%	41.2 100%	18.6 100%	10.2 100%	4.8 100%	3.0 100%
crafted-8	SP	111.3 100%	52.4 100%	28.8 100%	23.9 100%	12.2 100%	8.8 100%
	AG2	46.5 100%	32.2 100%	13.2 100%	7.6 100%	6.3 100%	3.6 100%
craft-9	SP	125.7 100%	75.7 100%	44.6 100%	23.2 100%	13.5 100%	10.6 100%
	AG2	79.89 100%	42.64 100%	22.79 100%	13.45 100%	9.69 100%	7.32 100%

about $P(\text{timeout} < 10) \approx 0.03$ using 16 cores to $P(\text{timeout} < 10) \approx 0.69$ using 512 cores.

Additionally, Figure 2(a) shows the speedup (relative to a portfolio using 16 cores) using 512 cores for each algorithm. Algorithms above the dashed line indicate that a super-linear speedup is reached and below the line indicate a sub-linear speedup. In this figure, we observe linear speedups for the following instances: crafted-1 (AG2-512), crafted-2 (AG2-512), crafted-3 (AG2-512), and crafted-4 (Sparrow-512). Moreover, except for crafted-8, crafted-9, and crafted-4 (AG-512), we observe an interesting speedup for all algorithms, this shows the scalability of the parallel portfolio approach when considering an important number of cores. It is also worth noticing that the speedup observed for both algorithms (up to 512 cores) is similar for nearly all instances.

Table 1 summarizes the results for each portfolio using 16, 32, 64, 128, 256, and 512 cores¹. Hereafter, bold figures indicate statistically significant differences (Mann-Whitney U test with 95% confidence level). It can be observed that AG2-16 is considerably better than Sparrow-16, however, the difference in the performances becomes smaller as the number of cores increases. In addition, to solve crafted-5, the effectiveness (i.e. percentage of solved instances) of AG2 increases as the number of cores increases, i.e. from 96% (AG2-16) to 100% (AG2-32).

4.2 Quasigroup Instances

Figure 1(c) shows the RTD for Sparrow and AG2 for QWH problems. Here, we observe that overall Sparrow is better than AG2 for QWH instances using 16 cores. Notice that the probability of solving a given instance within 10 seconds for Sparrow-512 and AG2-512 is $P(\text{timeout} < 10) > 0.9$.

Unlike crafted instances where both algorithms exhibit a good speedup for nearly all instances up 512 cores, here only AG2 exhibits an interesting speedup

¹ In the following tables 'SP' stands for Sparrow.

Table 2. Performance summary for QWH instances. Each cell indicates the median runtime (top) and the percentage of solved instances (bottom) for each instance.

Instance	Alg	Number of Cores					
		16	32	64	128	256	512
qwh-1	SP	1.90 100%	1.26 100%	1.26 100%	0.95 100%	0.70 100%	0.81 100%
	AG2	104.07 100%	63.06 100%	39.54 100%	18.82 100%	9.40 100%	4.94 100%
qwh-2	SP	1.33 100%	0.80 100%	0.67 100%	0.57 100%	0.55 100%	0.51 100%
	AG2	101.81 100%	33.02 100%	22.40 100%	9.68 100%	6.16 100%	4.11 100%
qwh-3	SP	1.02 100%	0.69 100%	0.57 100%	0.53 100%	0.44 100%	0.47 100%
	AG2	29.16 100%	17.87 100%	10.11 100%	8.86 100%	4.46 100%	2.70 100%
qwh-4	SP	0.78 100%	0.57 100%	0.37 100%	0.34 100%	0.34 100%	0.36 100%
	AG2	5.02 100%	3.59 100%	1.83 100%	0.93 100%	0.54 100%	0.57 100%

Instance	Alg	Number of Cores					
		16	32	64	128	256	512
qwh-5	SP	20.98 100%	10.61 100%	4.77 100%	2.89 100%	2.01 100%	1.99 100%
	AG2	0.72 100%	0.71 100%	0.71 100%	0.72 100%	0.75 100%	0.81 100%
qwh-6	SP	6.15 100%	3.91 100%	2.81 100%	1.78 100%	1.61 100%	1.60 100%
	AG2	1.11 100%	1.12 100%	1.13 100%	1.14 100%	1.16 100%	1.22 100%
qwh-7	SP	20.15 100%	18.06 100%	11.18 100%	7.47 100%	5.70 100%	4.99 100%
	AG2	4.05 100%	4.04 100%	4.09 100%	4.09 100%	4.16 100%	4.29 100%
qwh-8	SP	17.88 100%	12.87 100%	12.32 100%	12.10 100%	12.13 100%	12.37 100%
	AG2	11.25 100%	11.15 100%	11.29 100%	11.23 100%	11.49 100%	11.57 100%

for four instances, i.e. qwh-[1-4]. Additionally, we would like to point out that in Table 2, it can be observed that AG2 (qwh-1) exhibits a super-linear speedup up to 256 cores. More importantly, due to the great scalability of the portfolio-base approach the difference in the performance between the algorithms decreases as the number of cores increases. For instance, to solve qwh-1, Sparrow-16 is 54 times faster than AG2-16, while Sparrow-512 is only 6 times faster than AG2-512 to solve the same instance.

4.3 Verification Instances

For this set of instances we limit our attention to our reference solver Sparrow and VW. VW has been reported in the literature as a very efficient algorithm for this set of problems (see [21]) Figure 1(b) shows an important difference between VW-16 and Sparrow-16. The difference lies primarily in the probability of solving a given instance within the time limit, i.e. VW-16 reports $P(\text{timeout} < 3600) \approx 0.96$ while Sparrow-16 reports $P(\text{timeout} < 3600) \approx 0.82$. Both algorithms exhibit an improvement when the number of cores increases (from 16 to 512), $P(\text{timeout} < 3600) \approx 1$ in both cases.

Figure 2(b) shows that Sparrow achieves a near optimal speedup for 5 out of 9 instances (cbmc-[2,3,5,7,8]), and VW achieves a near optimal speedup for cbmc-7. Moreover, Table 3 shows the benefit of increasing the number of cores. For instance, the effectiveness of Sparrow (crafted-8) gradually increases as the number cores increases from 44% to 100%, i.e. 44% (Sparrow-16); 77% (Sparrow-32); 92% (Sparrow-64); and 100% (Sparrow-128). Whereas, the effectiveness of VW (crafted-7) increases from 90% to 100%, i.e. 90% (VW-16); 98% (VW-32); and 100% (VW-64).

4.4 Random Instances

Because these instances are known to be hard for SAT solvers, we limit our attention to two solvers; Sparrow and PAWS using 16, 128, 256, and 512 cores. Figure 1(d) shows the RTD for Sparrow-16, Sparrow-512, and PAWS-512. Notice

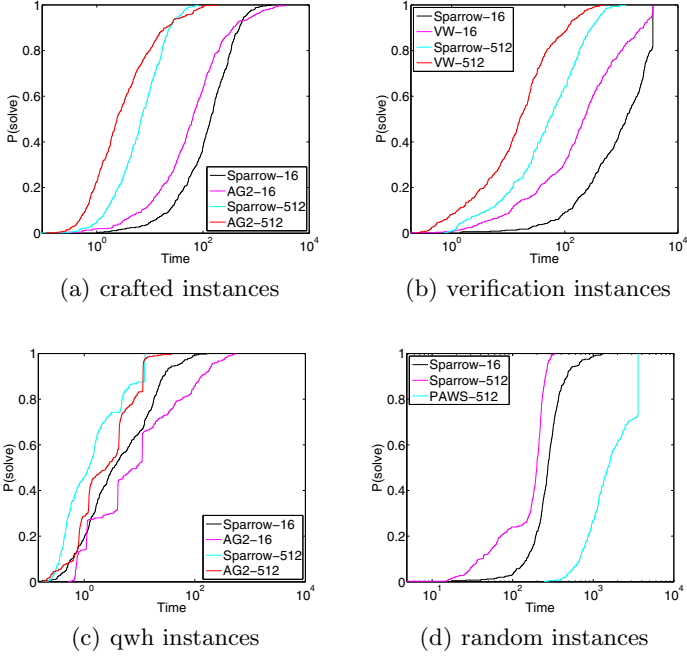


Fig. 1. RTD for portfolios using 16 and 512 cores

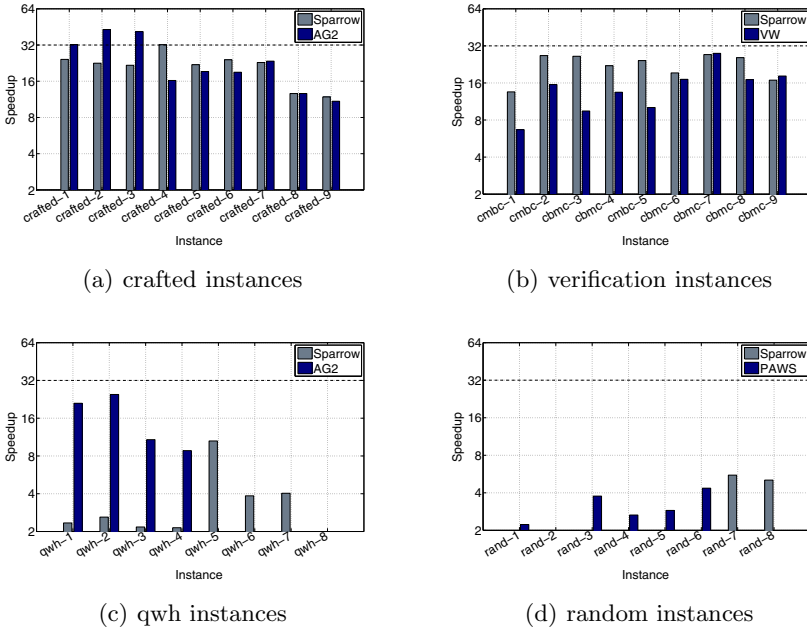


Fig. 2. Speedup (w.r.t. 16 cores) using 512 cores

Table 3. Performance summary for cbmc instances. Each cell indicates the median runtime (top) and the percentage of solved instances (bottom) for each instance.

Instance	Alg	Number of Cores					
		16	32	64	128	256	512
cbmc-1	SP	141.2	104.7	69.3	53.3	22.0	10.4
	VW	5.9	3.4	2.4	0.9	0.8	0.8
cbmc-2	SP	827.8	409.1	223.0	103.5	43.2	31.0
	VW	153.5	67.8	35.5	16.2	12.6	9.9
cbmc-3	SP	1052.6	500.3	216.2	135.8	83.8	39.9
	VW	112.8	95.1	34.5	20.9	16.1	11.9
cbmc-4	SP	910.4	395.0	192.3	117.3	78.8	41.2
	VW	198.1	106.75	53.78	27.5	14.3	14.7
cbmc-5	SP	1782.0	820.8	406.7	238.7	136.5	73.3
	VW	161.2	84.8	50.1	35.0	18.5	15.9

Instance	Alg	Number of Cores					
		16	32	64	128	256	512
cbmc-6	SP	1772.0	842.9	505.1	168.1	116.2	91.8
	VW	200.7	160.8	58.7	33.5	20.8	11.7
cbmc-7	SP	2510.1	1703.1	417.3	283.3	182.0	92.3
	VW	1526.3	462.2	289.5	151.5	89.6	54.8
cbmc-8	SP	3600.0	1808.0	807.4	458.0	230.1	140.2
	VW	1229.3	631.6	398.2	143.7	84.9	72.2
cbmc-9	SP	2140.4	900.5	533.1	325.1	190.3	127.0
	VW	1209.2	547.9	313.4	249.3	129.4	66.4

that PAWS-16 is not included because it solves a limited number of instances (see Table 4). In this figure, we observe a small improvement when increasing the number of cores. In particular, the runtime that Sparrow requires to solve a given instance within the time limit decreases from $P(\text{timeout} < 1250) \approx 1$ for Sparrow-16 to $P(\text{timeout} < 290) \approx 1$ for Sparrow-512.

On the other hand, the speedup observed for these instances is considerably different than the previous benchmarks. In fact, the speedup reported for PAWS in the figure is an approximation, because PAWS-16 timed-out for an important number of instances. For this reason, this figure (for PAWS) should be taken as a lower bound of the actual speedup.

Another interesting behavior observed in these experiments is that the speedup varies from instance to instance (see Table 4). For example, the speedup for instances near the phase transition (i.e. rand-[1-6]) is substantially lower than the speedup for the remaining instances (i.e. rand-[7-8]). We plan to conduct a more detailed investigation to fully characterize the performance of random instances near the phase transition region when using massively parallel systems.

Finally, it is worth noticing that although the speedup is limited for these instances, Table 4 shows an important improvement in the effectiveness of PAWS for nearly all instances. For instance, the effectiveness to solve rand-1 increases as follows: 16% (PAWS-16), 72% (PAWS-128), 88% (PAWS-256), and 94% (PAWS-512). However, the the effectiveness of PAWS-512 degrades 2% with regard to PAWS-256, this corresponds to 1 timeout out of 50 executions of PAWS-512 to solve rand-4.

5 Conclusions and Future Work

This paper has presented extensive experimental results using parallel portfolios of local search algorithms for SAT. Overall the experiments suggest that the portfolio approach scales reasonably well up to an important number of cores (i.e. 512 cores) without the need of any particular tuning of the algorithm.

Table 4. Performance summary for random instances. Each cell indicates the median runtime (top) and the percentage of solved instances (bottom) for each instance.

Instance	Alg	Number of Cores			
		16	128	256	512
rand-1	SP	260.00 100%	188.00 100%	180.08 100%	192.26 100%
	PAWS	3600.00 16%	2320.55 72%	1951.96 88%	1616.24 94%
rand-2	SP	299.28 100%	240.75 100%	227.34 100%	232.95 100%
	PAWS	3600.00 12%	3216.08 68%	2933.89 78%	1855.01 94%
rand-3	SP	277.68 100%	190.56 100%	192.59 100%	200.32 100%
	PAWS	3600.00 32%	1501.92 98%	1108.49 100%	956.37 100%
rand-4	SP	274.04 100%	221.14 100%	204.24 100%	214.89 100%
	PAWS	3394.61 50%	1602.76 96%	1289.28 100%	1279.33 98%

Instance	Alg	Number of Cores			
		16	128	256	512
rand-5	SP	305.66 100%	252.28 100%	242.26 100%	242.82 100%
	PAWS	3600.00 30%	1564.17 92%	1287.87 98%	1246.72 100%
rand-6	SP	248.93 100%	200.02 100%	198.36 100%	213.42 100%
	PAWS	3600.00 36%	1210.02 100%	1024.20 100%	826.30 100%
rand-7	SP	179.21 100%	54.95 100%	39.74 100%	32.32 100%
	PAWS	3600.00 0%	3600.00 0%	3600.00 0%	3600.00 0%
rand-8	SP	325.62 100%	90.00 100%	73.11 100%	64.39 100%
	PAWS	3600.00 0%	3600.00 0%	3600.00 0%	3600.00 0%

In two out of four benchmark families (crafted and verification) the algorithms exhibit near optimal speedups, and super-linear in some particular cases. However for the quasigroup instances, we have observed that the best sequential algorithm reports surprisingly for half of the instances a very limited speedup factor (roughly a factor 2 w.r.t. 16 cores, even with 512 cores), while the other (slower) algorithm scales well up to an important number of cores, without however reaching the same raw performance. For the random instances, which are very hard problems close to the phase transition, it is worth noticing that parallel speedups are quite limited for all the algorithms studied. However, we would also like to point out that, as expected, the parallel portfolio approach helps to increase the effectiveness of the algorithms when increasing the number of cores. For instance, for the rand-5 problem, the effectiveness of the PAWS algorithm increases from 30% (PAWS-16) to 100% (PAWS-512).

Therefore our experiments show that on different problem families the behaviors of different solvers vary greatly. Although our initial question is still open, i.e. *Is the best sequential solver also the best one in a massively parallel context?*, we have presented an empirical study which suggests that the best sequential solver is not necessarily the one with the over all best speedup.

Our future work involves the study of cooperative algorithms that scale up significantly for a large number of cores (e.g. 512 cores). Indeed, current cooperative methods for parallel local search for SAT scale only up to 16 or 32 cores (see [16]). In addition, we plan to investigate the use of machine learning to identify potentially bad and good runs in the parallel portfolio.

Acknowledgements. The first author was supported by the Japan Society for the Promotion of Science (JSPS) under the JSPS Postdoctoral Program and the *kakenhi* Grant-in-aid for Scientific Research. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see <https://www.grid5000.fr>).

References

1. Cook, S.A.: The Complexity of Theorem-Proving Procedures. In: Third Annual ACM Symposium on Theory of Computing, STOC 1971, pp. 151–158. ACM (1971)
2. Chrabakh, W., Wolski, R.: GridSAT: A System for Solving Satisfiability Problems Using a Computational Grid. *Parallel Computing* 32(9), 660–687 (2006)
3. Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: A Parallel SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT* 6(4), 245–262 (2009)
4. Arbelaez, A., Hamadi, Y.: Improving Parallel Local Search for SAT. In: Coello Coello, C.A. (ed.) LION 2011. LNCS, vol. 6683, pp. 46–60. Springer, Heidelberg (2011)
5. Hoos, H., Stützle, T.: *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco (2004)
6. Selman, B., Kautz, H.A., Cohen, B.: Noise Strategies for Improving Local Search. In: AAAI 1994, vol. 1, pp. 337–343 (July 1994)
7. McAllester, D.A., Selman, B., Kautz, H.A.: Evidence for Invariants in Local Search. In: AAAI 1997, pp. 321–326 (1997)
8. Thornton, J., Pham, D.N., Bain, S., Ferreira Jr., V.: Additive versus Multiplicative Clause Weighting for SAT. In: AAAI 2004, pp. 191–196 (July 2004)
9. Prestwich, S.D.: Random Walk with Continuously Smoothed Variable Weights. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 203–215. Springer, Heidelberg (2005)
10. Li, C.-M., Huang, W.Q.: Diversification and Determinism in Local Search for Satisfiability. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 158–172. Springer, Heidelberg (2005)
11. Balint, A., Fröhlich, A.: Improving Stochastic Local Search for SAT with a New Probability Distribution. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 10–15. Springer, Heidelberg (2010)
12. Shylo, O.V., Middelkoop, T., Pardalos, P.M.: Restart Strategies in Optimization: Parallel and Serial Cases. *Parallel Computing* 37(1), 60–68 (2011)
13. Pham, D.N., Gretton, C.: gNovelty+. In: *Solver Description, SAT Competition 2007* (2007)
14. Roli, A.: Criticality and Parallelism in Structured SAT Instances. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 714–719. Springer, Heidelberg (2002)
15. Kroc, L., Sabharwal, A., Gomes, C.P., Selman, B.: Integrating Systematic and Local Search Paradigms: A New Strategy for MaxSAT. In: IJCAI 2009, pp. 544–551 (July 2009)
16. Arbelaez, A., Codognet, P.: Massively Parallel Local Search for SAT. In: ICTAI 2012, Athens, Greece, pp. 57–64. IEEE Computer Society (November 2012)
17. Martins, R., Manquinho, V., Lynce, I.: An Overview of Parallel SAT Solving. *Constraints* 17, 304–347 (2012)
18. Tompkins, D.A.D., Hoos, H.H.: UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 306–320. Springer, Heidelberg (2005)
19. Achlioptas, D., Gomes, C.P., Kautz, H.A., Selman, B.: Generating satisfiable problem instances. In: AAAI 2000, pp. 256–261 (July 2000)
20. Clarke, E., Kroening, D., Lerda, F.: A Tool for Checking ANSI-C Programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004)
21. Tompkins, D.A.D., Balint, A., Hoos, H.H.: Captain Jack: New Variable Selection Heuristics in Local Search for SAT. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 302–316. Springer, Heidelberg (2011)
22. Gent, I.P., Walsh, T.: The SAT Phase Transition. In: ECAI 1994, pp. 105–109 (August 1994)

Generalizing Hyper-heuristics via Apprenticeship Learning

Shahriar Asta¹, Ender Özcan¹, Andrew J. Parkes¹, and A. Şima Etaner-Uyar²

¹ School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, U.K.

{sba, exo, ajp}@cs.nott.ac.uk
<http://cs.nott.ac.uk/~{sba, exo, ajp}/>

² Department of Computer Engineering
Istanbul Technical University
Istanbul, 34469, Turkey
etaner@itu.edu.tr
<http://web.itu.edu.tr/~etaner/>

Abstract. An *apprenticeship-learning*-based technique is used as a hyper-heuristic to generate heuristics for an online combinatorial problem. It observes and learns from the actions of a known-expert heuristic on small instances, but has the advantage of producing a general heuristic that works well on other larger instances. Specifically, we generate heuristic policies for online bin packing problem by using expert near-optimal policies produced by a hyper-heuristic on small instances, where learning is fast. The "expert" is a policy matrix that defines an index policy, and the apprenticeship learning is based on observation of the action of the expert policy together with a range of features of the bin being considered, and then applying a k-means classification. We show that the generated policy often performs better than the standard best-fit heuristic even when applied to instances much larger than the training set.

Keywords: Hyper-heuristics, learning by demonstration, apprenticeship learning, generalization.

1 Introduction and Related Work

Meta-heuristics have long been used to solve optimization problems using many versions of neighborhood search. However, the efficiency of meta-heuristics depend on the problem domain and the neighborhood operator. Thus, meta-heuristics may have different performances on different problem domains or even on different instances of the same problem. In order to overcome these dependencies, automated search techniques have emerged [8–10], and now are often generically called hyper-heuristics. Hyper-heuristics take the search process one level higher to the space of heuristics. That is, there is a higher level (meta-)heuristic which at each instance of time, chooses some low level and often simpler heuristic

to solve the problem. According to one classification [4], hyper-heuristics, like many machine learning problems, can be divided into three categories depending on the feedback mechanism they employ: *on-line learning*, *off-line learning* and *no learning*. If the hyper-heuristic framework learns while searching, it is an on-line learning hyper-heuristic. On the contrary, an off-line learning hyper-heuristic learns prior to the search phase. When no feedback is acquired from the search space, then the corresponding hyper-heuristic framework is a no learning framework. Hyper-heuristics can also be classified into two groups: *selection* and *generation* hyper-heuristic. The former, selects a heuristic among a set of existing heuristics at each phase of the search. The latter, generates new heuristics from components of the existing low level heuristics. Both selection and generation hyper-heuristics can be further categorized into construction or perturbation heuristics. More on hyper-heuristics can be found in [3, 13, 16]. Selection hyper-heuristics have been well studied, and gave rise to the CHeSC 2011 competition¹; further details of this can be found in [7, 12] and at the CHeSC website, and of the winning hyper-heuristic by Misir et al. in [11].

However, this paper is about generation rather than selection hyper-heuristics, and so rather than the neighbourhood search of CHeSC, we study the generation of heuristics for an online problem. Specifically, we study the online bin-packing problem and follow the policy matrix methods of Özcan and Parkes [14]. In those methods the goal is to produce an ‘index policy’, that assigns a score to all potential actions and then selects the highest scoring action. This is done by using direct search by a genetic algorithm. Earlier related work on online bin-packing [17] had proposed a hyper-heuristic approach which learns how to choose a heuristic based on the dynamically changing problem state after placement of each item for bin packing. Subsequent work (e.g. [5, 6]) used genetic programming methods to evolve an arithmetic expression for the scoring function within the policy. Parkes, Özcan and Hyde [15] combined previous studies and presented a method based on policy matrices for analysing the effects of the genetic programming mutation operator in a regular run using online bin packing. The policy matrix methods, and the methods of this paper, differ from that of [17], as it attempts to learn a single heuristic rather than learning how to mix them to construct a solution.

Although the policy matrix approach in [14] was effective at generating heuristics with better performance than the standard ones, it had the drawback of directly only applying to a specific set of values for the bin capacity and range of item sizes. In this paper, we describe a method to take policy matrices learned on small instances and generalise them to apply to different instances, and with the particular aim to apply them to larger instances. We used a form of apprenticeship learning (a.k.a learning by demonstration or imitation learning) [1] for generalizing the demonstrations provided by an expert. Apprenticeship learning has a wide range of applications in control and robotics and is heavily based on Inverse Reinforcement Learning (IRL). Although we do not use IRL methods

¹ Cross-domain Heuristic Search Challenge:
<http://www.asap.cs.nott.ac.uk/external/chesc2011/>

directly in our approach, our study is mainly inspired by them. Our method generates a generalized policy by classifying the actions of some expert policies (heuristics) according to each search state, thus it also can be viewed as a hyper-heuristic. We also note that one intention originally motivating the work of [14] was to produce good policy matrices and then data-mine them to learn good patterns. This work is somewhat different in that it does not learn directly from the policy matrix, but rather by observing the decisions that it lead to.

Our method in use is an off-line classification method, needing to be trained on an available dataset, and so in categorization of hyper-heuristics in [4] it best fits into the category of *off-line learning generation hyper-heuristics*. The study here includes only experimental results on a single problem domain, however, we expect the general methods it will also be applicable to other domains.

2 Policy Matrices for Online Bin Packing

2.1 Online Bin Packing Problem

The bin packing problem is known to be a combinatorial NP-hard problem which deals with packing items of different sizes to bins of fixed capacity. The objective is to minimize the number of bins used. Different variants of the bin packing problem exist, one of which is the *online* bin packing problem. In this variant, we are dealing with partitioning a set of integer values into subsets with the constraint that the sum of integers within a subset does not exceed the capacity [14]. Moreover, as a distinguishing feature, items arrive sequentially and each item has to be assigned to a bin before the next one is disclosed. A decision has to be made dynamically at each step based on partial information regarding which bin should be used for placement. This is in contrast to the off-line bin packing problem where there is a complete information on the number of items and their sizes prior to solving a problem instance.

The bin capacity is a constant integer $C > 1$ and the items can have any size in the range $[1, C]$. An open bin has a remaining capacity which can accommodate at least one item assuming that the sizes of items are known. An empty new bin is always available and it is opened if the size of the current item is bigger than the remaining capacity of all open bins. In such a case, the new bin is opened and the item is placed into this new bin. A bin is closed if its remaining space is smaller than the minimum item size. The uniform bin packing instances are represented by the formalism: $UBP(C, s_{min}, s_{max}, N)$ (adopted from [14]) where C is the bin capacity, s_{min} and s_{max} are minimum and maximum item sizes and N is the number of total items. The item sizes at each step are chosen uniformly and independently random from the range $[s_{min}, s_{max}]$. Also, we have the assumption $s_{min} > 0$ and $s_{max} < C$. The fitness measure for each experiment on N items is computed according to the following equation.

$$f = \frac{1}{B} \sum_t f_t \quad (1)$$

where B is the number of bins used and f_t is the fullness of bin t .

2.2 Matrix Representation of Policies

As discussed earlier, in our framework we need a set of initial policies which work fine in their own domains (a specific *UBP* here). Our methodology then utilizes these expert policies to form a generalized model over the problem domain. This generalized model is independent of the underlying policy and a framework which generates expert policies on a given instance is sufficient for the task. Due to its simple implementation, ease of use and high performance, we chose to utilize the work in [14] to generate our expert policies. A description of this method is given below.

Özcan and Parkes [14] proposed a hyper-heuristic method to generate matrix policies to solve instances of online bin packing problem. In their method, policy matrix evolution for generation of heuristics, a policy is represented by a matrix of scores (policy matrix). Each row in this matrix represents the remaining bin capacity (r) prior to the item assignment and each column represents the current item size (s) to be assigned to a bin. The values of each matrix element are either -1 for inactive elements (irrelevant (r, s) pairs which never occur) or W_{rs} which is the score associated with assigning item of size s to a bin of remaining capacity r . The value for W_{rs} is chosen from the range $[w_{min}, w_{max}]$. In our experiments we chose $w_{min} = 1$ and $w_{max} = 2$ for simplicity. The policy matrix is then optimized using an off-line learning GA for a given problem instance (a specific *UBP* as described in Section.2). Each individual is consisted of the values of the active members of the policy matrix. A generation of these individuals is then generated which goes through selection, recombination, mutation and evaluation. Please note that, since a single policy matrix is a heuristic, then the GA is a hyper-heuristic which searches in the space of heuristics. Further detail on this method can be found on [14]. The experimental results show that this method produces reliable policies which solve a given *UBP* with a high performance.

3 The Proposed Approach

One of the major contributions of this study is to show that each search state can be seen and described as a feature set with which a generalized model can be constructed. Thus, the feature set is a crucial part of our framework since it affects the performance of our method which benefits from classification algorithms (namely k-means). In order to achieve a desirable performance, the extracted features should be instance independent. That is, they should not be dependent on the absolute values of the item size (s), bin capacity (C) and minimum or maximum item size (s_{min} or s_{max}), but rather to depend on relative sizes. Table 1 shows the list of considered features along with their formal and verbal descriptions. The features in Table 1 are extracted for each open bin on the arrival of each new item. The last two features of the feature vector described in Table 1, are designed to increase the prediction power of the generalized policy. In other words, regardless of the decision of the expert policy on selecting or rejecting the current bin, we have assumed that the item has been assigned to the bin (if it's size does not exceed the bin's remaining capacity) to see what

Table 1. Features of the search state. Note that the UBP instance defines the constants C , s_{min} , and s_{max} whereas the variables are s the current item size, and r the remaining capacity in the bin considered, and r' is simply $r - s$.

feature	description
$(s - s_{min}) / (s_{max} - s_{min})$	normalized current item size
r / C	normalized remaining capacity of the current bin
s / C	ratio of item size to bin capacity
s / r	ratio of item size to the current bin's remaining capacity
r' / C	normalized remaining capacity of the current bin after a feasible assignment
$(r' / s_{max}) - s_{min}$	ratio of remaining capacity of the current bin after a feasible assignment to the range of item size

changes such an assignment makes in the search state. The new remaining capacity of such a hypothetical assignment is noted by the symbol $r' = r - s$ in Table 1.

In classical machine learning techniques, each row of features in the dataset determines a certain class label. In this work we chose to use the action which the expert policy prefers for each bin, as the label for each row of features (records). Typically, what is being done by the policy matrices, or any other policy in fact, is to either open a new bin or to choose an open bin and assign the item to that bin. Thus, in our work the label determines if the bin is selected (label 1) or rejected (label 0).

Having determined the necessary features for our method, we can now use our expert policies to extract features and their corresponding labels for each search state. That is, we assume that we are in possession of a set of n expert policies $\{\pi_e^1, \dots, \pi_e^n\}$ in one dimensional on-line bin packing problem domain. These expert policies are obtained by the policy generation method discussed in Section.2.2. Each expert policy corresponds to a certain *UBP*. We run each expert policy once, on it's corresponding *UBP* for a certain and fixed number of items $N = 10^5$. While running, expert features, ϕ_e^t , are extracted for each state of the search (t). Here, ϕ_e^t is a r dimensional vector of features where r is the number of features representing a search state. At the end of each run for a policy π_e^i we will have a set of demonstrations like:

$$\mathcal{D}_{\pi_e^i} = \{(\phi_e^t, a_t) | \pi_e^i\} \quad (2)$$

where a_t is the action at step t . The demonstration sets for all training policies are then merged together to form a dataset.

$$\mathcal{D} = \bigcup_{i=1}^n \mathcal{D}_{\pi_e^i} \quad (3)$$

Having the feature vectors and their associated labels, we employ a k-means clustering algorithm to cluster the feature vectors of each class. The k-means

algorithm is a semi-parametric method which uses a mixture of densities to estimate the input sample [2]. The distance metric in use (d) is one minus cosine similarity (Eq.6). The clustering process is designed to generate 8 cluster centroid coordinates, 4 of which labeled as selected bins (feature vectors labeled 1) and the rest as rejected bins (feature vectors labeled 0). The number of clusters for each class has been determined experimentally.

$$\phi_{x_j} = \frac{1}{n_j} \sum_{\phi_e^t \in x_j} \phi_e^t \approx E(\phi_e^t | x_j \in \mathcal{D}), \phi_e^t \in x_j, \exists i \text{ s.t. } \phi_e^t \in \pi_e^i \quad (4)$$

Here, x_j is the j th centroid and n_j is the number of samples which belong to the centroid j . For an unseen problem instance (a *UBP*), at each state of the search, say, on the arrival of each new item, for each open bin, the state features are extracted ($\phi^{t'}$) and the closest matching centroid to the current feature vector in terms of cosine similarity is found. In case the centroid has a label 1 the bin is selected for the item assignment according to a probability. The probability is chosen to be 0.99 and is considered to introduce randomness to the decision making process. Eq.5 illustrates the decision making mechanism of the generalized policy, given a feature vector for a bin and a set of centroids.

$$\pi_g = \{a_{x_j} \in \{0, 1\} \mid \underset{j}{\operatorname{argmin}} d(\phi_{x_j}, \phi^{t'}), \phi_{x_j} \in \mathcal{D}\} \quad (5)$$

Here, π_g is the generalized policy, the subscript x_j indicates the j th centroid obtained by the k-means clustering algorithm, a_{x_j} is the action (label) which is associated to the centroid j and d is the distance metric which is given in Eq.6.

$$d(\phi_{x_j}, \phi^{t'}) = 1 - \frac{\sum_r \phi_{x_j} \cdot \phi^{t'}}{\sqrt{\sum_r \phi_{x_j}^2} \cdot \sqrt{\sum_r \phi^{t'^2}}} \quad (6)$$

The summations in Eq.6 are over r , the dimension of the feature vector which is not shown as index in the equation in order to reduce the complexity of notations.

4 Experiments

Since we have used the policy matrices generated by the method in [14], a first round of training has been performed to obtain a set of expert policies using the hyper-heuristic in [14]. Then each policy matrix is run on it's corresponding instance to obtain a set of features for search states and form a data set (\mathcal{D} in Eq. 3). However, since the underlying machine which performs the clustering is an Ubuntu 10.10 with 3GB of RAM, it only can handle small datasets. In order to keep the dataset small enough to be processed by the computer, the actions of the expert on each instance is sampled randomly. That is, not all the states are considered for feature extraction. Instead, a feature vector is extracted for each state according to a uniformly random distribution with a probability of 0.15. The dataset is then clustered which represents the expected feature vector of the

expert. For unseen instances of the one dimensional bin packing problem, the feature vector for each open bin is extracted and it is determined if the feature vector of the bin belongs to selected or rejected bins (a choice performed by the expert).

4.1 Experimental Design

In order to obtain expert policies a GA framework as in [14] has been used for which the parameter setting is given in Table 2. Except the values of w_{min} and w_{max} , basically, the entries of Table 2 are the settings which were used in [14] and are given here for convenience. It should be noted that these values were suitable for small instances, but tend not to converge for larger instances. The ‘expert policies’ obtained can sometimes be significantly sub-optimal, due to the large computational resources needed to learn policies in some cases.

Table 2. GA parameter setting

No. of iterations	200	pop. size	$\lceil \frac{C}{7} \rceil$
Selection	Tournament	Tour size	2
Crossover	Uniform	Crossover Probability	1.0
Mutation	Traditional	Mutation Rate	0.1
No. of trials	1000	No. of Items	10^5
w_{min}	1	w_{max}	2

The problem instances under consideration are a total of 10 instances. We assume that we have the expert policy corresponding to each instance. That is, we used the GA to obtain an expert policy matrix corresponding to each instance. As a consequence, we know the performance of each expert policy on it’s corresponding instance, which is used for comparison in later stages of our experiments. However, in order to train and construct the generalized policy (π_g), we utilize only 3 instances to form the dataset and construct our model. We use the k-means algorithm to construct a generalized model of the choices of expert policies on their corresponding instances. The generalized policy then uses the resulting data set and the model to solve the remaining 7 instances. For feature extraction in the training phase the expert policy is run on it’s corresponding instance for a single run which contains 10^5 items. For testing purposes, the generalized policy is tested on each problem instance in the test fold for 100 runs, each including 10^5 items. The instances used to train the π_g and form the dataset are $UBP(15, 5, 10, 10^5)$, $UBP(30, 4, 20, 10^5)$ and $UBP(40, 10, 20, 10^5)$.

4.2 Experimental Results

As mentioned earlier, in our experiments, the expert policy performs a single run on its corresponding instance, resulting in the training feature set. Subsequently, the test instances, are used to test the generalized policy. The results of this experiment is shown in Table 3. In order to have a better understanding of the

Table 3. A comparison between the performances of the expert policy (π_e), generalized policy (π_g) and the best fit heuristic(BF) on various unseen instances. Numbers are average bin fullness percentages.

Instance	Average Performance			Max. Performance			Min. Performance		
	π_e	π_g	BF	π_e	π_g	BF	π_e	π_g	BF
$UBP(20, 5, 10, 10^5)$	98.42	94.32	91.55	98.47	94.41	91.66	94.33	94.21	91.46
$UBP(30, 4, 25, 10^5)$	99.68	97.69	98.38	99.76	97.92	98.49	99.52	97.36	98.30
$UBP(50, 10, 25, 10^5)$	99.20	93.32	93.31	99.31	93.41	93.41	99.08	93.25	93.26
$UBP(60, 15, 25, 10^5)$	99.75	93.83	92.54	99.91	94.80	92.65	99.45	92.91	92.42
$UBP(75, 10, 50, 10^5)$	98.45	98.50	96.08	98.51	98.54	96.13	98.37	98.44	96.04
$UBP(80, 10, 50, 10^5)$	98.86	98.17	96.39	98.91	98.21	96.44	98.74	98.13	96.34
$UBP(150, 20, 100, 10^5)$	97.56	98.32	95.81	97.66	98.37	95.87	97.49	98.26	95.76

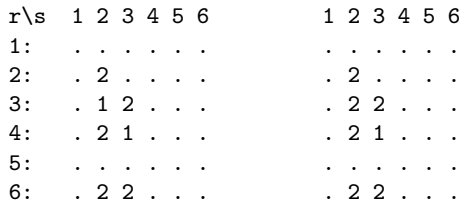


Fig. 1. The optimal and generalised matrix policies for $UBP(6, 2, 3, 10^5)$

results, also the results of the best fit (BF) heuristic is given as a lower bound. Please note that the reported results in Table 3 for π_e, π_g and best fit are obtained by running each policy for 100 runs on each problem instance.

The generalized policy (π_g) does not generate optimal policies for the test instances, however π_g follows the expert policy (π_e) in terms of performance as summarized in Table 3. Figure 1 illustrates the optimal policy along with a near optimal generalized policy yielding 96.45% mean bin fullness for the instance $UBP(6, 2, 3, 10^5)$ while BF generates a performance significantly worse than π_g with a mean bin fullness of 92.25%. The generalized policy is 1 Hamming-distance away from the known optimal, differing at $W_{3,2}$. In the case of the instance $UBP(30, 4, 25, 10^5)$ BF performs slightly better than the generalized policy, but in all other instances the generalized policy outperforms the BF. The performance differences are statistically significant for $UBP(20, 5, 10, 10^5)$, $UBP(60, 15, 25, 10^5)$ and the rest of the instances. It is observed that π_g is capable of generalizing the expert policies to larger problem instances. All problem instances in the training phase of π_g , are smaller in terms of bin capacity, minimum and maximum item size as compared to the instances in the test set. Applying the generalized policy to larger unseen problem instances still results with a performance similar to that of the expert policy. This achievement is important since by demonstrating expert actions on simple problem instances, our generalized method was able to perform well on larger instances without undergoing the time consuming cycle of genetic evolution.

5 Conclusion and Future Work

In this study, we have used the idea of apprenticeship learning to construct a generalized model of the problem domain using a set of expert policies derived by a hyper-heuristic. We have described each state of the search by a feature vector and used the feature vector to construct the generalized model. Our experiments show that without a need to re-construct new policies for new instances of the problem domain, our model is able to generalize some existing policies to the problem domain. Our conclusion is that this method can be generalized to a cross-domain level. However, in order to achieve such a level of generality, one has to first determine a common feature set which can be exploited in a domain-independent fashion. Our future work is to have an investigation on automatic feature extraction and selection methods for this purpose. Finally, one could 'complete the loop' and use the generalized policy to generate a policy matrix; which could be used to initialize the GA used in [14]. Such an initialization approach, instead of a randomized initialization scheme, may well be expected to reduce the total number of generations to generate a true expert policy on an unseen problem instance.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the Twenty-First International Conference on Machine Learning, ICML 2004, pp. 1–8. ACM, New York (2004)
2. Alpaydin, E.: Introduction to Machine Learning (Adaptive Computation and Machine Learning). The MIT Press (2004)
3. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Tech. Rep. No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham (2010)
4. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.: A Classification of Hyper-heuristic Approaches. In: Handbook of Metaheuristics. International Series in Operations Research & Management Science. Springer (2009)
5. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R.: The scalability of evolved on line bin packing heuristics. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation, pp. 2530–2537. IEEE Computational Intelligence Society, IEEE Press, Singapore (2007)
6. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO 2007, pp. 1559–1565. ACM, New York (2007)
7. Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.: Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In: Proceedings of the Multidisciplinary International Scheduling Conference (MISTA 2009), pp. 790–797 (2009)
8. Cowling, P., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)

9. Crowston, W.B., Glover, F., Thompson, G.L., Trawick, J.D.: Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR Research memorandum No. 117, GSIA, Carnegie Mellon University, Pittsburgh (1963)
10. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: *Industrial Scheduling*, pp. 225–251. Prentice-Hall (1963)
11. Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: A new hyper-heuristic as a general problem solver: An implementation in HyFlex. *Journal of Scheduling* (2012)
12. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search. In: Hao, J.-K., Middendorf, M. (eds.) *EvoCOP 2012*. LNCS, vol. 7245, pp. 136–147. Springer, Heidelberg (2012)
13. Özcan, E., Bilgin, B., Korkmaz, E.: A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.* 12, 3–23 (2008)
14. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011*, pp. 2011–2018. ACM, New York (2011)
15. Parkes, A.J., Özcan, E., Hyde, M.R.: Matrix Analysis of Genetic Programming Mutation. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) *EuroGP 2012*. LNCS, vol. 7244, pp. 158–169. Springer, Heidelberg (2012)
16. Ross, P.: Hyper-heuristics. In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, ch. 17, pp. 529–556. Springer (2005)
17. Ross, P., Marín-Blázquez, J.G., Schulenburg, S., Hart, E.: Learning a Procedure That Can Solve Hard Bin-Packing Problems: A New GA-Based Approach to Hyper-heuristics. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O’Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A.C., Dowsland, K.A., Jonoska, N., Miller, J. (eds.) *GECCO 2003*. LNCS, vol. 2724, pp. 1295–1306. Springer, Heidelberg (2003)

High-Order Sequence Entropies for Measuring Population Diversity in the Traveling Salesman Problem

Yuichi Nagata¹ and Isao Ono²

¹ Education Academy of Computational Life Sciences,
Tokyo Institute of Technology, Japan

² Interdisciplinary Graduate School of Science and Engineering,
Tokyo Institute of Technology, Japan

nagata@acl.s.titech.ac.jp, isao@dis.titech.ac.jp

Abstract. We propose two entropy-based diversity measures for evaluating population diversity in a genetic algorithm (GA) applied to the traveling salesman problem (TSP). In contrast to a commonly used entropy-based diversity measure, the proposed ones take into account high-order dependencies between the elements of individuals in the population. More precisely, the proposed ones capture dependencies in the sequences of up to $m + 1$ vertices included in the population (tours), whereas the commonly used one is the special case of the proposed ones with $m = 1$. We demonstrate that the proposed entropy-based diversity measures with appropriate values of m evaluate population diversity more appropriately than does the commonly used one.

1 Introduction

The maintenance of population diversity is one of the most important factors for fully exercising the capability of genetic algorithms (GAs). To develop an effective population diversity management strategy, it is important to design an appropriate measure of population diversity, which could be used in adaptively changing search strategies (e.g., change a search strategy when the degree of population diversity becomes less than a specified value) [6,7], in analyzing the behavior of GAs [9,5,8], and in evaluating individuals to maintain population diversity in a positive manner [1,10,3,4].

In the information theory, entropy, defined as $-\sum_{s \in S} p_s \log p_s$, is a measure of the uncertainty of a probability distribution p_s ($s \in S$), where S is a set of all possible events. This definition, however, cannot be directly used to measure population diversity (i.e., S is a set of all possible solution candidates) because the population size is usually extremely smaller than the number of all possible solution candidates. Therefore, to the best of our knowledge, the entropy-based diversity measures proposed in previous works are all defined as the sum of the entropies of the univariate marginal distributions of all variables. For example, let the solution space be $\{x_1, \dots, x_n\}$, where x_i is a variable

taking values in a discrete set A_i . The entropy of the i -th variable is defined as $H_i = -\sum_{j \in A_i} p_{ij} \log p_{ij}$, where p_{ij} is the probability that x_i has a value j in the population. Then, the entropy-based diversity measure is defined as $H = \sum_{i=1}^n H_i$. In this paper, we call an entropy-based diversity measure defined in this manner an *independent entropy measure*. In previous works, independent entropy measures were designed for the knapsack problem [2], binary quadratic programming problem [8], traveling salesman problem [9,1,6,5,3,4], and others [10].

The independent entropy measure (and other commonly used population diversity measures), however, is not able to take into account dependencies between variables of individuals in the population. For example, consider an extreme example on the n -dimensional binary solution space where half of the population members are ‘00...00’ and the other half are ‘11...11’. The value of the independent entropy measure of this population will be almost the same as that of a randomly generated population because $p_{i0} = p_{i1} = 0.5$ ($i = 1, \dots, n$) for both cases, even though “true” population diversity is extremely low in the former case. Therefore, our motivation here is to design a more appropriate entropy-based diversity measure by considering the dependencies between variables of individuals in the population. We call such a diversity measure a *high-order entropy measure*.

In this paper, we propose high-order entropy measures for the traveling salesman problem (TSP) in order to design a more appropriate population diversity measure. In our previous work [4], we proposed a powerful GA for the TSP, which is one of the most effective heuristic (approximate) algorithms for the TSP. One important factor for achieving top performance is to maintain population diversity by evaluating each of the offspring solutions on the basis of the original evaluation function (tour length) as well as the contribution of diversity to the population when it is selected for inclusion in the population. Here, an independent entropy measure was used for evaluating population diversity. In this paper, we perform this GA by replacing the original independent entropy measure with each of the proposed high-order entropy measures in the evaluation function in order to show their ability to measure population diversity. Experimental results show that the use of high-order entropy measures improved the performance of the GA with respect to the solution quality.

The remainder of this paper is organized as follows. In Section 2, we first describe the original independent entropy measure for the TSP and present two types of high-order entropy measures. The GA framework, under which the population diversity measures were tested, is described in Section 3. Computational results are presented in Section 4. In Section 5, conclusions are given.

2 Entropy-Based Measures for the TSP

Let a TSP be defined on a complete directed graph (V, E) with a set of vertices $V = \{1, \dots, n\}$ and a set of edges $E = \{(i, j) \mid i, j \in V\}$. Note that entropy-based diversity measures are more easily defined in the asymmetric TSP (ATSP) than

in the symmetric TSP (STSP). In what follows, we explain them in the symmetric case (those in the asymmetric case can be naturally understood) and call both STSP and ATSP simply TSP unless otherwise stated. Let the population consists of N_{pop} individuals (tours). We first describe the independent entropy measure for the TSP, which was used in [1,5,3,4], and we call it *independent edge entropy*. Then, we present two high-order entropy measures for the TSP, which we call *high-order sequence entropies*.

2.1 Independent Edge Entropy H^{ind}

Let X_i ($i = 1, \dots, n$) be a random variable representing a vertex that follows vertex i in a tour of the population. Let the marginal probability distribution of the vertices that follow vertex i in the population be denoted as $P(X_i = x_i)$ or $P(x_i)$ for simplicity. For each tour in the population, we consider the two tours traveled in both directions because population diversity for the STSP should not depend on the travel direction. Therefore, $P(x_i)$ is the probability distribution of the vertices linked to vertex i in the population. The independent edge entropy, which we denote as H^{ind} , is then defined as follows: $H^{ind} = -\sum_{i=1}^n \sum_{x_i=1}^n P(x_i) \log P(x_i)$.

Consider the probability distribution given by

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^n P(x_i), \tag{1}$$

which is a rough estimation of the joint probability distribution of individuals in the population where all variables are independent of each other. We can see that H^{ind} is in fact the entropy of this probability distribution. That is,

$$H^{ind} = -\sum_{x_1} \dots \sum_{x_n} P(x_1, \dots, x_n) \log P(x_1, \dots, x_n) = -\sum_{i=1}^n \sum_{x_i=1}^n P(x_i) \log P(x_i). \tag{2}$$

We also define another commonly used population diversity measure for the TSP. Most commonly used one besides H^{ind} would be the average distance between all pairs of individuals in the population. The distance between two individuals is defined as the number of (undirected) edges of one solution that do not exist in the other. Using the above definition of $P(x_i)$, the average distance, which we denote as D , is calculated as follows (the derivation is omitted):

$$D = n - 2 \sum_{i=1}^n \sum_{x_i=1}^n P(x_i)^2. \tag{3}$$

As suggested in [4], one advantage of H^{ind} is the sensitivity to the change of rare edges in the population, and this feature makes H^{ind} a more appropriate population diversity measure than other commonly used ones such as D . We will compare H^{ind} with D in the experiments to show the advantage of H^{ind} .

2.2 High-Order Sequence Entropy H_m

A good way to define a more appropriate entropy-based diversity measure would be to approximate the joint probability distribution by taking into account dependencies between variables and then to calculate its entropy. The joint probability distribution encoded by a Bayesian Network may be useful for this purpose; it is represented as follows: $P(x_1, x_2, \dots, x_n) = \prod_{i=1}^n P(x_i \mid \mathbf{pa}_i)$ where \mathbf{pa}_i is a set of the variables on which a variable X_i depends. However, it is usually difficult to detect an appropriate high-order dependency structure in advance, and we therefore model the joint probability distribution in a different way.

We model the joint probability distribution of individuals in the population on the assumption that the probability of a vertex appearing in the population (tours) depends on the sequence of m (≥ 1) precedent vertices. Let S_i ($i = 1, \dots, n$) be a random variable representing the i -th vertex in a tour of the population. Given that a tour has a cyclic structure, the joint probability distribution, denoted as $P(s_1, s_2, \dots, s_n)$, is modeled as¹

$$P(s_1, s_2, \dots, s_n) = \prod_{i=1}^n P(s_{i+m} \mid s_i, \dots, s_{i+m-1}), \tag{4}$$

where $i + n$ ($1 \leq i \leq m$) corresponds to i .

We define an m -th order sequence entropy as the entropy of this probability distribution. The entropy H of this probability distribution is calculated as follows:

$$H = - \sum_{s_1} \dots \sum_{s_n} P(s_1, \dots, s_n) \log P(s_1, \dots, s_n) \tag{5}$$

$$= - \sum_{s_1} \dots \sum_{s_n} P(s_1, \dots, s_n) \sum_{i=1}^n \log P(s_{i+m} \mid s_i, \dots, s_{i+m-1}) \tag{6}$$

$$= - \sum_{i=1}^n \left\{ \sum_{s_i} \dots \sum_{s_{i+m}} P(s_i, \dots, s_{i+m}) \log P(s_{i+m} \mid s_i, \dots, s_{i+m-1}) \right\}. \tag{7}$$

Given that a tour can start from an arbitrary vertex, $P(s_i, \dots, s_{i+m})$ and $P(s_{i+m} \mid s_i, \dots, s_{i+m-1})$ should be equivalent to $P(s_1, \dots, s_{m+1})$ and $P(s_{m+1} \mid s_1, \dots, s_m)$, respectively, regardless of the value of i . Then, Eq. (7) can be simplified as follows:

$$H = -n \sum_{s_1} \dots \sum_{s_{m+1}} P(s_1, \dots, s_{m+1}) \log P(s_{m+1} \mid s_1, \dots, s_m) \tag{8}$$

$$= -n \sum_{s_1} \dots \sum_{s_{m+1}} P(s_1, \dots, s_{m+1}) \log \frac{P(s_1, \dots, s_{m+1})}{P(s_1, \dots, s_m)} \tag{9}$$

$$= -n \left\{ \sum_{s_1} \dots \sum_{s_{m+1}} P(s_1, \dots, s_{m+1}) \log P(s_1, \dots, s_{m+1}) \right\}$$

¹ If a cyclic structure is not assumed, the joint probability distribution is modeled as $P(s_1, s_2, \dots, s_n) = P(s_1, \dots, s_m) \prod_{i=1}^{n-m} P(s_{i+m} \mid s_i, \dots, s_{i+m-1})$.

$$- \sum_{s_1} \dots \sum_{s_m} P(s_1, \dots, s_m) \log P(s_1, \dots, s_m) \} \tag{10}$$

$$= n(\overline{H_{m+1}} - \overline{H_m}), \tag{11}$$

where

$$\overline{H_k} = - \sum_{s_1} \dots \sum_{s_k} P(s_1, \dots, s_k) \log P(s_1, \dots, s_k). \tag{12}$$

By ignoring the constant factor, we define an m -th order sequence entropy, which we denote H_m , as

$$H_m = \overline{H_{m+1}} - \overline{H_m}. \tag{13}$$

To compute H_m , we must estimate $P(s_1, \dots, s_k)$ for $k = m, m + 1$ by sampling sequences of vertices from individuals in the population. Let $N(s_1, \dots, s_k)$ denote the number of a sequence of vertices $\{s_1, \dots, s_k\}$ in the population, i.e., the number of tours containing this sequence in the population. Note that sequences of length k are sampled in both travel directions, resulting in $2n$ samples for each tour in the population (see Fig. 1). Then, we define $P(s_1, \dots, s_k) = \frac{N(s_1, \dots, s_k)}{2nN_{pop}}$. We store $N(s_1, \dots, s_k)$ ($k \leq m + 1$) in the form of a tree as illustrated in Figure 2 because it is impractical to store all possible entries in a table for a large value of m .

We should note that H_m is known as the entropy rate of an m -th order Markov information source. A central theorem of information theory states that the entropy rate of a data source means the average number of bits per symbol needed to encode it. Therefore, the existence of the same sequence consisting of up to $m + 1$ vertices in the population will decrease the value of H_m . We should also note that $H_1 (= \overline{H_2} - \overline{H_1})$ is essentially equivalent to H^{ind} because $P(S_1 = i, S_2 = j) = \frac{1}{n}P(X_i = j)$ and $\overline{H_1}$ is a constant value. In fact, the following relation holds:

$$H_1 = \frac{1}{n}H^{ind} + const. \tag{14}$$

As the value of m is increased, the high-order sequence entropy H_m would capture higher-order dependencies in the sequences of vertices included in the population. At the same time, however, it would not be likely to obtain a sufficient number of samples (sequences of vertices) from the population necessary to compute H_m . In particular, the estimation of $P(s_{m+1} | s_1, \dots, s_m)$ in Eq. (8), which is calculated as $\frac{N(s_1, \dots, s_{m+1})}{N(s_1, \dots, s_m)}$, is unreliable when $N(s_1, \dots, s_m)$ is small. Therefore, H_m will become useless if the value of m is too large. This problem will be more pronounced when the population size is set to a smaller value.

2.3 High-Order Sequence Entropy H_m'

We propose another variant of high-order sequence entropy. As previously mentioned, the high-order sequence entropy H_m would not be a meaningful population diversity measure for a greater value of m due to the lack of available samples necessary for computing it. To handle this problem, we consider

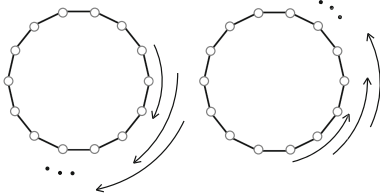


Fig. 1. An illustration on how to sample sequences of length k in a tour ($k = 3$)

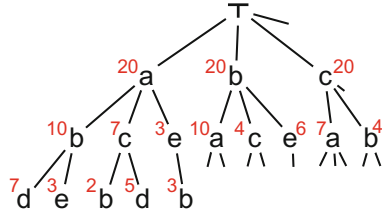


Fig. 2. A tree representation of $N(s_1, \dots, s_k)$ ($k \leq 3, N_{pop} = 10$). For example, $N(a, b) = 10$, $N(a, b, d) = 7$ and $N(a, b, c) = 0$.

a weighted sum of the high-order sequence entropies H_k ($k = 1, \dots, m$) defined as $\gamma_1(\overline{H_2} - \overline{H_1}) + \gamma_2(\overline{H_3} - \overline{H_2}) + \dots + \gamma_m(\overline{H_{m+1}} - \overline{H_m})$. In particular, when $\gamma_1 = \gamma_2 = \dots = \gamma_m = 1$, the weighted sum, which we denote H_m' , is given by

$$H_m' = \overline{H_{m+1}} - \overline{H_1}. \tag{15}$$

Compared to H_m , the high-order sequence entropy H_m' is easy to compute ($\overline{H_1}$ is a constant value) and will alleviate the problem of the lack of available samples for a greater value of m .

3 GA Framework

To show the advantage of the two high-order sequence entropies over the independent edge entropy, we perform the GA proposed in [4] with each of the three population diversity measures (H^{ind} , H_m , and H_m'). This GA is one of the most effective heuristic (approximate) algorithms for the TSP. One important factor for achieving top performance is to maintain population diversity by evaluating each of offspring solutions on the basis of the original evaluation function (tour length) as well as the contribution of diversity to the population when it is selected for inclusion in the population. The independent entropy measure was originally used for evaluating population diversity, and we will use the two high-order sequence entropies instead of the original one.

Algorithm 1 gives the GA framework proposed in [4]. The population consists of N_{pop} individuals. The initial population is generated by a greedy local search algorithm with the $2-opt$ neighborhood (line 1). At each generation (lines 3–8) of the GA, each of the population members is selected, once as parent p_A and once as parent p_B , in random order (lines 3 and 5). For each pair of parents, edge assembly crossover (EAX) operator generates N_{ch} (parameter) offspring solutions (line 6). Then, a best solution is selected in terms of a given evaluation function from the generated offspring solutions and p_A , and the selected one replaces the population member selected as p_A (line 7). Therefore, no replacement occurs if

all offspring solutions are worse than p_A . Note that only parent p_A is replaced in order to better maintain population diversity because EAX typically generates offspring solutions similar to p_A . Iterations of generation are repeated until a termination condition is met (line 9). For more details, we refer the reader the original paper.

Algorithm 1. Procedure GA

```

1:  $\{x_1, \dots, x_{N_{pop}}\} := \text{GENERATE\_INITIAL\_POP}()$ ;
2: repeat
3:    $r(\cdot) :=$  a random permutation of  $1, \dots, N_{pop}$ ;
4:   for  $i := 1$  to  $N_{pop}$  do
5:      $p_A := x_{r(i)}$ ,  $p_B := x_{r(i+1)}$ ;  $(r(N_{pop} + 1) = r(1))$ 
6:      $\{c_1, \dots, c_{N_{ch}}\} := \text{CROSSOVER}(p_A, p_B)$ ;
7:      $x_{r(i)} := \text{SELECT\_BEST}(c_1, \dots, c_{N_{ch}}, p_A)$ ;
8:   end for
9: until a termination condition is satisfied
10: return the best individual in the population;

```

The individual that replaces $x_{r(i)}$ ($= p_A$) is selected from the offspring solutions and p_A according to a given evaluation function (line 7). Let L be the average tour length of the population and H the population diversity (H^{ind} , H_m , or H_m'). The individual to replace $x_{r(i)}$ is selected such that $L - TH$ is minimized after the replacement where T is a parameter that takes a balance of L and H . However, we select the individual to replace $x_{r(i)}$ only from those that do not increase the value of L in order to prevent the population from not converging (without this restriction, it is difficult to make appropriate adjustment of T). Therefore, offspring solutions and p_A are evaluated by the following evaluation function², and the one with the smallest value is selected to replace $x_{r(i)}$.

$$Eval(y) = \begin{cases} \Delta L(y) - T\Delta H(y) & (\Delta L \leq 0) \\ \infty & (\Delta L > 0) \end{cases} \quad (16)$$

Here $\Delta L(y)$ and $\Delta H(y)$ denote the differences in L and H , respectively, when $x_{r(i)}$ is replaced with an offspring solution y . Note that if all evaluation values of the offspring solutions are greater than zero, p_A is selected (i.e., no replacement occurs) because $Eval(p_A) = 0$. We update the parameter T as follows. At the beginning of the GA, T is set to ∞ (a sufficiently large value). The value of T is updated each time the replacement of $x_{r(i)}$ is performed. Let L' and H' be the values of L and H , respectively, when the replacement of $x_{r(i)}$ N_{pop} times ago was performed (replacement by p_A is not counted). The value of T is updated as follows: $T = \frac{L-L'}{H-H'}$ if $H - H' < 0$, or ∞ (a sufficiently large value) otherwise. The later situation occurs at the early stage of the GA where the average tour length L can be decreased while increasing the population diversity H .

² Although a slightly different evaluation function was used in [4], but we employ this evaluation function in this research because this evaluation function is more natural.

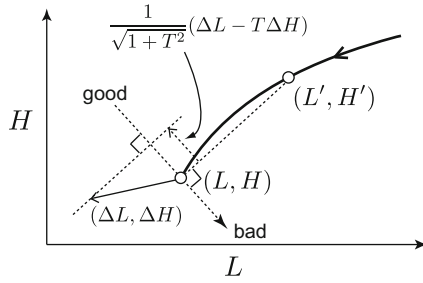


Fig. 3. The meaning of the evaluation function $\Delta L(y) - T\Delta H(y)$. The curve is the trajectory of (L, H) .

Figure 3 illustrates the meaning of the evaluation function. Note that during the first N_{pop} replacements of $x_{r(i)}$, T is not updated.

For every offspring solution y , we must compute $\Delta L(y)$ and $\Delta H(y)$ to obtain the value of $Eval(y)$. We need a little ingenuity for an efficient computation of $\Delta H(y)$, which makes it possible to compute $\Delta H(y)$ in $O(km)$ time, where k is the number of edges of an offspring solution y that do not exist in the parent p_A (k is usually much smaller than n). Each time $x_{r(i)}$ is replaced, the tree storing $N(s_1, \dots, s_k)$ must be updated, which takes $O(km)$ time.

4 Computational Experiments

4.1 Experimental Settings

To investigate the ability of the proposed high-order sequence entropies for measuring population diversity, we performed the GA described in the previous section by using different diversity measures in the evaluation function Eq. (16). The important parameters for the GA were set to the same values as in the previous work (see [4] for details): $N_{pop} = 300$ and $N_{ch} = 30$. We tested the four diversity measures summarized below. Note that H_1 and H_1' are equivalent to H^{ind} .

- The independent edge entropy: H^{ind} .
- The high-order sequence entropy: H_m ($m = 2, 3, 4, 6$).
- The high-order sequence entropy: H_m' ($m = 2, 3, 4, 6$).
- The average distance between all pairs of individuals in the population: D .

We also performed the GA with no use of a diversity measure by setting $T = 0$ in order to show the impact of the use of the four diversity measures in the evaluation function. In this case, N_{pop} was set to 600 because the population converges more rapidly when no diversity measure is used.

For each setting, we performed the GA 30 times on 21 instances with sizes ranging from 10,000 to 25,000 in the well-known benchmark sets for the STSP: TSPLIB (<http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95>) and National and VLSI TSP benchmarks (<http://www.tsp.gatech.edu/world/countries.html>).

4.2 Main Results

Table 1 shows the results in the following format: the instance name (instance) together with the optimal or best known solution (Opt. or UB), the number of runs that succeeded in finding the optimal or best-known solution (#S), and the average percentage excess over the optimal or best-known solutions (A-Err). We performed the one-sided Wilcoxon rank sum test for the null hypothesis that the median of the distribution of tour length derived from the GA using each of D , H_m and H_m' is greater than that of GA using H^{ind} . If the null hypothesis is rejected as a significant level of 0.05, results in the table are indicated by the asterisk. In addition, results are also indicated by the dagger if the opposite null hypothesis is rejected. Note that we employed a non-parametric statistical test because the data (tour length) is not normally distributed if an optimal solution is frequently found for each instance.

Table 1 shows that the GA using the independent edge entropy H^{ind} outperformed the GA using the average distance D in terms of the solution quality. As suggested in [4], one important advantage of H^{ind} over D is the sensitivity to the change of rare edges in the population, and this feature makes H^{ind} a more appropriate population diversity measure than D . When no diversity measure was used (i.e., $T = 0$), the average #S and A-Err over all instances were respectively 1.2 and 0.00544 (results are omitted in Table 1), even though population size was set to 600. Therefore, the diversity preserving selection using each H^{ind} and D clearly has a positive impact on the solution quality.

Table 1 shows that the use of each of the high-order sequence entropies H_m and H_m' with various values of m more or less further improved the solution quality achieved by the GA using H^{ind} . Let us make more detailed comparisons. First, we focus on the result of the high-order sequence entropy H_m . Table 1 shows that the GA using H_m with $m = 2, 3$, and 4 achieved better solution quality than did the GA using H^{ind} , indicating that H_m with $m = 2, 3$, and 4 are better population diversity measures than H^{ind} . The use of H_m with $m = 6$, however, did not show a significant improvement. As described in Section 2.2, a major reason would be the lack of available samples necessary to compute H_m for a greater value of m .

Next, we focus on the results of the high-order sequence entropy H_m' . Table 1 shows that the GA using H_m' achieved better solution quality than did the GA using H^{ind} for all values of m . Contrary to the results of H_m , the improvement was most pronounced at $m = 4$ and 6, meaning that the use of H_m' successfully reduces the problem of the lack of available samples for a greater value of m . Overall, the GA using H_m' with $m = 4$ or 6 achieved the best solution quality among all GAs tested, indicating that H_m' with $m = 4$ or 6 is the most appropriate population diversity measure.

Our research focus in this paper is to show the potential of the high-order entropy measure rather than to improve the performance of the GA. However, we should mention the effect on the computation time of the GA when the two high-order sequence entropies were used. The GA was implemented in C++ and executed on a cluster with Intel Xeon 2.93 GHz nodes. Table 2 shows the

Table 1. Solution quality of the GA with different diversity measures

		D		H_m							
		#S	A-Err	$m = 2$		$m = 3$		$m = 4$		$m = 6$	
Instance	Opt.(UB)			#S	A-Err	#S	A-Err	#S	A-Err	#S	A-Err
xmc10150	(28387)	18	0.00223	23	0.00106	22	0.00094	24	0.00070	28	0.00023*
fi10639	520527	7	0.00028 [†]	23	0.00008	23	0.00022	24	0.00011	14	0.00037 [†]
rl11849	923288	9	0.00103 [†]	26	0.00017*	26	0.00019*	25	0.00014*	10	0.00049
usa13509	19982859	5	0.00071 [†]	14	0.00017	15	0.00017	22	0.00010	11	0.00028 [†]
xvb13584	(37083)	23	0.00081	25	0.00072	23	0.00063	29	0.00009*	27	0.00036
brd14051	469385	19	0.00026 [†]	25	0.00006 [†]	23	0.00012 [†]	23	0.00017 [†]	21	0.00022 [†]
mo14185	(427377)	0	0.00091 [†]	20	0.00017	24	0.00009*	19	0.00014	20	0.00012
xrb14233	(45462)	0	0.00645 [†]	8	0.00345	2	0.00440	10	0.00279*	15	0.00198*
d15112	1573084	11	0.00022 [†]	21	0.00004	19	0.00007	16	0.00014	16	0.00013
it16862	557315	0	0.00112 [†]	2	0.00038*	0	0.00054	6	0.00023*	2	0.00041*
xia16928	(52850)	1	0.00391 [†]	14	0.00177	15	0.00139	24	0.00076*	25	0.00050*
pjh17845	(48092)	6	0.00208	13	0.00118*	19	0.00076*	13	0.00132*	12	0.00146
d18512	645238	17	0.00014	23	0.00005	21	0.00005	21	0.00009	13	0.00019 [†]
frh19289	(55798)	23	0.00048 [†]	30	0.00000	30	0.00000	30	0.00000	23	0.00066 [†]
fnc19402	(59287)	8	0.00281	21	0.00062*	22	0.00045*	19	0.00067*	18	0.00067
ido21215	(63517)	6	0.00352 [†]	21	0.00068	18	0.00089	23	0.00058	17	0.00079
fma21553	(66527)	11	0.00150	16	0.00080	16	0.00070	15	0.00090	10	0.00150
vm22775	569288	0	0.00254 [†]	0	0.00162 [†]	0	0.00121	0	0.00140	1	0.00150
lsb22777	(60977)	13	0.00137 [†]	22	0.00044	21	0.00049	21	0.00055	23	0.00038
xrh24104	(69294)	10	0.00125 [†]	29	0.00005	29	0.00005	29	0.00005	25	0.00038
sw24978	855597	1	0.00077 [†]	5	0.00042	14	0.00029	9	0.00039	7	0.00062
Average		9.0	0.00164	18.1	0.00066	18.2	0.00065	19.1	0.00054	16.1	0.00063

		H^{ind}		H_m'							
		#S	A-Err	$m = 2$		$m = 3$		$m = 4$		$m = 6$	
Instance	Opt.(UB)			#S	A-Err	#S	A-Err	#S	A-Err	#S	A-Err
xmc10150	(28387)	21	0.00129	21	0.00129	22	0.00106	22	0.00106	28	0.00023*
fi10639	520527	21	0.00006	22	0.00010	25	0.00007	24	0.00006	26	0.00012
rl11849	923288	20	0.00039	28	0.00006*	29	0.00004*	23	0.00027	28	0.00009*
usa13509	19982859	15	0.00014	11	0.00020	22	0.00007*	22	0.00009*	21	0.00012
xvb13584	(37083)	22	0.00081	22	0.00072	28	0.00018*	28	0.00018*	26	0.00036
brd14051	469385	29	0.00002	29	0.00002	28	0.00004	30	0.00000	26	0.00009
mo14185	(427377)	18	0.00020	23	0.00010	21	0.00014	26	0.00006*	24	0.00009*
xrb14233	(45462)	5	0.00396	6	0.00381	6	0.00359	5	0.00359	8	0.00308*
d15112	1573084	16	0.00010	21	0.00003*	21	0.00002*	24	0.00001*	21	0.00004
it16862	557315	5	0.00060	9	0.00047	8	0.00041*	6	0.00023*	10	0.00023*
xia16928	(52850)	10	0.00221	9	0.00233	11	0.00252	15	0.00132*	20	0.00114*
pjh17845	(48092)	8	0.00194	19	0.00083*	11	0.00132*	21	0.00062*	14	0.00125*
d18512	645238	21	0.00009	26	0.00003	20	0.00007	24	0.00004	25	0.00003
frh19289	(55798)	29	0.00012	30	0.00000	30	0.00000	30	0.00000	29	0.00006
fnc19402	(59287)	15	0.00180	18	0.00118	25	0.00039*	18	0.00073	24	0.00034*
ido21215	(63517)	23	0.00068	24	0.00058	27	0.00016	23	0.00063	23	0.00047
fma21553	(66527)	14	0.00105	21	0.00055*	17	0.00070	19	0.00065	20	0.00050*
vm22775	569288	1	0.00132	0	0.00117	0	0.00136	0	0.00097*	0	0.00125
lsb22777	(60977)	24	0.00044	21	0.00049	25	0.00027	26	0.00033	29	0.00005*
xrh24104	(69294)	26	0.00024	25	0.00024	27	0.00014	30	0.00000*	29	0.00005
sw24978	855597	7	0.00045	14	0.00026*	12	0.00028	14	0.00028*	21	0.00016*
Average		16.7	0.00085	19.0	0.00069	19.8	0.00061	20.5	0.00053	21.5	0.00046

computation time in seconds for a single run of the GA. Due to space limitation, results are presented for a limited number of instances. As shown in the table, the computation time increased with increasing the value of m in both cases. The main cause was the increase of the computational cost for computing $\Delta H(y)$.

Table 2. Computation time (in seconds) of the GA with different diversity measures

	D	H^{ind}	H_m				H_m'			
			$m = 2$	$m = 3$	$m = 4$	$m = 6$	$m = 2$	$m = 3$	$m = 4$	$m = 6$
usa13509	3055	3533	4793	4298	5442	7459	4493	4735	5286	6547
d15112	5085	5918	6141	8382	6657	7603	5979	6350	7526	9746
it16862	3886	4248	4781	6034	7777	10234	5440	5930	8164	10958
pjh17845	1847	2417	3125	3143	3920	5024	2824	3073	3746	5528
fma21553	2849	3336	3480	3845	4262	7433	4131	3680	4495	6684
sw24978	6692	9387	11226	10914	10413	13219	9354	9848	11465	14860

4.3 Effect of the Population Size

We investigate the relation between the appropriate value of m for H_m and H_m' and the population size. As described in Section 2.2 and demonstrated in Section 4.2, the high-order sequence entropy H_m will become useless if the value of m is too large because it would not be likely to obtain a sufficient number of samples (sequences of vertices) from the population necessary to compute H_m . This problem will be more pronounced when the population size is set to a smaller value. The high-order sequence entropy H_m' would also have the same problem even if this problem is alleviated.

To confirm the above hypothesis, we conducted the same experiment with different population sizes; $N_{pop} = 50, 100,$ and 300 (original value). Table 3 shows the results but only the average #S and A-Err over all instances are presented. For each row, the best value is highlighted in boldface. We can see that the best value of m for H_m increases with increasing the population size. We can also see that H_m' has the same tendency, but compared to the results of H_m , the best result was obtained with a greater value of m for each population size.

Table 3. Solution quality of the GA with different diversity measures and with different population sizes (only average results over all instances)

Diversity (N_{pop})	$m = 1$		$m = 2$		$m = 3$		$m = 4$		$m = 6$	
	#S	A-Err	#S	A-Err	#S	A-Err	#S	A-Err	#S	A-Err
H_m (50)	0.00	0.02229	0.00	0.02077	0.00	0.02219	0.00	0.02408	0.00	0.03322
H_m (100)	0.29	0.00622	0.76	0.00529	0.90	0.00490	0.43	0.00536	0.19	0.00724
H_m (300)	16.7	0.00085	18.1	0.00066	18.2	0.00065	19.1	0.00054	16.1	0.00063
H_m' (50)	0.00	0.02229	0.00	0.01998	0.00	0.01928	0.00	0.01893	0.00	0.01976
H_m' (100)	0.29	0.00622	0.76	0.00500	1.14	0.00461	1.19	0.00445	1.24	0.00429
H_m' (300)	16.7	0.00085	19.0	0.00069	19.8	0.00061	20.5	0.00053	21.5	0.00046

5 Conclusions

We have proposed two entropy-based diversity measures for evaluating population diversity in the GA applied to the TSP. The first one, denoted as H_m , is equivalent to the entropy rate of an m -th order Markov information source where the probability of a vertex appearing in the population (tours) is assumed to depend on the sequence of m precedent vertices. However, H_m is not a meaningful population diversity measure for a greater value of m due to the lack of

available samples (sequences of vertices) necessary for computing it. To alleviate this problem, we have proposed another variant of entropy-based diversity measure H_m' . The experimental results show that H_m and in particular H_m' with the appropriate values of m evaluate population diversity more appropriately than does the commonly used entropy-based diversity measure H^{ind} that does not take into account dependencies in the sequence of vertices in the population.

Future work will be devoted to (1) an more efficient implementation for the computation of ΔH , (2) an adaptation of the value of m for two entropy-based diversity measures during the search, and (3) possible extensions of the presented diversity measures to other permutation based problems.

Acknowledgement. This work was partially supported by Grant-in-Aid for Scientific Research of Japan: No. 22700231.

References

1. Maekawa, K., Mori, N., Tamaki, H., Kita, H., Nishikawa, Y.: A genetic solution for the traveling salesman problem by means of a thermodynamical selection rule. In: Proceedings of 3rd IEEE Conference on Evolutionary Computation, pp. 529–534 (1996)
2. Mori, N., Kita, H., Nishikawa, Y.: Adaptation to a Changing Environment by Means of the Feedback Thermodynamical Genetic Algorithm. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 149–158. Springer, Heidelberg (1998)
3. Nagata, Y.: Fast EAX Algorithm Considering Population Diversity for Traveling Salesman Problems. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 171–182. Springer, Heidelberg (2006)
4. Nagata, Y., Kobayashi, S.: Powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS Journal on Computing (in press) (published as an article in advance)
5. Tsai, H., Yang, J., Tsai, Y., Kao, C.: An evolutionary algorithm for large traveling salesman problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34(4), 1718–1729 (2004)
6. Tsujimura, Y., Gen, M.: Entropy-based genetic algorithm for solving tsp. In: Proceedings of 2nd International Conference on Knowledge-Based Intelligent Electronic Systems, pp. 285–290. IEEE (1998)
7. Vallada, E., Ruiz, R.: Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. Omega 38(1), 57–67 (2010)
8. Wang, Y., Lü, Z., Hao, J.-K.: A Study of Multi-parent Crossover Operators in a Memetic Algorithm. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part I. LNCS, vol. 6238, pp. 556–565. Springer, Heidelberg (2010)
9. Yao, X.: An empirical study of genetic operators in genetic algorithms. Microprocessing and Microprogramming 38(1-5), 707–714 (1993)
10. Zhang, C., Su, S., Chen, J.: Efficient Population Diversity Handling Genetic Algorithm for QoS-Aware Web Services Selection. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 104–111. Springer, Heidelberg (2006)

Investigating Monte-Carlo Methods on the Weak Schur Problem

Shalom Eliahou¹, Cyril Fonlupt², Jean Fromentin¹, Virginie Marion-Poty²,
Denis Robilliard², and Fabien Teytaud²

¹ Univ Lille Nord de France
ULCO, LISIC, BP 719
F-62228 Calais, France

{fonlupt,poty,robilliard,teytaud}@lisisic.univ-littoral.fr

² Univ Lille Nord de France
ULCO, LMPA, BP 699
F-62228 Calais, France

{eliahou,fromentin}@lmpa.univ-littoral.fr

Abstract. Nested Monte-Carlo Search (NMC) and Nested Rollout Policy Adaptation (NRPA) are Monte-Carlo tree search algorithms that have proved their efficiency at solving one-player game problems, such as morpion solitaire or sudoku 16x16, showing that these heuristics could potentially be applied to constraint problems. In the field of Ramsey theory, the *weak Schur number* $WS(k)$ is the largest integer n for which there exists a partition into k subsets of the integers $[1, n]$ such that there is no $x < y < z$ all in the same subset with $x + y = z$. Several studies have tackled the search for better lower bounds for the Weak Schur numbers $WS(k)$, $k \geq 4$. In this paper we investigate this problem using NMC and NRPA, and obtain a new lower bound for $WS(6)$, namely $WS(6) \geq 582$.

1 Introduction

Nested Monte-Carlo Search (NMC) [5] and the recent Nested Rollout Policy Adaptation (NRPA) [17] are Monte-Carlo tree search algorithms that have proved their efficiency at solving constrained problems, although mainly in the AI field (e.g. sudoku 16x16, morpion solitaire game, Same Game). Within the field of Ramsey theory, challenging problems are the search for the Van der Waerden numbers [4] and for the Schur numbers [10], or the search for better lower or higher bounds for these numbers. Finding new lower bounds can be tackled with computational tools, by constructing a mathematical object that exhibits the required properties. In general this construction implies exploring a heavily constrained combinatorial space of huge dimension. In this paper we investigate the search for better lower bounds for the so-called Weak Schur numbers, using NMC and NRPA. First, we present the definition of Weak Schur numbers, next we recall the principles of the two Monte-Carlo search algorithms that we used, then we compare the results obtained, notably the discovery of a new lower bound $WS(6) \geq 582$, before concluding.

2 Weak Schur Numbers

The Weak Schur numbers originate from two Ramsey theory theorems, dating back to the first half of the 20th century. We recall their definition and the current state of knowledge on this topic, before presenting experimental data that motivate the choice of Monte-Carlo search methods.

2.1 Mathematical Description

First, we explain the concept of Schur numbers. A set P of integers containing no elements x, y, z with $x + y = z$ is called *sum-free*. A theorem of Schur [18] states that, given $k \geq 1$, there is a largest integer n for which the integer interval set $[1, n]$ (i.e. $\{1, 2, \dots, n\}$) admits a partition into k sum-free sets. The largest such integer n is called the k -th Schur number, and is denoted by $S(k)$.

As a somewhat weaker notion than sum-free, a set P of integers containing no *pairwise distinct* elements x, y, z with $x + y = z$ is called *weakly sum-free*. A result similar to that of Schur was shown by Rado [13] : given $k \geq 1$, there is a largest integer n for which the interval set $[1, n]$ admits a partition into k weakly sum-free sets. This largest such integer n is called the k -th Weak Schur number, and is denoted by $WS(k)$.

For example, in the case $k = 2$, a weakly sum-free partition of the first 8 integers is provided by:

$$\{1, 2, 3, 4, 5, 6, 7, 8\} = \{1, 2, 4, 8\} \cup \{3, 5, 6, 7\}.$$

It is straightforward to verify, with exhaustive search, that increasing the interval to $[1, 9]$ yields a set that does not admit any weakly sum-free partition into 2 sets, thus we have $WS(2) = 8$.

2.2 State of the Art

The exact values of $WS(k)$ (and of $S(k)$) are only known up to $k = 4$:

- $WS(1) = 2$ and $WS(2) = 8$ are easily verified
- $WS(3) = 23$, $WS(4) = 66$ were shown by exhaustive computer search in [2].

For the general case $k \geq 5$, known results are a lower bound from Abbott and Hanson [1] and an upper bound by Bornshtein [3]:

$$c89^{k/4} \leq S(k) \leq WS(k) \leq \lfloor k!ke \rfloor$$

with c a small positive constant.

Some special cases are better known through experimental studies. In [2] it was shown that: $WS(5) \geq 189$, while a much older note by Walker [19] claimed, without proof, that $WS(5) = 196$. More recently, [8] provided a weakly sum-free partition of the set $[1, 196]$ in 5 sets, confirming Walker's claim that $WS(5)$ is at least as large as 196, and also gave a weakly sum-free partition of $[1, 572]$ in

6 sets, implying $WS(6) \geq 572$. In [9] the lower bound for $WS(6)$ was pushed to $WS(6) \geq 574$ using meta-heuristic search. This result was superseded by [12] establishing a current best bound at $WS(6) \geq 581$.

This result in [12] was obtained with a constraint solver. Extra constraints, nicknamed *streamliners*, were added to the problem in order to reduce the search space size, with the (unproven) assumption that optimal solutions were preserved. Some of these streamliners were taken either from the previous studies on this subject [8,9] or from the literature on the related (non weak) Schur problem [10].

2.3 Methodology and Experimental Data

In this paper, we consider using AI methods, namely variants of Monte Carlo Tree Search (MCTS) in order to tackle the search of better lower bounds for $WS(5)$ and $WS(6)$. The motivation behind this method choice is supported by a study of weakly sum-free 3-partitions of $[1, WS(3)]$ and 4-partitions $[1, WS(4)]$. We now explain this last case.

A weakly sum-free 4-partition (i.e. partition into 4 subsets) of $[1, n]$ can be coded by a word, $w = a_1a_2 \dots a_n$, where the letters $\{a_i\}$ are in the symbol set $\{1, 2, 3, 4\}$ and $a_i = j$ means that integer i is in the subset j . For example, the word 1, 1, 2, 1, 3, 4 encodes the partition $\{\{1, 2, 4\}, \{3\}, \{5\}, \{6\}\}$. Up to symmetry, we can systematically enumerate in lexicographic order all words associated to partitions that are *terminal*, i.e. such that it is not possible to extend the partition by placing the next successive integer while keeping the weakly sum-free property. An example of such a word is given by $w = 112233444444442333332221$, associated to the terminal partition below:

$$\left\{ \begin{array}{l} \{1, 2, 24\}, \\ \{3, 4, 15, 21, 22, 23\}, \\ \{5, 6, 16, 17, 18, 19, 20\}, \\ \{7, 8, 9, 10, 11, 12, 13, 14\} \end{array} \right\}$$

Clearly it is not possible to add 25 in any subset of this partition, thus it is terminal, and we also call w a terminal word. An exhaustive computer enumeration shows that there are exactly 536 995 391 721 terminal words that correspond to terminal 4-partitions. To reduce the size of the data, we cluster them in groups of 1 billion consecutive words in lexicographic order, and then plot the mean and maximal word length (including solutions of maximal length $WS(4) = 66$) from each group in Figure 1. Mean and maximal length exhibit a covariance correlation of 0.7, thus they are somewhat correlated, and the same behavior is also observed for 3-partitions. This means that good samples on average, can lead to better solutions¹. Assuming that this good property also holds for 5-partitions

¹ Note that, quite interestingly, this is not the case for standard Schur Numbers, at least $S(3)$ and $S(4)$.

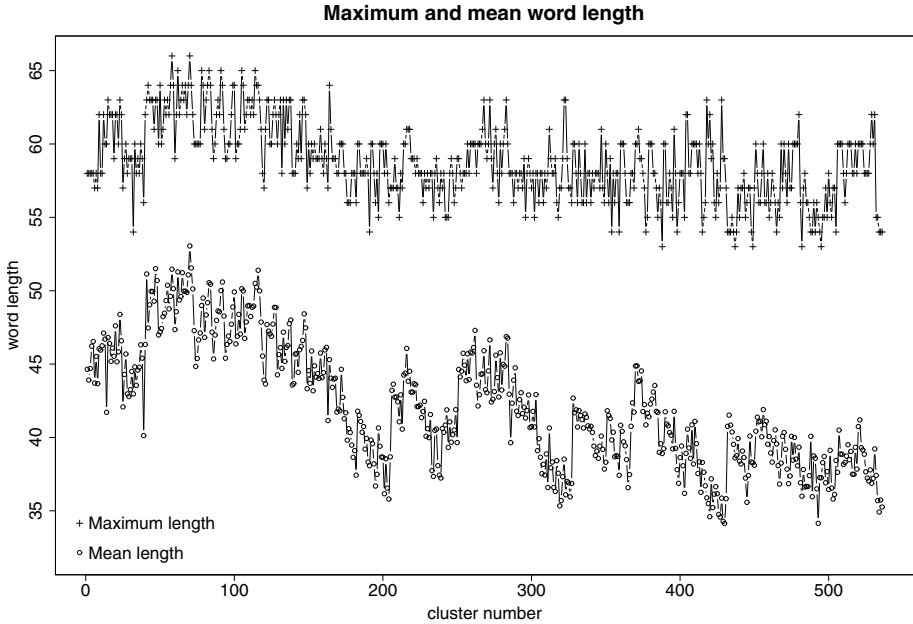


Fig. 1. Mean and maximal length of terminal word groups associated to 4-partitions, listed in lexicographic order

and 6-partitions, Monte Carlo sampling appears as a promising method for searching lower bounds for $WS(5)$ and $WS(6)$. It is also noticeable that optimal solutions are clustered in the beginning with respect to the lexicographic ordering.

3 Methods

In this section we present the two Monte-Carlo heuristics that were used for our experiments.

3.1 Nested Monte-Carlo Search

The Nested Monte-Carlo Search (NMC) algorithm [5] is a tree search algorithm. The basic idea is to incrementally build a solution with the particularity that each decision is based on the result of lower level calls of the algorithm. The lowest level, the level 0, is a single Monte-Carlo payout. A Monte-Carlo payout is a sequence for which each remaining decision is made randomly, until no more possible decision can be made. For higher levels > 0 , all possible decisions are tried (i.e. recursively explored) and the branch of the search tree associated to the best result is chosen as the new decision.

The pseudo-code of NMC is presented in Algorithm 1, where:

- a *position* describes the state of the solution which is being constructed (the root position is empty, no decision having been chosen already). Here a solution is a weakly sum-free k -partition. Notice that *position* is always passed as argument by copy, and never by reference.
- `play(position, d)` is a function that returns the new position obtained after having performed decision d relatively to *position*. In our Weak Schur numbers problem implementation, a decision consists in choosing in which subset of the partition to place the next integer. Notice that the selection of the next integer is deterministic (see Sect. 4.1) and is not part of the decision.
- `MonteCarlo(position)` is a function that completes the *position* by playing random decisions, until the k -partition is terminal (see Sect. 2.3). It returns a 2-tuple: the evaluation of the terminal partition, and the sequence of decisions that were made to obtain it, called *payout* in Algorithm 1. As integers are not always played in successive order, possibly leaving "holes" in the partition, its evaluation is the greatest integer L such that the set of consecutive integers $[1, L]$ is in the partition, i.e. we stop counting at the first hole.

A level 1 maximization example is presented in Figure 2. The leftmost tree illustrates the start. A Monte-Carlo payout is performed for all 3 possible decisions. At the end of each payout a reward is given, and the decision with the best reward is chosen. This new decision is performed leading to a new state, and will never be backtracked. The process is repeated.

NMC provides a good compromise between exploration and exploitation. It is particularly efficient for one-player games and gives good results even without domain knowledge. However, the results can be improved by the addition of heuristics [15].

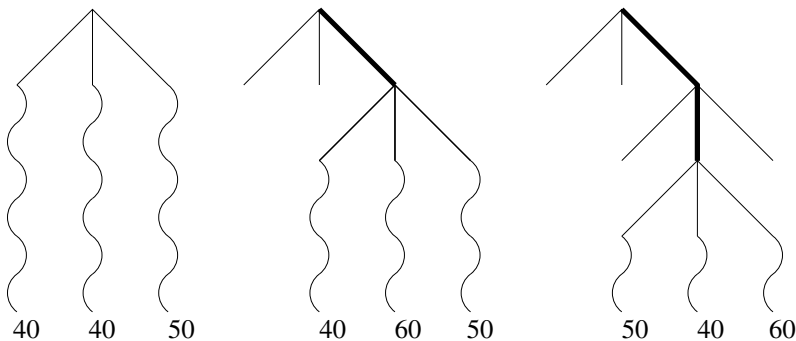


Fig. 2. This figure illustrates three successive decision steps of a level 1 search for a maximization problem, rewards being given at the bottom of branches. Level 1 exploration is represented in thin lines, level 0 Monte-Carlo payouts are shown with wavy lines, decisions are represented with bold lines.

Algorithm 1. Nested Monte-Carlo [5]

```

function NMC(position, level) :
  best_score  $\leftarrow -\infty$ 
  best_playout  $\leftarrow \{\}$ 
  while not solution completed do
    if level = 1 then
      // collapse level 1 and 0
      // score_max, p_max: score and playout associated to best branch
      (score_max, p_max)  $\leftarrow \arg \max_d(\text{MonteCarlo}(\text{play}(\text{position}, d)))$ 
    else
      (score_max, p_max)  $\leftarrow \arg \max_d(\text{NMC}(\text{play}(\text{position}, d), \text{level} - 1))$ 
    end if
    if score_max > best_score then
      best_score  $\leftarrow \text{score\_max}$ 
      best_playout  $\leftarrow \text{p\_max}$ 
    end if
    d_best  $\leftarrow$  first decision in best_playout
    position  $\leftarrow \text{play}(\text{position}, d\_best)$ 
  end while
  return (score(position), best_playout)

```

3.2 Nested Rollout Policy Adaptation

The Nested Rollout Policy Adaptation (NRPA) algorithm [17] is inspired by NMC. As in NMC, each level of the algorithm calls a lower level, and each level returns the best score and best playout sequence found at this level. Two main differences are:

- The playout, or rollout, policy is no more a standard Monte-Carlo, but based on a learned policy.
- There is no systematic evaluation of every possible decision associated to any given position, a given number of recursive calls being made instead. So all decisions may not be explored, depending on the policy-based sampling.

The NRPA algorithm is presented in Algorithm 2, where:

- *policy* is a vector of weights associated to decisions, $\text{policy}[x]$ is used to compute the probability of choosing decision x . The initial policy corresponds to a classic Monte-Carlo playout (i.e. equiprobability).
- $\text{code}(p, d_i)$ returns a domain-specific integer associated to the decision d_i leading from position p to its i^{th} child in the search tree. This integer serves as index in the policy vector. As stated in [17], the function $\text{code}(p, d_i)$ should preferably be bijective.
- $\text{MonteCarloPolicy}(\text{position}, \text{policy})$ does a playout that differs from classic Monte-Carlo, by choosing the decision not equiprobably but using *policy*. During the playout, a decision is chosen proportionally to $\exp(\text{policy}[\text{code}(\text{position}, i)])$. This function returns a 2-tuple: the evaluation of the terminal position, and the sequence of decisions that were made to obtain it.

Algorithm 2. Nested Rollout Policy Adaptation [17]

```

function NRPA (level,policy)
  if level = 0 then
    // complete position by choosing decisions with probability proportional
    // to exp(policy[code(position, i)])
    return MonteCarloPolicy(position,policy)
  else
    best_score ← -∞
    for N iterations do
      (score,playout) ← NRPA (level - 1, policy)
      if score ≥ best_score then
        best_score ← score
        best_playout ← playout
      end if
      policy ← Adapt(policy,best_playout)
    end for
  end if
  return (best_score, best_playout)

function Adapt (policy,playout)
  position ← root
  policy' ← policy
  for each decision d in playout do
    // increase probability of best decision
    policy'[code(position, d)] ← policy'[code(position, d)] + α
    // compute normalization factor
    z ← ∑i exp(policy[code(position, i)]) over all possible decisions i at position
    for each decision i at position do
      policy'[code(position, i)] ← policy'[code(position, i)] -
        α × exp(policy[code(position, i)]) / z
    end for
    // prepare to iterate on next decision in best playout
    position ← play(position, d)
  end for
  return policy'

```

In the pseudo-code, the variables *decision* and *position*, and the function `play(position, decision)` have the same meaning as for the NMC algorithm, see Sect. 3.1.

The NRPA learning is comparable in its motivations with that done in the Upper Confidence Tree search algorithms [7,14,16] and consists in increasing probabilities of good decisions while decreasing those for bad decisions. As pointed out in [6], a drawback of the learning is the convergence to local optima.

4 Experimental Results

First, we detail the adaptation of the two Monte-Carlo heuristics to the problem at hand, then we report the results obtained on the $WS(5)$ and $WS(6)$ problems.

4.1 Experimental Settings

It has been shown in [11,15] that it is often possible to improve the performance of NMC by adding specific knowledge. A usual way is to bias the probability of decision choice, based of expert knowledge. Having such a probability, rather than a deterministic expert-based choice, may be important to keep diversity [14].

As seen in Section 3.2, the NRPA algorithm learns a policy. We encountered difficulties with the `(p, di)` function. As stated in Sect. 2.3, the number of possible positions (i.e. weakly sum-free partitions under construction) is already too huge for $WS(4)$ to allow the storage of a policy with a bijective code function. It is even more huge when exploring partitions for $WS(5)$ and $WS(6)$. Thus we have chosen to use a 2-dimensional vector of size $WS(k) \times k$, that stores for each integer and each subset the weight associated to the probability of putting this integer in this subset. So the function `(position, decision)` considers, as *position* argument, only the current integer to be placed, rather than the whole partition information. So `code` is not bijective. One could perhaps consider using a larger storage with a hashing function to take into account more information from the partition.

We had disappointing results with the standard NRPA algorithm, thus we tried to add expert knowledge. Intuitively, adding expert knowledge in the rollout is harder, as it needs to coexist with the learning and not to interfere with it. To implement this, instead of always making decisions according to the learned policy, we choose with probability $\frac{1}{2}$ a decision according only to the expert knowledge. This allows to take into account both learning and expert knowledge.

For both NMC and NRPA algorithms, a natural implementation is to put integers in the ascending order (i.e. we look for placing integer 1 in a subset, then we want to place integer 2 and so on). Another possibility, which comes from the constraint programming field, is to choose the most constrained integer as the next integer to place. This has the benefit of cutting dead branches of the search tree earlier, thus focusing the search on more promising ones. This significantly improves the search.

Expert knowledge used in our experiments is based on the known optimal partitions of $WS(4)$ and $WS(5)$: as pointed out in [8,9], all $WS(4)$ solutions are extensions of 2 of the 3 optimal partitions for $WS(3)$. Assuming that this property may hold, more or less strictly, for k -partitions with a larger k , we fix the 23 first integers, with an exception for 16 which can be placed either in subset 1 or 3. From the same knowledge we forbid integers lower than $WS(4) + 1 = 67$ from the last two subsets (subsets 5 and 6). In [12], the best known solution so far showing $WS(6) \geq 581$ has integer 196 as the smallest member of the sixth subset, thus we also ban all integers lower than 196 from subset 6. The lowest allowed integer in the last subset is then greater than or equal to 196. It is

very noticeable, as pointed out by [12], that subsets of partitions often contain intervals of consecutive numbers. Thus we add a 90% probability that an integer is put in the first subset (in increasing order) where its immediate predecessor or successor already stays.

Another knowledge from [8] is, for subsets 5 and 6, to try to build sequences of shape $\{a\} \cup [a+2, \dots, 2a+1]$, with a the first (lowest) integer of the subset. It then seems interesting to place integer $a+1$ in the first subset. We add a 90% probability to play out each of these decisions (i.e. each integer placement).

4.2 Results

We ran 30 independent executions of a level 3 Nested Monte-Carlo search, with embedded expert knowledge as explained above.

Table 1. A weakly sum-free 6-partition of $[1, 582]$

$$\left\{ \begin{array}{l} \{ 1-2, 4, 8, 11, 22, 25, 31, 40, 50, 63, 68, 73, 82, 87, 92, 97, 116, 121, 133, 139, 149, \\ 154, 159, 177, 182, 187, 192, 197, 252, 304, 342, 370, 394, 407, 412, 417, 435, \\ 440, 445, 450, 455, 464, 469, 474, 479, 488, 493, 502, 507, 521, 526, 531, 536, \\ 541, 554, 564, 569, 582\}, \\ \{ 3, 5-7, 19, 21, 23, 37, 51-53, 64-66, 79-81, 93-95, 109-111, 122-124, 136-138, \\ 150-152, 167-168, 179-181, 193-195, 368, 395-397, 408-410, 424-425, 437-439, \\ 451-453, 465-467, 480-482, 495-497, 512, 523-525, 537-539, 551-553, 566-568, \\ 579-581\}, \\ \{ 9-10, 12-18, 20, 54-62, 103-108, 140-148, 183-186, 188-191, 398-406, 441-444, \\ 446-449, 486-487, 490, 492, 494, 527-530, 532-535, 570-578\}, \\ \{ 24, 26-30, 32-36, 38-39, 41-49, 98-102, 153, 155-158, 160-166, 169-176, 178, 292, \\ 411, 413-416, 418-423, 426-434, 436, 540, 542-550, 555-563, 565\}, \\ \{ 67, 69-72, 74-78, 83-86, 88-91, 96, 112-115, 117-120, 125-132, 134-135, 454, \\ 456-463, 468, 470-473, 475-478, 483-485, 489, 491, 498-501, 503-506, 508-511, \\ 513-520, 522\}, \\ \{ 196, 198-251, 253-291, 293-303, 305-341, 343-367, 369, 371-392\} \end{array} \right\}$$

A weakly sum-free 6-partition of the integer set $[1, 582]$ was found each time, after an averaged number of 624 600 Monte-Carlo sampling, taking on average less than a minute per run on a standard PC. The 6-partitions obtained were usually different on each run. This result yields a new best lower bound for $WS(6)$, namely $WS(6) \geq 582$, and we were not able to obtain a weakly sum-free 6-partition of a larger interval. An example partition showing $WS(6) \geq 582$ is given in Table 1. As for $WS(5)$ we could not increase the current bound of 196, which is its conjectured exact value.

The best result obtained with our knowledge-aware NRPA was 571, while the basic algorithm, without expert knowledge, never attained 300. It remains to experiment if increasing the policy storage could improve on the results.

5 Conclusion

Being now very popular in the AI for games field, Monte-Carlo Tree Search techniques have the ability to address a wider domain. As often, obtaining a successful application relies on the introduction of expert knowledge to foster the potential of the method. It is to be noticed that it is rather easy to embed such knowledge in NMC by simply biasing the sampling probability of solutions. In this study we obtained a new bound for the 6th Weak Schur number, using Cazenave’s NMC algorithm and several streamliners to further constrain the problem and reduce its state space. We recall that using these streamliners may constrain too much the search space, thus maybe preventing the discovery of even better bounds.

This work also serves as a validation benchmark for NMC and NRPA. Both heuristics are unable to solve the problem without the addition of expert knowledge. However, it feels more awkward to bias the sampling probabilities in NRPA, since this should normally be done by the policy learning. In our experiments the largest partitions obtained with NRPA have been slightly smaller than those given by NMC. This may be due to the tendency of NRPA to converge to local optima. As possible future works, it may be interesting for the NRPA algorithm to find either a better *code* function, or another way to incorporate expert knowledge. Hybridization of NMC/NRPA with a constraint solver could also be a promising approach.

References

1. Abbott, H., Hanson, D.: A problem of Schur and its generalizations. *Acta Arith.* 20, 175–187 (1972)
2. Blanchard, P.F., Harary, F., Reis, R.: Partitions into sum-free sets. *Integers* 6 A7 (2006)
3. Bornsztein, P.: On an extension of a theorem of Schur. *Acta Arith.* 101, 395–399 (2002)
4. Brown, T., Landman, B.M., Robertson, A.: Note: Bounds on some van der waerden numbers. *J. Comb. Theory Ser. A* 115(7), 1304–1309 (2008)
5. Cazenave, T.: Nested Monte-Carlo search. In: Boutilier, C. (ed.) *IJCAI*, pp. 456–461 (2009)
6. Cazenave, T., Teytaud, F.: Application of the Nested Rollout Policy Adaptation Algorithm to the Traveling Salesman Problem with Time Windows. In: Hamadi, Y., Schoenauer, M. (eds.) *LION 2012. LNCS*, vol. 7219, pp. 42–54. Springer, Heidelberg (2012)
7. Drake, P.: The last-good-reply policy for Monte-Carlo go. *ICGA Journal* 32(4), 221–227 (2009)
8. Eliahou, S., Marín, J.M., Revuelta, M.P., Sanz, M.I.: Weak Schur numbers and the search for G. W. Walker’s lost partitions. *Computer and Mathematics with Applications* 63, 175–182 (2012)
9. Robilliard, D., Fonlupt, C., Marion-Poty, V., Boumaza, A.: A Multilevel Tabu Search with Backtracking for Exploring Weak Schur Numbers. In: Hao, J.-K., Legrand, P., Collet, P., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) *EA 2011. LNCS*, vol. 7401, pp. 109–119. Springer, Heidelberg (2012)

10. Fredricksen, H., Sweet, M.M.: Symmetric sum-free partitions and lower bounds for Schur numbers. *Electr. J. Comb.* 7 (2000)
11. Gelly, S., Silver, D.: Combining online and offline knowledge in uct. In: Ghahramani, Z. (ed.) *ICML. ACM International Conference Proceeding Series*, vol. 227, pp. 273–280. ACM (2007)
12. Le Bras, R., Gomes, C.P., Selman, B.: From streamlined combinatorial search to efficient constructive procedures. In: *Proceedings of the 15th International Conference on Artificial Intelligence, AAAI 2012* (2012)
13. Rado, R.: Some solved and unsolved problems in the theory of numbers. *Math. Gaz.* 25, 72–77 (1941)
14. Rimmel, A., Teytaud, F.: Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., Neri, F., Preuss, M., Togelius, J., Yannakakis, G.N. (eds.) *EvoApplications 2010, Part I. LNCS*, vol. 6024, pp. 201–210. Springer, Heidelberg (2010)
15. Rimmel, A., Teytaud, F., Cazenave, T.: Optimization of the Nested Monte-Carlo Algorithm on the Traveling Salesman Problem with Time Windows. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Urquhart, N., Uyar, A.Ş. (eds.) *EvoApplications 2011, Part II. LNCS*, vol. 6625, pp. 501–510. Springer, Heidelberg (2011)
16. Rimmel, A., Teytaud, F., Teytaud, O.: Biasing Monte-Carlo Simulations through RAVE Values. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) *CG 2010. LNCS*, vol. 6515, pp. 59–68. Springer, Heidelberg (2011)
17. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo tree search. In: Walsh, T. (ed.) *IJCAI*, pp. 649–654. *IJCAI/AAAI* (2011)
18. Schur, I.: Über die kongruenz $x^m + y^m \equiv z^m \pmod{p}$. *Jahresbericht der Deutschen Mathematiker Vereinigung* 25, 114–117 (1916)
19. Walker, G.: A problem in partitioning. *Amer. Math. Monthly* 59, 253 (1952)

Multi-objective AI Planning: Comparing Aggregation and Pareto Approaches

Mostepha R. Khouadjia¹, Marc Schoenauer¹,
Vincent Vidal², Johann Dréo³, and Pierre Savéant³

¹ TAO Project, INRIA Saclay & LRI Paris-Sud University, Orsay, France
{mostepha-redouane.khouadjia,marc.schoenauer}@inria.fr

² ONERA-DCSD, Toulouse, France
Vincent.Vidal@onera.fr

³ THALES Research & Technology, Palaiseau, France
{johann.dreo,pierre.saveant}@thalesgroup.com

Abstract. Most real-world Planning problems are multi-objective, trying to minimize both the makespan of the solution plan, and some cost of the actions involved in the plan. But most, if not all existing approaches are based on single-objective planners, and use an aggregation of the objectives to remain in the single-objective context. *Divide-and-Evolve* is an evolutionary planner that won the temporal deterministic satisficing track at the last International Planning Competitions (IPC). Like all Evolutionary Algorithms (EA), it can easily be turned into a Pareto-based Multi-Objective EA. It is however important to validate the resulting algorithm by comparing it with the aggregation approach: this is the goal of this paper. The comparative experiments on a recently proposed benchmark set that are reported here demonstrate the usefulness of going Pareto-based in AI Planning.

1 Introduction

Most, if not all, classical AI planning solvers are single-objective. Given a planning domain (a set of predicates that describe the state of the system, and a set of actions with their pre-requisites and effects), and an instance of this domain (a set of objects on which the predicates are instantiated into boolean atoms, an initial state and a goal state), classical planners try to find, among the set of all feasible plans (sequences of actions such that, when applied to the initial state, the goal state becomes true), the one with the minimal number of actions (STRIP planning), or with the smallest cost (actions with costs) or with the smallest makespan (temporal planning, where actions have durations and can be applied in parallel). A detailed introduction to (single-objective) AI planning can be found in [1]. It is clear, however, that most planning problems are in fact multi-objective, as the optimal solution in real-world problems often involve some trade-off between makespan and cost [2]. A few trials have been made to turn some classical planners into multi-objective optimizers, either using some

twist in PDDL 2.0¹ to account for both makespan and cost [3–5], or using the new hooks for several objectives offered by PDDL 3.0 [6]. However, all these approaches are based on a linear aggregation of the different objectives, and were not pursued, as witnessed by the new “net-benefit” IPC track, dedicated to aggregated multiple objectives, that took place in 2006 [7] and 2008 [8], . . . but was canceled in 2011 due to a lack of entries.

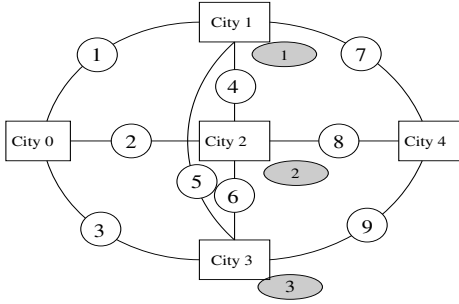
In the framework of Evolutionary Algorithms (EAs), Pareto multi-objective optimization has received a lot of attention [9], and any single-objective EA can “easily” be turned into a multi-objective EA, by modifying the selection step (and possibly adding some archiving mechanism). Unfortunately, there exist very few evolutionary AI planners. Directly evolving plans, as in [10], obviously does not scale up, and was never extended to multi-objective setting. Hence, as far as we are aware of, the state-of-the-art in evolutionary AI planning is the previous work of some of the authors, *Divide-and-Evolve* (DAE). DAE evolves variable length sequences of states, that start with the problem initial state and end at the problem goal state. DAE relies on a classical embedded planner to sequentially reach each state of the sequence from the previous one. The concatenation of all plans given by the embedded planner is a solution plan of the original problem. DAE can thus solve all types of planning problems that the embedded planner can solve. Proof-of-concept for DAE was obtained with DAE_{CPT} [11], where the embedded planner was CPT, an exact planner [12] – and already included some small multi-objective experiments. Since then, the DAE paradigm has evolved, and YAHSP a sub-optimal lookahead strategy planning system [13] is now used as the embedded planner [14], and DAE_{YAHSP} has reached state-of-the-art results in all planning domains [15], winning the temporal deterministic satisficing track at the last IPC in 2011².

The very preliminary work in [11] regarding multi-objective optimization has also been recently revisited with DAE_{YAHSP}. The lack of existing benchmark suite for multi-objective planning led us to extend the small toy problem from [11] into a tunable benchmark domain, on which different multi-objectivization of DAE_{YAHSP} (MO-DAE_{YAHSP}) were compared [16]. But because the only other approach in AI Planning is the aggregation of the objectives, there is a need to compare the multi-objective approach for DAE_{YAHSP} with the single-objective approach based on the linear aggregation of the objectives: this is the purpose of the present work. Section 2 will briefly present planning problems and DAE_{YAHSP} in the single-objective setting. In Section 3, the multi-objective context will be introduced. The multi-objective benchmark suite will be presented, and the multi-objectivization of DAE_{YAHSP} will be detailed: because YAHSP is a single-objective planner³, but can be asked to optimize either the makespan or the cost, specific strategies had to be designed regarding how it is called within

¹ Planning Domain Definition Language, a dedicated language for the description of planning problems, set up for the International Planning Competitions (IPC).

² See <http://www.plg.inf.uc3m.es/ipc2011-deterministic>

³ Note that it seems difficult, if at all possible, to adapt it directly to multi-objective optimization, as it uses very different strategies for the makespan and the cost.



Flight durations are attached to the possible routes (white circles), costs/risks are attached to landing in the central cities (grey circles). Four sets of values given on the right. Default values in the first (“Lin.”) column in the Table.

Dur./edge	Lin.	Cvx	Ccve
1	2	2	2
2	4	4	3
3	6	6	4
4	3	3	1
5	5	5	2
6	3	3	1
7	2	2	2
8	4	4	3
9	6	6	4
Cost/city			
1	30	30	30
2	20	11	29
3	10	10	10

Fig. 1. Schematic view, and 3 instances, of simple MULTIZENO benchmark

MO-DAE_{YAHSP}. Section 4 describes the experimental settings, detailing in particular the implementation of the aggregation approach for DAE_{YAHSP} and the intensive parameter tuning that was performed for all competing algorithms using the off-line problem-independent tuner PARAMILS [17]. The results will be presented and discussed in Section 5, and as usual, conclusion and hints about on-going and further work will be given in Section 6.

2 Single-objective Background

AI Planning Problems: A planning domain D is defined by a set of object types, a set of predicates, and a set of possible actions. An instance is defined by a set of objects of the domain types, an initial state, and a goal state. A predicate that is instantiated with objects is called an atom, and takes a boolean value. For a given instance, a state is defined by assigning values to all possible atoms. An action is defined by a set of *pre-conditions* (atoms) and a set of *effects* (changing some atom values): the action can be executed only if all pre-conditions are true in the current state, and after an action has been executed, the state is modified: the system enters a new state. The goal is to find a plan (sequence of actions) such that it leads from the initial state to the goal state, and minimizes either the number or costs of actions, or the makespan in the case of temporal planning where actions have durations and can be run in parallel.

A simple temporal planning problem in the domain of logistics (inspired by the well-known ZENO problem of IPC series) is given in Figure 1, and will be the basis of the benchmark used in this work: the problem involves cities, passengers, and planes (object types). Passengers can be transported from one city to another (action `fly`), following the links on the figure. One plane can only carry one passenger at a time from one city to another, and the flight duration (number on the link) is the same whether or not the plane carries a passenger (this defines

the *domain* of the problem). In the simplest non-trivial *instance* of such domain, there are 3 passengers and 2 planes. In the initial state, all passengers and planes are in `city 0`, and in the goal state, all passengers must be in `city 4`. In the default case labeled “Lin.” in the table right (forget about the costs for now), the not-so-obvious makespan-optimal solution has a total makespan of 8 and is left as a teaser for the reader.

Divide-and-Evolve: Let $\mathcal{P}_D(I, G)$ denote the planning problem defined on domain D with initial state I and goal state G . In order to solve $\mathcal{P}_D(I, G)$, the basic idea of DAE_X is to find a sequence of states S_1, \dots, S_n , and to use some embedded planner X to solve the series of planning problems $\mathcal{P}_D(S_k, S_{k+1})$, for $k \in [0, n]$ (with the convention that $S_0 = I$ and $S_{n+1} = G$). The generation and optimization of the sequence of states $(S_i)_{i \in [1, n]}$ is driven by an evolutionary algorithm. The fitness of a sequence is computed using the embedded planner X , that is called in turn on each of the sub-problems $\mathcal{P}_D(S_k, S_{k+1})$. The concatenation of the corresponding plans (possibly compressed to take into account possible parallelism in the case of temporal planning) is a solution of the initial problem. In case one sub-problem cannot be solved by the embedded solver, the individual is said *unfeasible* and its fitness is highly penalized in order to ensure that unfeasible individuals are always selected after feasible ones. A thorough description of DAE_X can be found in [15]. The rest of this section will briefly recall the evolutionary parts of DAE_X .

An individual in DAE_X is a variable-length list of partial states of the given domain (similar to the goal state), and a partial state is a variable-length list of atoms (instantiated predicates). The initialization procedure is based on a heuristic estimation, for each atom, of the earliest time from which it can become true [18]. Furthermore, most existing planners (and this is true for CPT and YAHSP, that have been used within DAE) start by computing some partial mutual exclusion between possible atoms: this information is also used to reduce the search space in DAE_X , whenever possible. An individual in DAE_X is hence a variable-length time-consistent sequence of partial states, and each partial state is a variable-length list of atoms that are not pairwise mutually exclusive.

Crossover and mutation operators are applied with respective user-defined probabilities p_{Cross} and p_{Mut} . They are defined on the DAE_X representation in a straightforward manner - though constrained by the heuristic chronology and the partial mutex relation between atoms. **One-point crossover** is adapted to variable-length representation: both crossover points are independently chosen, uniformly in both parents. Only one offspring is kept, the one that respects the approximate chronological constraint on the successive states. **Four different mutation operators** are included, and operate either at the individual level, by adding (`addState`) or removing (`delState`) a state, or at the state level by adding or modifying (`addChangeAtom`) or removing (`delAtom`) some atoms in a uniformly chose state. The choice among these operators is made according to user-defined relative weights (named `w-mutationname` - see Table 1).

3 Multi-objective Background

3.1 Pareto-Based Multi-objective Divide-and-Evolve

Two modifications of $\text{DAE}_{\text{YAHSP}}$ are needed to turn it into an EMOA: use some multi-objective selection engine in lieu of the single-objective tournament selection that is used in the single-objective context; and compute the value of both objectives (makespan and cost) for both individuals. The former modification is straightforward, and several alternatives have been experimented within [16]. The conclusion is that the indicator-based selection using the hypervolume difference indicator [19] performs best – and only this one will be used in the following, denoted here $\text{MO-DAE}_{\text{YAHSP}}$. As explained above, the computation of the fitness is done by YAHSP- and YAHSP , like all known planners to-date, is a single-objective planner. It is nevertheless possible, since PDDL 3.0 [6], to specify other quantities of interest that are to be computed throughout the execution of the final plan, without interfering with the search. Within $\text{MO-DAE}_{\text{YAHSP}}$, two strategies are then possible for YAHSP : it can be asked to optimize either the makespan or the cost, and to simply compute the cost or the makespan when executing the solution plan (for feasible individuals).

The choice between both strategies is governed by user-defined weights, named respectively W -makespan and W -cost (see table 1). For each individual, the actual strategy is randomly chosen according to those weights, and applied to all subproblems of the individual. Note that those weights are tuned using ParamILS (see Section 4), and it turned out that the optimal values for $\text{MO-DAE}_{\text{YAHSP}}$ have always been equal weights: something that was to be expected, as no objective should be preferred to the other.

3.2 Aggregation-Based Multi-objective Divide-and-Evolve

Aggregation is certainly the easiest and most common way to handle multi-objective problems with a single-objective optimization algorithm: a series of single-objective optimization problems are tackled in turn, the fitness of each of these problems is defined by a linear combination of the objectives. In the case of makespan and cost, both to be minimized, each linear combination can be defined by a single parameter α in $[0, 1]$. In the following, F_α will denote $\alpha * \text{makespan} + (1 - \alpha) * \text{cost}$, and $\text{DAE}_{\text{YAHSP}}$ run optimizing F_α will be called the α -run. One “run” of the aggregation method thus amounts to running several α -runs, and returns the set of non-dominated individuals among the union of all final populations⁴. Note that different *alpha*-runs might have different optimal values for their parameters: a complete parameter tuning run of PARAMILS must be performed for each α -run to ensure a fair comparison with other well-tuned approaches.

The choice of the number of values to choose for the different α depends on the available resources. But the choice of the actual values aims at exploring

⁴ Some adaptive method has been proposed [20], where parameter α is adapted on-line, spanning all values within a single run: this is left for further work.

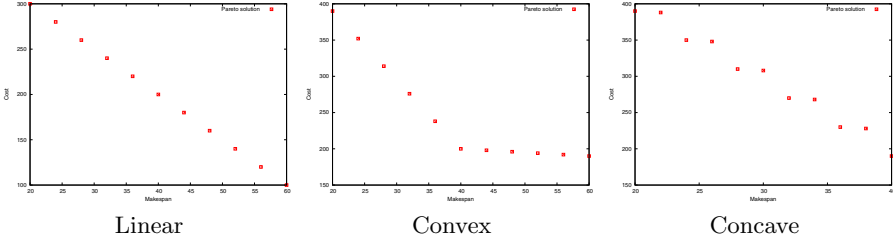


Fig. 2. Pareto Fronts for the MULTIZENO₆_{Cost} problems described in Figure 1

the objective space as uniformly as possible, and some issues might arise if both objectives are not scaled similarly. We hence propose here to use some evenly spaced values for α (see Section 4), but only after both objectives have been scaled into $[0,1]$. However, for such scaling to be possible, some bounds must be known for each objective. When they are not known, these bounds can be approximated from single-objective runs on each of the objectives in turn.

3.3 Multi-objective Benchmarks:

The reader will have by now solved the little puzzle set in Section 2, and found the solution with makespan 8, that manages to leave no plane idle (detailed solution in [16]). In order to turn this problem into a multi-objective one, costs (or risks) are added to the `fly` actions that land in one of the central cities, leading to two types of problem: In MULTIZENO_{Cost}, the second objective is the total costs, that is accumulated every time a plane lands in a central city; In MULTIZENO_{Risk}, the second objective is the maximal risk encountered during the complete execution of a plan; both are to be minimized. The complexity of the instances can be increased by adding more passengers: instances with 3, 6 and 9 passengers will be used here. Finally, by tuning the values of the flight durations and the costs/risks, different shapes of the Pareto front can be obtained: Figure 1 summarizes three possible instances for the MULTIZENO domain, and the corresponding Pareto fronts for the 6-passengers case are displayed in Figure 2.

4 Experimental Settings

Parameter Tuning: It is now widely acknowledged that the large number of parameters of most EAs, even though it is a source of flexibility, is also a weakness, in that a poor parameter setting can ruin the performances of the most promising algorithm. Whereas no generic approach exists for on-line control, there are today many available methods for off-line parameter tuning that should be used within any evolutionary experiment, in spite of their huge computational cost.

In this work, unless otherwise stated, the user-defined parameters of both MO-DAE_{YAHSP} and DAE_{YAHSP} shown in Table 1 have been tuned anew for each instance, using the PARAMILS framework [17]. PARAMILS performs an Iterated

Table 1. Set of parameters off-line tuned using PARAMILS

Parameters	Range	Description
W-makespan	[0,5]	Weight for makespan strategy for YAHSP
W-cost	[0,5]	Weight for cost/risk strategy for YAHSP
Pop-size	[10,300]	Population size
Proba-cross	[0,1]	Probability to apply cross-over
Proba-mut	[0,1]	Probability to apply one of the mutation
w-addatom	[1,10]	Weight for addChangeAtom mutation
w-addgoal	[1,10]	Weight for addGoal mutation
w-delatom	[1,10]	Weight for delAtom mutation
w-delgoal	[1,10]	Weight for delGoal mutation
Proba-change	[0,1]	Probability to change each atom in the addChangeAtom mutation
Proba-delatom	[0,1]	Probability to delete each atom in the delAtom mutation
Radius	[1,10]	Number of neighbour goals to consider for the addGoal mutation

Local Search in the space of possible parameter configurations, evaluating each configuration by running the algorithm to be optimized with this configuration on the given instance.

Stopping Criteria: Due to the variable number of calls to YAHSP the number of function evaluation is not representative of the CPU effort of runs of DAE_{YAHSP} . Hence the stopping criterion of all DAE_{YAHSP} run was set to a given wall-clock time (300, 600 and 1800 seconds for MULTIZENO3, 6 and 9 respectively (on an Intel(R) Xeon(R) @ 2.67GHz or equivalent)). That of $MO-DAE_{YAHSP}$ was set accordingly: for the sake of a fair comparison, because one run of the aggregated approach requires n runs of the single-objective version of DAE_{YAHSP} , $MO-DAE_{YAHSP}$ was run for n times the time of each of the DAE_{YAHSP} runs. In the following, n will vary from 3 to 8 (see Section 5). The stopping criterion for PARAMILS was likewise set to a fixed wall-clock time: 48h (resp. 72h) for MULTIZENO3 and 6 (resp. MULTIZENO9), corresponding to 576, 288, and 144 parameter configuration evaluations for MULTIZENO3, 6 and 9 respectively.

Performance Metrics and Results Visualization: The quality measure used by PARAMILS to optimize the parameters of both $MO-DAE_{YAHSP}$ and each of the α -runs of DAE_{YAHSP} is the unary hypervolume I_{H-} [19] of the set of non-dominated points output by the algorithm with respect to the complete true Pareto front (only instances where the true Pareto front is fully known have been experimented with). The lower the better (a value of 0 indicates that the exact Pareto front has been reached). All reported differences in hypervolume have been tested using Wilcoxon signed rank test at 95% confidence level, unless otherwise stated.

However, and because the true front is made of a few scattered points (at most 17 for MULTIZENO9 in this paper), it is also possible to visually monitor the empirical Cumulative Distribution Function of the probability to discover each point, as well as the whole front. This allows some deeper comparison between algorithms even when none has found the whole front. Such *hitting plots* will be used in the following, together with more classical plots of hypervolume vs time. Finally, because hitting plots only tell if a given point was reached and do not

provide any information regarding how far from the other points the different runs ended, more details on the approximated Pareto fronts will be given by visualizing the merged final populations of all runs of given settings.

Implementation: For all experiments, 11 independent runs have been performed, implemented within the PARADISEO-MOEO framework⁵. All performance assessment procedures (hypervolume calculations, statistical tests), have been achieved using the PISA performance assessment tool⁶.

5 Experimental Results

This section will compare the Pareto-based MO-DAE_{YAHSP} and the aggregation approach AGG-DAE_{YAHSP} on MULTIZENO3, 6 and 9. Unless otherwise stated, the default domain definition leading to a linear Pareto front (see Figure 1 and 2-left) will be used, and one AGG-DAE_{YAHSP} run will be made of 7 different α -runs, with α taking the values 0, 0.1, 0.3, 0.5, 0.7, 0.9, and 1.

The MultiZeno3 Problem proved to be too easy: both MO-DAE_{YAHSP} and AGG-DAE_{YAHSP} find the complete Pareto fronts, and the hitting plots reach 100% in less than 80s (resp. 90s) for the COST (resp. RISK) version of the instance (not shown here). MO-DAE_{YAHSP} is slightly slower (resp. faster) than AGG-DAE_{YAHSP} in the COST (resp. RISK) instance, but no significant difference is to be reported. Only instances -6 and -9 will be looked at in the following.

The Risk Objective: On these instances, however, the RISK objective proved to be almost too difficult to be of interest here, even though there are only 3 points on the Pareto Front, whatever the number of passengers: as can be seen on Figure 6, no algorithm could identify the complete Pareto front for the MULTIZENO9 instance (line 4); for MULTIZENO6 (line 2), MO-DAE_{YAHSP} could reliably identify the whole front (in 9 runs out of 11), while only a single run of AGG-DAE_{YAHSP} could identify the middle point (40,20). MO-DAE_{YAHSP} is hence a clear winner here - however, too little information is brought by the risk value, as one single stop in a risky station will completely hide the possibly low-risk remaining of the plan. Further work will aim at designing a smoother fitness for such situations.

The rest of the paper will hence concentrate on the COST versions of MULTIZENO6 and 9 (simply denoted MULTIZENO{6,9}), where significant differences between both approaches can be highlighted.

Results on the Default Instance: From the plots of the evolution of the average hypervolumes (Figure 3), MO-DAE_{YAHSP} is the winner for MULTIZENO6, and AGG-DAE_{YAHSP} is the winner for MULTIZENO9. Taking a closer look at the hitting plots (Figure 6), we can see for MULTIZENO6 (line 1) that all runs of

⁵ <http://paradiseo.gforge.inria.fr/>

⁶ <http://www.tik.ee.ethz.ch/pisa/>

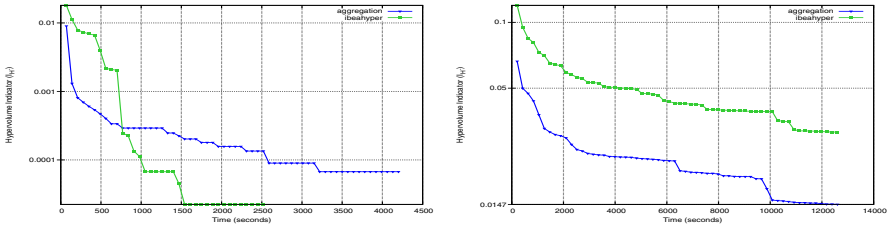


Fig. 3. Evolution of hypervolume for DAE_{YAHSP} (green squares) and $AGG-DAE_{YAHSP}$ (blue triangles) for MULTIZENO6 (left) and MULTIZENO9 (right)

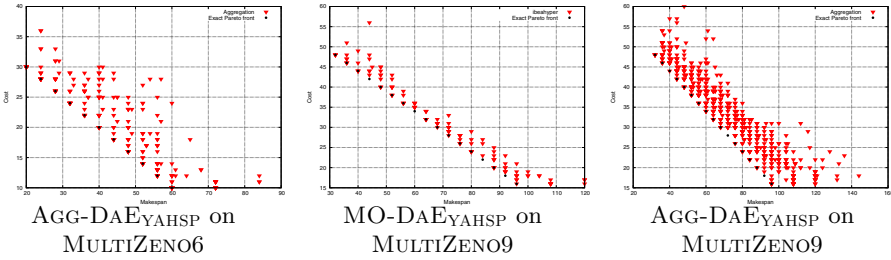


Fig. 4. Pareto fronts approximations (union of all final populations)

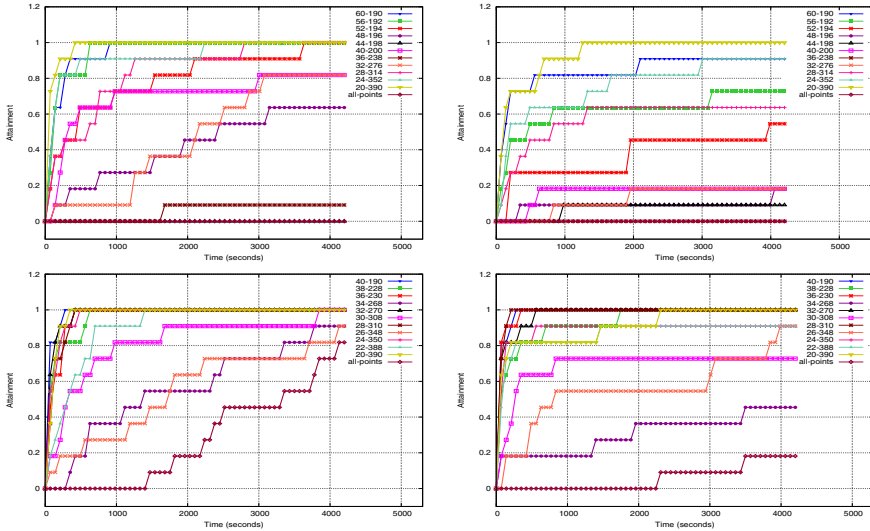


Fig. 5. Hitting plots for $MO-DAE_{YAHSP}$ (left) and $AGG-DAE_{YAHSP}$ (right), for instances 2, 3, and 4 of MULTIZENO6 from Figure 1 (from top to bottom)

$MO-DAE_{YAHSP}$ reach the complete Pareto front in around 2500s, while only 9 runs out of 11 do reach it. On the other hand, for MULTIZENO9, and though the figures of line 3 are more difficult to read because they contain the CDF for 17 points, slightly more points seem to be reached by $AGG-DAE_{YAHSP}$ than

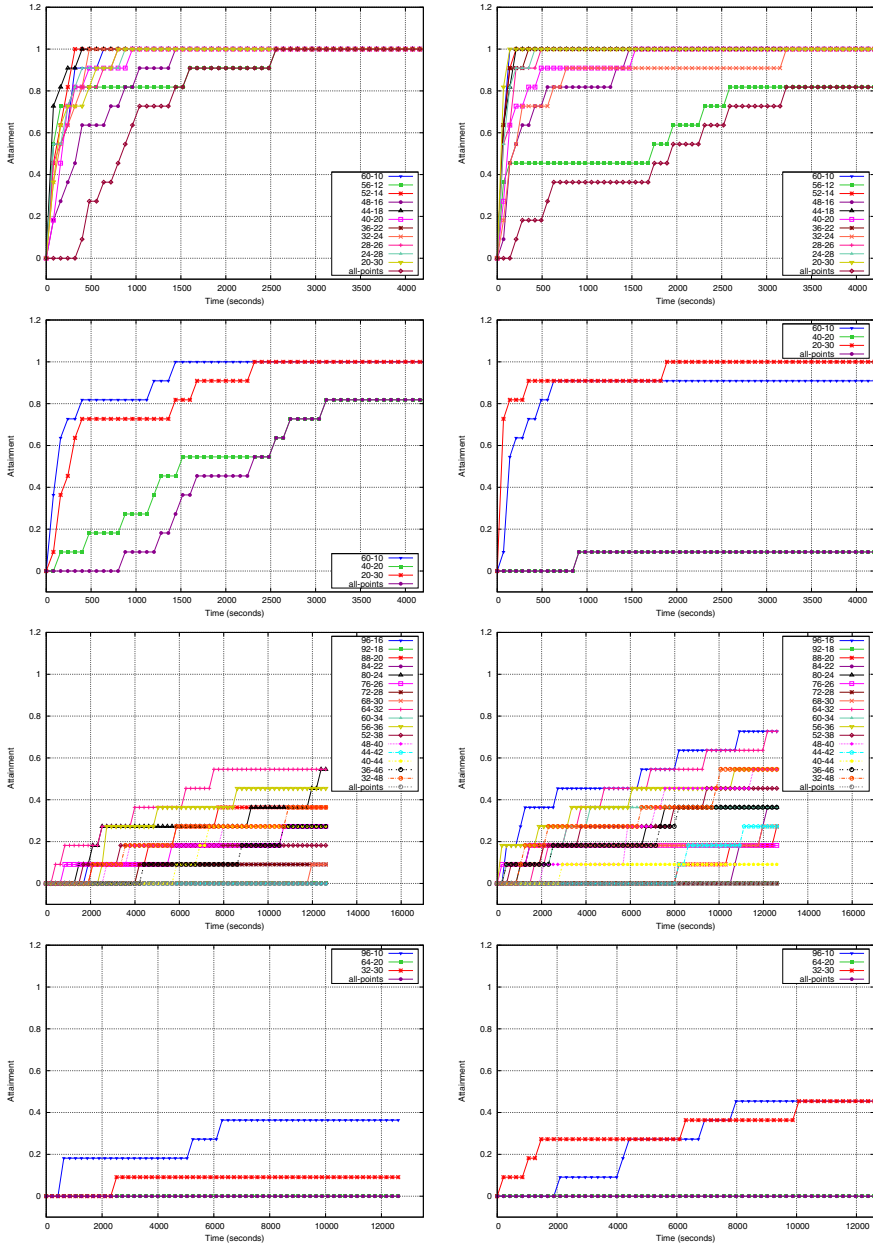


Fig. 6. Hitting plots for MO-DAE_{YAHSP} (left) and AGG-DAE_{YAHSP} (right), for instances (from top to bottom) MULTIZENO6_{Cost}, MULTIZENO6_{Risk}, MULTIZENO9_{Cost}, and MULTIZENO9_{Risk}. The lower line on each plots is the experimental CDF for the probability to reach the whole Pareto front.

by MO-DAE_{YAHSP}. Looking now at the approximations of the Pareto fronts (Figure 4), the fronts returned by AGG-DAE_{YAHSP} for MULTIZENO6 show a large dispersion away from the true front, whereas the same figure for MO-DAE_{YAHSP} (not shown) only contains the true front. Regarding MULTIZENO9, even though it reaches less points from the true front, MO-DAE_{YAHSP} demonstrates a much more robust behavior than AGG-DAE_{YAHSP}, for which the approximate fronts are, again, quite dispersed, sometimes far from the true front.

Results on other MultiZeno6 Instances: Further experiments have been conducted on different variants of MULTIZENO6 instance, described in Figure 1. The corresponding hitting plots can be seen on Figure 5. As in the LINEAR default case, MO-DAE_{YAHSP} is a clear winner – and this is confirmed by the plots of the approximate Pareto fronts (not shown), for which AGG-DAE_{YAHSP} again shows a much larger dispersion away from the true front than MO-DAE_{YAHSP}.

All results presented until now have been obtained by first optimizing the parameters of all algorithms with PARAMILS. Interestingly, when using the parameters optimized by PARAMILS for the Linear instance on these other instances, the results are only slightly worse: this observation will motivate further work dedicated to the generalization of the parameter tuning across instances.

6 Discussion and Conclusion

The experiments presented in this paper have somehow demonstrated the greater efficiency of the Pareto-based approach to multi-objective AI Planning MO-DAE_{YAHSP} compared to the more traditional approach by aggregation of the objectives AGG-DAE_{YAHSP}. The case is clear on MULTIZENO6, and on the different instances that have been experimented with, where MO-DAE_{YAHSP} robustly finds the whole Pareto front (except for the CONVEX instance), whereas AGG-DAE_{YAHSP} performs much worse in all aspects. This is also true on the MULTIZENO9 instance, in spite of the better hypervolume indicator: indeed, a few more points on the Pareto front are found a little more often, but the global picture remains a poor approximation of the Pareto front. Other experiments on more instances are needed to confirm these first results, and on-going work is concerned with solving instances generated from IPC benchmarks by merging the cost and the temporal domains when the same instances exist in both.

Regarding the computational cost, one AGG-DAE_{YAHSP} run requires several single-objective runs – and as many parameter tuning procedures. We have chosen here to use 7 different values for α , and it was clear from results not shown here that taking away a few of these resulted in a decrease of quality of the results. The computational cost of the parameter tuning could be reduced, too: first, a complete tuning anew for each instance is unrealistic, and was only done here for the sake of a fair comparison between both approaches; second, even on a single instance, it should be possible to tune all parameters (except those of YAHSP strategy) for all α -runs together. Finally, one of the most promising directions for future research is the on-line tuning of YAHSP strategy, e.g., using a self-adaptive approach, where the strategies are attached to the individual.

References

1. Ghallab, M., Nau, D., Traverso, P.: *Automated Planning, Theory and Practice*. Morgan Kaufmann (2004)
2. Kambhampati, S.: 1001 ways to skin a planning graph for heuristic fun and profit. Invited talk at ICAPS 2003 (2003)
3. Do, M., Kambhampati, S.: SAPA: A Multi-Objective Metric Temporal Planner. *J. Artif. Intell. Res. (JAIR)* 20, 155–194 (2003)
4. Refanidis, I., Vlahavas, I.: Multiobjective Heuristic State-Space Planning. *Artificial Intelligence* 145(1), 1–32 (2003)
5. Gerevini, A., Saetti, A., Serina, I.: An Approach to Efficient Planning with Numerical Fluents and Multi-Criteria Plan Quality. *Artificial Intelligence* 172(8-9), 899–944 (2008)
6. Gerevini, A., Long, D.: Preferences and Soft Constraints in PDDL3. In: *ICAPS Workshop on Planning with Preferences and Soft Constraints*, pp. 46–53 (2006)
7. Chen, Y., Wah, B., Hsu, C.: Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *J. of Artificial Intelligence Research* 26(1), 323–369 (2006)
8. Edelkamp, S., Kissmann, P.: Optimal Symbolic Planning with Action Costs and Preferences. In: *Proc. 21st IJCAI*, pp. 1690–1695 (2009)
9. Deb, K.: *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley (2001)
10. Brie, A.H., Morignot, P.: Genetic Planning Using Variable Length Chromosomes. In: Biundo, S., Myers, K.L., Rajan, K. (eds.) *15th Intl Conf. on Automated Planning and Scheduling*, pp. 320–329. AAAI Press (2005)
11. Schoenauer, M., Savéant, P., Vidal, V.: Divide-and-Evolve: A New Memetic Scheme for Domain-Independent Temporal Planning. In: Gottlieb, J., Raidl, G.R. (eds.) *EvoCOP 2006*. LNCS, vol. 3906, pp. 247–260. Springer, Heidelberg (2006)
12. Vidal, V., Geffner, H.: Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. In: *Proc. AAAI 2004*, pp. 570–577 (2004)
13. Vidal, V.: A Lookahead Strategy for Heuristic Search Planning. In: *Proceedings of the 14th ICAPS*, pp. 150–159. AAAI Press (2004)
14. Bibai, J., Savéant, P., Schoenauer, M., Vidal, V.: On the Benefit of Sub-optimality within the Divide-and-Evolve Scheme. In: Cowling, P., Merz, P. (eds.) *EvoCOP 2010*. LNCS, vol. 6022, pp. 23–34. Springer, Heidelberg (2010)
15. Bibai, J., Savéant, P., Schoenauer, M., Vidal, V.: An Evolutionary Metaheuristic Based on State Decomposition for Domain-Independent Satisficing Planning. In: Brafman, R., et al. (eds.) *Proc. 20th ICAPS*, pp. 18–25. AAAI Press (2010)
16. Khouadjia, M.R., Schoenauer, M., Vidal, V., Dréo, J., Savéant, P.: Multi-objective AI Planning: Evaluating DAE_{YAHSP} on a Tunable Benchmark. In: Purshouse, R.C., Fleming, P.J., Fonseca, C.M., Greco, S., Shaw, J. (eds.) *EMO 2013*. LNCS, vol. 7811, pp. 36–50. Springer, Heidelberg (2013)
17. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)* 36, 267–306 (2009)
18. Haslum, P., Geffner, H.: Admissible Heuristics for Optimal Planning. In: *Proc. AIPS 2000*, pp. 70–82 (2000)
19. Zitzler, E., Künzli, S.: Indicator-Based Selection in Multiobjective Search. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiño, P., Kabán, A., Schwefel, H.-P. (eds.) *PPSN VIII*. LNCS, vol. 3242, pp. 832–842. Springer, Heidelberg (2004)
20. Jin, Y., Okabe, T., Sendhoff, B.: Adapting Weighted Aggregation for Multiobjective Evolution Strategies. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) *EMO 2001*. LNCS, vol. 1993, pp. 96–110. Springer, Heidelberg (2001)

Predicting Genetic Algorithm Performance on the Vehicle Routing Problem Using Information Theoretic Landscape Measures

Mario Ventresca¹, Beatrice Ombuki-Berman², and Andrew Runka³

¹ Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada

`mario.ventresca@utoronto.ca`

² Department of Computer Science, Brock University, St. Catharines, Canada

`bombuki@brocku.ca`

³ Department of Computer Science, Carleton University, Ottawa, Canada

`arunka@connect.carleton.ca`

Abstract. In this paper we examine the predictability of genetic algorithm (GA) performance using information-theoretic fitness landscape measures. The outcome of a GA is largely based on the choice of search operator, problem representation and tunable parameters (crossover and mutation rates, etc). In particular, given a problem representation the choice of search operator will determine, along with the fitness function, the structure of the landscape that the GA will search upon. Statistical and information theoretic measures have been proposed that aim to quantify properties (ruggedness, smoothness, etc) of this landscape. In this paper we concentrate on the utility of information theoretic measures to predict algorithm output for various instances of the capacitated and time-windowed vehicle routing problem. Using a clustering-based approach we identify similar landscape structures within these problems and propose to compare GA results to these clusters using performance profiles. These results highlight the potential for predicting GA performance, and providing insight self-configurable search operator design.

1 Introduction

We study the well known \mathcal{NP} -hard [7] vehicle routing problem (VRP). Due to its wide applicability the VRP has been widely studied (for detailed reviews, see [3,17,10]). In this paper, we focus on the capacitated vehicle routing problem (CVRP) [22] and vehicle routing problems with time windows (VRPTW) [3]. A typical VRP aims to design least-cost routes from a central depot to a set of geographically dispersed points/customers with various demands. Each customer is to be serviced exactly once by only one vehicle, and each vehicle has a limited capacity. The Vehicle Routing Problem with Time Windows (VRPTW) is an extension of the VRP whereby a *time window* during which service must be completed is associated with each customer. A vehicle may arrive early, but

it must wait until the designated time window to open before service can commence. The objective of the VRPTW is to minimize the number of vehicles used and the total distance travelled to service the customers without violating the capacity and time window constraints.

The main question we shed light on in this paper is whether common information theoretic summary measures of the fitness landscape structure actually provide reliable feedback as to the relative difficulty of solving specific VRP problem instances. That is, whether problem instances can be grouped based on these measures, and whether these groups are indicative of the eventual solution quality one could observe after an evolutionary algorithm terminates. So, we are not aiming to predict the final objective value or behaviour of the algorithm over time, rather, whether there is a correspondence between the measures and performance. We address this question by considering a large subset of benchmark VRP and VRPTW instances and measure the influence that common mutation and crossover search operators have.

A fitness-distance based analysis of problem difficulty for the CVRP was conducted in [9], using a variety of distance measures. Their results indicate the existence of a possible “big valley” structure in the landscape that contains more than half of the sampled problem instances. They argue that this provides a plausible explanation for the success of some well known heuristics on the given problem instances. The analysis considered a much smaller set of problem instances than is provided in this study, as well as considering only the CVRP, whereas we consider VRPTW as well. Similar conclusions were found in [5,6]. The waste-collection vehicle routing problem with time windows was studied in [19]. Many other studies of fitness landscapes exist in the literature for a variety of problems [13,12,1,11,21,25,16,20,18]. To our best knowledge, cluster analysis of information theoretic measures and their relationship to observed GA performance is unique.

2 Fitness Landscapes

A fitness landscape can be defined as a tuple $\mathcal{L} = (\mathcal{S}, f, \mathcal{N})$ where \mathcal{S} is the search space of feasible solutions, $f: \mathcal{S} \mapsto \mathbb{R}$ is a fitness function. The function $\mathcal{N}(s)$ assigns to every s a set of neighbour solutions. Traditionally, neighbours are solutions reachable through a single application of the search operator, but this need not be the case.

Without loss of generality, the following assumes a maximization problem with search space \mathcal{S} where a solution $s \in \mathcal{S}$ is defined to be a *local maximum* if its fitness is greater than or equal to all of its neighbours, i.e., $f(s) \geq f(w) \forall w \in \mathcal{N}(s)$, where the *neighbourhood* $\mathcal{N}(s)$ is defined as the set of solutions reachable from s by a single application of the search operator being considered. If a relatively high number of local optima are present in the landscape, it is termed *rugged*. When few optima exist, the landscape could be either smooth or flat depending on the existence of large attractive basins.

A *basin of attraction* of a solution s_n is defined [8] as the set of vertices $B(s_n) = \{s_0 \in V | \exists s_1, \dots, s_n \in V \text{ where } s_{i+1} \in \mathcal{N}(s_i) \text{ and } f(s_{i+1}) > f(s_i) \forall i, 0 \leq i \leq n\}$.

The size of a basin is generally considered to be defined as the number of solutions within it. Local optima with relatively small attractive basins can be considered *isolated* [8]. Larger basins of attraction typically imply a smoother landscape. Landscapes characterized by few local optima generally contain large amounts of *neutrality* [2]; the fitness of neighbouring solutions remains essentially equal. When existing in neutral epochs, the current set of solutions will randomly drift about these neutral networks.

Problem difficulty may be deduced from analyzing the characteristics defined above. For instance, a landscape having few isolated optima with a high degree of neutrality is likely going to be more difficult to search than a smooth landscape with a single global optima (i.e., a large hill) because on average searching the landscape provides little information indicating the location of peaks. Various measures have been proposed to ascertain properties of the search space, for example [26,14,24,8]. We focus on the information theoretic measures proposed in [24] and [23].

The Information Content (IC) measures the ruggedness with respect to the flat or neutral areas of the landscape. The degree of flatness sensitivity is based on an empirically decided parameter ε which is restricted to the range $[0, \dots, L]$, where L is the maximum fitness difference along the random walk. Consequently, the analysis will be most sensitive when $\varepsilon = 0$. This measure is calculated a

$$H(\varepsilon) = - \sum_{p \neq q} Pr_{[pq]} \log_6 Pr_{[pq]} \tag{1}$$

where probabilities $Pr_{[pq]}$ represent the probabilities of possible fitness transitions from solution p to q while performing a random walk. Each $[pq]$ are elements of the string $S(\varepsilon) = s_1 s_2 s_3 s_n$, of symbols $s_i \in \{\bar{1}, 0, 1\}$, where each s_i is recursively obtained for a particular value of ε based on Equation (2), so $s_i = \Psi_f(i, \varepsilon)$. Thus, ε can be said to represent an accuracy or sensitivity parameter of the analysis.

$$\Psi(i, \varepsilon) = \begin{cases} \bar{1}, & \text{if } f_i - f_{i-1} < -\varepsilon \\ 0, & \text{if } |f_i - f_{i-1}| \leq \varepsilon \\ 1, & \text{if } f_i - f_{i-1} > \varepsilon \end{cases} \tag{2}$$

The Partial Information Content (PIC) indicates the modality or number of local optima present on the landscape. The underlying idea is to filter out repeated symbols of $S(\varepsilon)$ in order to acquire an indication of the modality of the random walk. The formula for computing PIC is given in Equation (3), where n is the length of the original walk and μ is the length of the summarized string $S'(\varepsilon)$.

$$M(\varepsilon) = \frac{\mu}{n} \tag{3}$$

The value for $\mu = \Phi_s(1, 0, 0)$ is determined via the recursive function

$$\Phi_s(i, j, k) = \begin{cases} k, & \text{if } i > n \\ \Phi(i + 1, i, k + 1), & \text{if } j = 0 \text{ and } s_i \neq 0 \\ \Phi(i + 1, i, k + 1), & \text{if } j > 0, s_i \neq 0, s_i \neq s_j \\ \Phi(i + 1, j, k), & \text{otherwise.} \end{cases} \tag{4}$$

When the value of $M(\varepsilon) = 0$ it indicates that no slopes were present on the path of the random walk, meaning that the landscape is rather flat or smooth. Similarly, if $M(\varepsilon) = 1$ then the path is maximally multi-modal and likely very rugged. Furthermore, it is possible to calculate the expected number of optima of a random walk of length n via

$$\mathbb{E}[M(\varepsilon)] = \left\lfloor \frac{nM(\varepsilon)}{2} \right\rfloor, \quad (5)$$

although we do not consider the expected modality in this analysis.

The Density-Basin Information (DBI) measure (Equation 6) indicates the flat and smooth areas of the landscape as well as the density and isolation of peaks. It therefore provides an idea of the landscape structure around the optima.

$$h(\varepsilon) = - \sum_{p \in \{1,0,1\}} Pr_{[pp]} \log_3 Pr_{[pp]} \quad (6)$$

$Pr_{[pp]}$ represents the probability of sub-blocks $\bar{1}\bar{1}$, 00 and 11 of occurring. A high number of peaks within a small area results in a high DBI value. Conversely, if the peak is isolated the measure will yield a low value. Thus, this information gives an idea as to the size and nature of the basins of the landscape. Landscapes with a high DBI content should be easier for an evolutionary algorithm to attract to the area of fitter solutions.

3 Representation and Genetic Operators

We use 66 standard VRPTW and CVRP benchmark instances¹ and consider seven search operators, four mutation and three crossover [4,15]:

- **Swap:** swap two random elements.
- **Inversion:** reverse the order of a contiguous segment of elements (i.e., 2-opt).
- **Insertion:** move an element to a random index.
- **Displacement:** select and move a contiguous segment of elements.
- **PMX:** exchange contiguous segments of elements between parents.
- **UOX:** randomly select subset of elements from each parent, maintaining ordering.
- **CX:** include a random element from parent 1 (P1), then include the element in P1 found at the index of P2 corresponding to the previous included element. Repeat until an element is encountered that already exists in the child, then repeat using an unselected element of P2.

We represent solutions as an array of integers, where each integer appears only once, and corresponds to a stop of the vehicle (i.e., a city). Crossover and mutation operators are applied directly to the representation. Transcribing a solution representation into a valid solution is accomplished by linearly traversing

¹ Available at web.cba.neu.edu/~msolomon/problems.htm and <http://osiris.tuwien.ac.at/wgarn/VehicleRouting/neo/Problem%20Instances/instances.html>.

the representation and adding each stop to a vehicle until its capacity limit is reached. The process repeats using a new vehicle until the representation is fully traversed.

4 Landscape Analysis Results

In this section we present the results of the landscape analysis. The required statistics and probabilities are gathered by taking 2,000 random walks each of length 10,000 steps. We consider the PIC, IC and DBI values as features for a given (instance, operator) pairing and perform a clustering on the scaled values. The optimal clustering model is determined according to the Bayesian Information Criteria (BIC) for expectation maximization initialized by hierarchical clustering for parametrized Gaussian mixture models. The implied landscape of each group can then be analyzed separately. Due to space limitations we provide summary results, but full statistics are available by contacting the authors.

Figure 1 presents the clustering results for the CVRP, presented with respect to the two main principal components. The left figure displays the 8 clusters found for the crossover operators, and the right figure shows the 11 mutation operator-based clusters. Each point represents a (problem instance, operator) pairing and is labelled according to the operator, as indicated in the figure caption. Immediately noticeable is that most clusters are composed of solely a single type of search operator, indicating similarly induced landscape structure for the corresponding problem instances. Additionally, the DBI value is found to explain very little of the total variance.

Table 1 shows the cluster means for each cluster in Figure 1. Class names have been determined by using the operator name (I)nsertion, (D)isplacement, (S)wap, in(V)ersion and sequence of the particular 1, 2 or 3-combination. For instance IV-1, is the first cluster that is represented entirely by insertion and inversion operators, where there are more insertion operators in the class. Cluster ALL contains a mix of all operators.

All 66 UOX results cluster together (UOX-1), indicating that the operator is invariant with respect to the chosen metrics to the problem being considered. The PMX operator is grouped into three clusters. PMX-1 and PMX-2 have very similar values; indeed they are adjacent in the clustering of Figure 1. The practical difference between these two classes is that PMX-1 will have a slightly more rugged landscape. Class PMX-3 has 4 elements, and their corresponding information theoretic measures indicate the landscapes contain a larger variety of shapes, but the overall landscape is slightly flatter with a relatively high degree of peak density. The CX operator has four very distinct landscape structures. CX-1 has a relatively large IC value, implying a landscape that may contain more ruggedness. The DBI measure shows that the density of these peaks is moderately high, compared to UOX-1. The majority (52/66) of PMX results have a landscape that seems less rugged. This leads to the hypothesis of UOX having the most desirable search space, followed by PMX and CX, respectively.

An important aspect is the separation of clusters by problem size, except for UOX-1, which seems problem-invariant. Clusters 1 and 4, represent the PMX

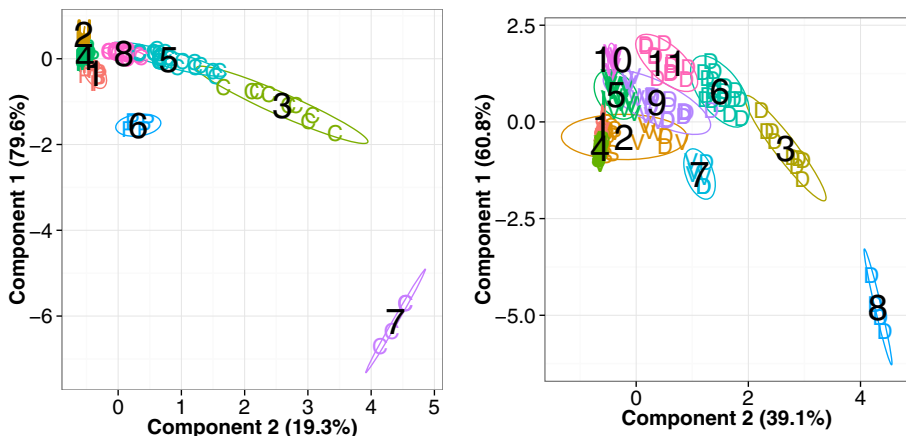


Fig. 1. Optimal clustering of the crossover (left) and mutation (right) problem-operator pairings for CVRP. The plot is shown with respect to the top 2 principal components. Using this approach, 8 distinct crossover and 11 mutation landscapes have been labelled, respectively. Points are placed using a multidimensional scaling with Euclidean distance metric and labelled as (C)X, (U)OX and (P)MX crossover. Mutations are labelled as (S)wap, in(V)ersion, (I)nsertion and (D)isplacement.

operator, where the separation of clusters corresponds nearly perfectly to a difference in problem size; cluster 1 contains C/R10X problem and cluster 4 contains mostly C/R20X problems, respectively. Similarly for clusters 3 and 5, but considering the CX operator. Given this information it can be deduced that the CX and PMX operators have similar landscape structures.

Figure 2 displays the clustering results for the VRPTW. The crossover landscapes form 5 clusters, each having nearly negligible covariance between the principal components. In the right diagram 10 mutation landscape clusters are identified. Both clusterings show very little overlap between elements and sufficient separation that subsequent comparisons could be more straightforward than for the CVRP.

Table 2 shows the cluster means for the VRPTW clustering results shown in Figure 2. For mutation-based clustering, clusters 4 and 9 are composed mostly of displacement operator results. Both of these search spaces contain a larger variety of shapes than the other 8 clusters, while maintaining a high degree of ruggedness as is evident from the PIC and DBI measures. Moreover, the attractive basins seem to be relatively small as well. In contrast, clusters containing the swap and inversion operators have indications of smoother landscapes (low IC and PCI accompanied by high DBI measures). The inversion operator has characteristics that typically result in landscapes that contain a slightly higher degree of ruggedness.

Table 1. The cluster means for CVRP. The first 8 clusters are crossover and the next 11 are mutation-based.

Cluster #	Class	IC	PIC	DBI	# elements
1	PMX-1	0.4196	0.5821	0.5936	10
2	UOX-1	0.4059	0.6273	0.5699	66
3	CX-1	0.6601	0.5465	0.5804	9
4	PMX-2	0.4074	0.6008	0.5851	52
5	CX-2	0.5148	0.5983	0.5716	27
6	PMX-3	0.4754	0.5234	0.6142	4
7	CX-3	0.8054	0.3009	0.6361	4
8	CX-4	0.4571	0.6051	0.5764	26
1	ISV-1	0.3940	0.5327	0.6183	53
2	ALL-1	0.4275	0.5241	0.6186	21
3	D-1	0.6699	0.5163	0.5979	9
4	SIV-1	0.3915	0.5166	0.6247	68
5	IV-1	0.4156	0.5535	0.6076	28
6	D-2	0.5712	0.5556	0.5876	19
7	VD-1	0.5406	0.4985	0.6183	7
8	D-3	0.8091	0.4066	0.6430	4
9	VD-2	0.4765	0.5480	0.6033	27
10	V-1	0.4095	0.5777	0.5969	13
11	D-4	0.4912	0.5753	0.5879	15

Table 2. The cluster means for VRPTW. The first 5 clusters are crossover and the next 10 are mutation-based.

Cluster #	Class	IC	PIC	DBI	size
1	PMX-1	0.4089	0.6338	0.5656	12
2	UOX-1	0.4073	0.6443	0.5590	21
3	CX-1	0.4647	0.6333	0.5581	12
4	PMX-2	0.4068	0.6152	0.5770	9
5	CX-2	0.4631	0.6180	0.5679	9
1	SI-1	0.4295	0.5795	0.5938	15
2	V-1	0.4230	0.6205	0.5721	8
3	I-1	0.4672	0.6038	0.5757	7
4	D-1	0.5058	0.6151	0.5628	12
5	S-1	0.4013	0.5473	0.6117	7
6	VS-1	0.4273	0.5969	0.5849	7
7	S-2	0.4004	0.5309	0.6186	3
8	VS-2	0.4153	0.5668	0.6016	7
9	DI-1	0.4989	0.5888	0.5795	12
10	V-2	0.4079	0.6136	0.5779	6

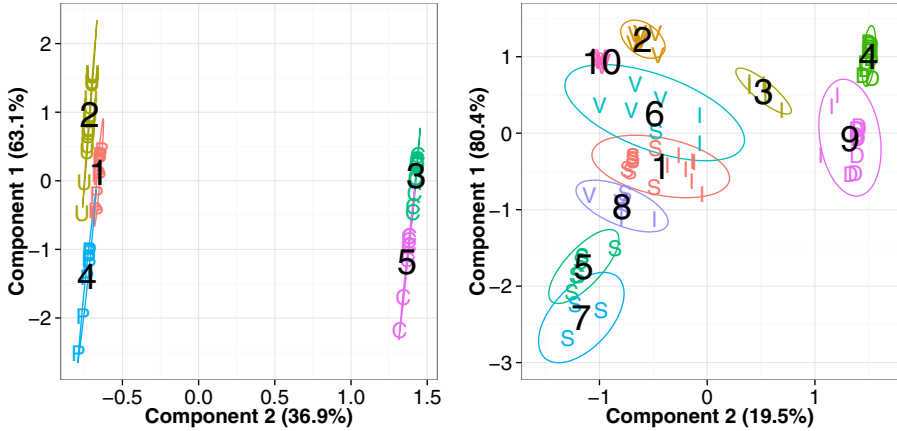


Fig. 2. Optimal clustering of the crossover (left) and mutation (right) problem-operator pairings for VRPTW. Using this approach, 5 distinct crossover and 10 mutation landscapes have been discovered, respectively. Points are placed using a multidimensional scaling with Euclidean distance metric and labelled as (C)X, (U)OX and (P)MX crossover. Mutations are labelled as (S)wap, in(V)ersion, (I)nsertion and (D)isplacement.

5 Genetic Algorithm Results

We employed a genetic algorithm that exclusively uses each of the seven search operators. The GA is run for 5000 generations with a population size of 200. Selection is according to a 3-tournament whereby the best of three randomly selected individuals is carried on to the next population (repeated until 200 individuals are selected). An elitism strategy is also incorporated; we maintain the top two best found solutions at each generation. The GA is run 30 times.

Given space limitations we forgo presenting full statistics about the obtained objective value, etc. Such results are obtainable from the authors. Our goal is to ascertain whether the information theoretic measures are useful indicators of problem difficulty. Since optimal objective values are not scaled to the same range, and in general each problem instance will have different possible evaluations. Instead, we examine how the different search operators compare relative to each other and create performance clusters based on these results. Subsequent comparison of the elements of these performance clusters and landscape clusters is then performed.

CVRP. In all cases the UOX crossover operator showed significantly better fitness across all problem instances when compared to the other crossover operators. Moreover, UOX typically yielded a more desirable outcome than all the

Table 3. Performance profile clusters for mutation operators on CVRP and VRPTW. A < indicates a statistically significant difference between the respective groups and a ~ represents a non-significant result. The Welsh t-test was used at a 0.95 confidence level to ascertain significance. Left to right ordering of operators is according to mean value. The first 20 groups are CVRP and the next 11 are VRPTW.

Group	Relationship	Problems
1	inversion < insertion < swap ~ displace	A-n53-k7, A-n54-k7, A-n55-k9, B-n66-k9, B-n67-k10, c50
2	inversion ~ insertion ~ swap < displace	A-n60-k9, B-n50-k8, B-n57-k9
3	insertion ~ inversion < swap < displace	A-n62-k8, f134
4	inversion < displace < swap ~ insertion	A-n63-k9, E-n76-k10
5	inversion < insertion ~ swap < displace	A-n63-k10, A-n80-k10, B-n63-k10, c75, E-n101-k14, M-n151-k12, P-n55-k10
6	inversion < insertion < swap < displace	A-n64-k9, A-n69-k9, B-n68-k9, B-n78-k10, c100, c100b, E-n76-k7, E-n76-k8, E-n101-k8, M-n101-k10, M-n121-k7, C101, R101, RC101
7	inversion ~ displace < insertion ~ swap	A-n65-k9, B-n51-k7
8	inversion ~ insertion < displace ~ swap	B-n50-k7
9	inversion < insertion < displace ~ swap	B-n52-k7, E-n51-k5, C201, R201
10	inversion ~ insertion < swap ~ displace	B-n56-k7, C101_50, R101_50
11	inversion ~ insertion < swap < displace	c120, c150
12	swap < inversion ~ insertion < displace	c199
13	inversion < insertion < displace < swap	f71, C201_50, R201_50, RC101_50, RC201_50, RC201
14	swap ~ inversion ~ insertion < displace	M-n200-k17
15	inversion < swap ~ insertion < displace	P-n60-k10
16	insertion < inversion < swap ~ displace	tai75a, tai75b, tai150c
17	insertion ~ inversion < swap ~ displace	tai75c
18	insertion ~ inversion < displace ~ swap	tai75d
19	insertion < inversion < swap < displace	tai100a, tai100b, tai100d, tai150b, tai150d
20	insertion < inversion ~ swap < displace	tai150a
1	swap < insertion < inversion < displace	C101, C207
2	swap ~ insertion ~ inversion < displace	C102
3	inversion ~ insertion ~ swap < displace	C103, C203
4	inversion < insertion ~ swap < displace	C104
5	swap ~ insertion < inversion < displace	C105, C107
6	swap < insertion ~ inversion < displace	C106, C206, C208
7	insertion ~ swap ~ inversion < displace	C108, C109, R103
8	swap < insertion < inversion ~ displace	C201, C202, C205, R101
9	inversion < insertion < swap < displace	C204
10	swap ~ insertion < inversion ~ displace	R102
11	insertion < inversion < swap < displace	R104

mutation operators. The PMX operator consistently yielded more desirable results when compared to the CX operator. However, there is no significant trend of more desirable results of PMX when compared to the mutation operators.

Table 3 presents the results of comparing the final mean values for each of the 66 problem instances. The four mutation operators are compared using a Welch t-test at 0.95 confidence level, and the pairwise results of this test are reported, where a $<$ indicates statistical significance and \sim notes the lack thereof, respectively. The relationship ordering was determined according to the mean values attained (not shown). Overlap with problem instances shown in the landscape analysis is large, yielding approximately 70% similarity. Merging clusters with single elements into an existing cluster increases the similarity to 85% similarity. Similar results were also observed for crossover landscapes, but were omitted due to space limitations.

The displacement operator typically occupies the lowest rank (in all but 6 groups), as would be expected considering the landscape analysis. Moreover, Class D-2 in Table 2 indicates a relatively easy search space. Investigating the particular instances for the associated problems A-n63-k9, A-n65-k9, B-n51-k7 and E-n76-k10 further supports the landscape analysis. The results for the displacement operator on these instances is significantly improved from other displacement results (using the relationship ranking as a measure), as the operator is the second rank for Group 4 and 7, respectively.

VRPTW. The results found when running a GA using the four mutation operators are given in Table 3, and grouped according to the statistical dominance relation described above. The swap, inversion and insertion operators tend to occupy the lowest rank (i.e., most desirable outcome), with the swap operator being most frequent. The results from the clustering of landscape measures had indicated that this result should occur.

Another prediction implied by the search space analyses is the deficiency of the displacement operator. For all the 11 rankings in Table 3 displacement is found to be least desirable. Table 3 shows the relatively large degree of displacement operator deficiency as the mean results can be observed to have large effect sizes (in the negative result sense).

We conducted a similar analysis for the three crossover operators (not shown), revealing the relative power of the UOX operator for these problems. Indeed, dominance was observed over all crossover and mutation operators; additionally, a very large effect size was evident. As was also discovered above for the CVRP, the PMX operator consistently, and statistically, dominates the results obtained by CX. In comparison to the fitness landscape clusters we see approximately 87% overlap of problem instances in the clusters.

6 Conclusion

The main question we aimed to address in this study was aimed at whether information theoretic landscape measures can actually be used to discriminate

between problem instance difficulty for VRPTW and CVRP. To this end, numerous benchmark problem instances were considered and seven common search operators were examined. We found that the landscape measures can be clustered into groups that tend to contain mostly one type of search operator. This was true of both CVRP and VRPTW. In order to ascertain whether these clusters can be used to predict outcomes of a genetic algorithm we proposed the use of performance profiles that represent relative ordering of GA results. These profiles are also clustered according to the ordering they represent. We find significant overlap between the landscape and performance clusters. Further study may shed light on automatic search operator design and configuration.

More study of the performance profile approach, and other methods of clustering and comparing landscape and algorithm output may provide deeper insight into predictability of GAs. Future work also includes the examination of different problem representations, which have greatly impact the ability of an algorithm to obtain quality results and whether these results are limited to GAs. In the same vein, consideration of combinations of these, and more advanced, search operators may give some insight into practical implementations.

Acknowledgement. We thank the anonymous reviewers for their valuable insight and comments, and the Natural Sciences and Engineering Research Council of Canada for funding.

References

1. Alander, J.T., Zinchenko, L.A., Sorokin, S.N.: Analysis of fitness landscape properties for evolutionary antenna design. In: IEEE International Conference on Artificial Intelligence Systems, pp. 363–368 (2002)
2. Barnett, L.: Netcrawling-Optimal Evolutionary Search with Neutral Networks. In: Congress on Evolutionary Computation, pp. 30–37 (2001)
3. Braysy, O., Gendreau, M.: Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science* 39, 119–139 (2005)
4. Caramia, M., Onori, R.: Experimenting crossover operators to solve the vehicle routing problem with time windows by genetic algorithms. *International Journal of Operational Research* 3(5), 497–514 (2008)
5. Czech, Z.J.: Statistical measures of a fitness landscape for the vehicle routing problem. In: IEEE International Symposium on Parallel and Distributed Processing, pp. 1–8 (2008)
6. Czech, Z.J.: A parallel simulated annealing algorithm as a tool for fitness landscape exploration. In: Ros, A. (ed.) *Parallel and Distributed Processing*, pp. 247–271. In-Tech (2010)
7. Garey, M.R., Johnson, D.S.: *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company (1979)
8. Jones, T.: *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico (1995)
9. Kubiak, M.: Distance measures and fitness-distance analysis for the capacitated vehicle routing problem. In: Doerner, K.F., Gendreau, M., Greistorfer, P., Gutjahr, W., Hartl, R.F., Reimann, M. (eds.) *Metaheuristics. Operations Research Computer Science Interfaces*, vol. 39, pp. 345–364. Springer (2007)

10. Laporte, G.: Fifty years of vehicle routing. *Transportation Science* 43, 408–416 (2009)
11. Mattfeld, D.C., Bierwirth, C., Kopfer, H.: A search space analysis of the job shop scheduling problem. *Annals of Operations Research* 86, 441–453 (1999)
12. Merz, P., Freisleben, B.: Memetic Algorithms and the Fitness Landscape of the Graph Bi-Partitioning Problem. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) *PPSN 1998*. LNCS, vol. 1498, pp. 765–774. Springer, Heidelberg (1998)
13. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation* 4(4), 337–352 (2000)
14. Naudts, B., Kallel, L.: A Comparison of Predictive Measures of Problem Difficulty in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 4(1), 1–16 (2000)
15. Nazif, H., Lee, L.S.: Optimized crossover genetic algorithm for vehicle routing problem with time windows. *American Journal of Applied Sciences* 7(1), 95–101 (2010)
16. Ombuki-Berman, B., Ventresca, M.: Search difficulty of two-connected ring-based topological network designs. In: *IEEE Symposium on Foundations of Computational Intelligence*, pp. 267–274 (2007)
17. Potvin, J.: State-of-the art review evolutionary algorithms for vehicle routing. *INFORMS Journal on Computing* 21, 518–548 (2009)
18. Reeves, C.: Direct statistical estimation of GA landscape properties. In: *Foundations of Genetic Algorithms* 6, pp. 91–107 (2000)
19. Runka, A., Ombuki-Berman, B., Ventresca, M.: A search space analysis for the waste collection vehicle routing problem with time windows. In: *Genetic and Evolutionary Computation Conference*, pp. 1813–1814 (2009)
20. Schiavinotto, T., Stutzle, T.: A review of metrics on permutations for search landscape analysis. *Computers and Operations Research* 34(10), 3143–3153 (2007)
21. Tavares, J., Pereira, B., Costa, E.: Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 38(3), 604–616 (2008)
22. Toth, P., Vigo, D.: *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications (2002)
23. Vassilev, V.K., Fogarty, T.C., Miller, J.F.: Information Characteristics and the Structure of Landscapes. *Evolutionary Computation* 8(1), 31–60 (2000)
24. Vassilev, V.K., Fogarty, T.C., Miller, J.F.: Fitness Landscapes: from Theory to Application. In: *Advances in Evolutionary Computation: Theory and Applications*, pp. 3–44. Springer (2003)
25. Ventresca, M., Ombuki-Berman, B.: Search space analysis of recurrent spiking and continuous-time neural networks. In: *IEEE International Joint Conference on Neural Networks*, pp. 8947–8954 (2006)
26. Weinberger, E.: Correlated and Uncorrelated Landscapes and How to Tell the Difference. *Biological Cybernetics* 63, 325–336 (1990)

Single Line Train Scheduling with ACO

Marc Reimann¹ and Jose Eugenio Leal²

¹ Institute of Production and Operations Management, University of Graz,
Universitätsstrasse 15, 8010 Graz, Austria

`marc.reimann@uni-graz.at`

² Department of Industrial Engineering, Pontificia Universidade Catolica do Rio de
Janeiro, Rua Marques de Sao Vicente 225, Gavea, Rio de Janeiro, CEP 22453-900,
Brazil

`jel@puc-rio.br`

Abstract. In this paper we study a train scheduling problem on a single line that may be traversed in both directions by trains with different priorities travelling with different speeds. We propose an ACO approach to provide decision support for tackling this problem. Our results show the strong performance of ACO when compared to optimal solutions provided by CPLEX for small instances as well as to other heuristics on larger instances.

1 Introduction and Related Work

In this paper we present a new meta-heuristic approach based on Ant Colony Optimization (ACO) to tackle the train scheduling problem motivated and faced by a Brazilian cargo train company seeking an automated tool to assign track segments to trains over time (see also [13]). We apply our approach to a model commonly studied in literature, the so-called Single Line Train Scheduling Problem (SLTSP). The SLTSP seeks an optimal conflict-free schedule of a set of trains traversing the line in either direction between given origin and destination stations. For each individual train a time schedule of planned arrivals and departures at the visited stations is known and conflicts occur when trains try to occupy the same line segment between two consecutive stations at the same time. Resolving these conflicts - by allowing trains to cross or overtake other trains at the stations - leads to delays of trains with respect to their planned schedules. The objective of the problem is then to find a feasible solution that minimizes the weighted total delay of all trains, where the weights model the different priorities of trains.

Research on mathematical models and solution techniques for the single line train scheduling problem dates back to the early 1970s. In [20] an integer linear programming (ILP) model and a branch & bound (B&B) algorithm were designed to find the best positions for overtaking and crossing of trains. Another B&B algorithm was presented in [11], where the main emphasis was on a strong lower bound to speed up the algorithm. However, since this problem belongs to the class of NP-hard problems (see [17]) it is typically tackled with

heuristics and meta-heuristics. In [3] simple priority rules were applied to a simplified version of the SLTSP and their performance was shown on instances with up to 20 trains. Focusing on a setting where all trains traverse the line in the same direction, a lagrangian heuristic was presented in [4]. In [5] a more general model is considered where trains may again traverse the line in both directions. The model is then tackled by a heuristic where the trains are ordered (according to some appropriate measure) and then a mixed-integer linear programming (MILP) model is solved for each train keeping the relative order of the previously scheduled trains fixed. In [12] several local search, tabu search and genetic algorithms as well as their hybrids are studied for a version of the basic model where double line segments may exist, but are allocated to the two directions of trains a-priori. A heuristic based on a limited depth B&B was proposed in [13] and tested on a real-world instance from a Brazilian cargo train operator. Several papers model train scheduling problems as variants of job-shop scheduling problems (see e.g. [2], [6] and [22]). A rich model and a simple algorithm based on constraint programming were proposed in [14]. The main advantage of the presented approach is the fact that additional constraints can be easily handled within the constraint programming framework. However, the quality of the presented solutions falls far short of the results obtained with the approaches from [12]. In [21] a genetic algorithm is used to schedule new trains into an existing timetable of trains. Most of these approaches focus on special cases of the general SLTSP and are evaluated on very limited sets of typically real-world problem instances.

In this paper we focus on a general model formulation and present an ACO algorithm for its resolution over a large set of structurally different benchmark instances. For small instances we show that the ACO is capable of finding optimal solutions quickly. Moreover, we compare the ACO with its randomized and deterministic counterparts to show the beneficial impact of randomization per se and the use of the pheromone for learning. The remainder of the paper is organized as follows. In the next section the mathematical model underlying our solution approach is shown, while the ACO itself is presented in Section 3. Results from our thorough numerical study are analyzed in Section 4. The paper concludes with a summary of our findings and an outlook on open research questions in Section 5.

2 Mathematical Formulation

The mathematical formulation shown in this section is a variant of the LP presented in [12]. While in that paper the total weighted travel time is minimized, we are interested in minimizing the total weighted tardiness. Let n denote the number of trains and m denote the number of stations. The first $n_1 \leq n$ trains travel outbound, while the remaining trains travel inbound. Note that stations are numbered consecutively and a train is said to travel *outbound* if the number of its origin station is smaller than the number of its terminal station. Consequently, an *inbound* train traverses the line in the opposite direction.

For each train $i = 1, \dots, n$ the origin and destination stations o_i and t_i , respectively and the earliest departure time at the origin station e_i are given. Further, the travel time of an outbound train i between stations $k = 1, \dots, m - 1$ and $k + 1$ is denoted by $\delta_{i,k}$, while the travel time of an inbound train j between stations $k = 2, \dots, m$ and $k - 1$ is denoted by $\delta_{j,k}$. Trains may have scheduled service in a given station k and the associated service time is given by $s_{i,k}$. With these data, we can compute planned arrival times of trains in their destination stations. Clearly, these planned arrival times serve as lower bounds for the actual arrival times, i.e. a train can never be early with respect to these planned arrival times. Below we will see that this means that we can ignore these planned arrival times in our model. Finally, trains may have different priorities w_i .

There are two types of decision variables in the model. The flow of the trains is modeled by real valued variables $d_{i,k}$ which denote the actual departure time of train i in station k . To model the sequence of trains in congested line segments the binary decision variables $x_{i,j,k}$ are introduced, where $x_{i,j,k} = 1$ if train i traverses the line segment after station k before train j and $x_{i,j,k} = 0$ otherwise.

Finally, let M denote a sufficiently large number. Based on these data and variables the model can be written as follows:

$$\min \sum_{i=1}^n w_i d_{i,t_i} \tag{1}$$

$$d_{i,o_i} \geq e_i \quad \forall i = 1, \dots, n \tag{2}$$

$$d_{i,k} \geq d_{i,k-1} + \delta_{i,k-1} + s_{i,k} \quad \forall i = 1, \dots, n_1 \tag{3}$$

and $k = 2, \dots, m$

$$d_{i,k} \geq d_{i,k+1} + \delta_{i,k+1} + s_{i,k} \quad \forall i = n_1 + 1, \dots, n \tag{4}$$

and $k = 1, \dots, m - 1$

$$d_{i,k} \geq d_{j,k} + \delta_{j,k} - Mx_{i,j,k} \quad \forall i, j = 1, \dots, n_1 \tag{5}$$

and $k = 1, \dots, m$

$$d_{j,k} \geq d_{i,k} + \delta_{i,k} - M(1 - x_{i,j,k}) \quad \forall i, j = 1, \dots, n_1 \tag{6}$$

and $k = 1, \dots, m$

$$d_{i,k} \geq d_{j,k} + \delta_{j,k} - Mx_{i,j,k} \quad \forall i, j = n_1 + 1, \dots, n \tag{7}$$

and $k = 1, \dots, m$

$$d_{i,k} \geq d_{j,k} + \delta_{j,k} - Mx_{i,j,k} \quad \forall i, j = n_1 + 1, \dots, n \tag{8}$$

and $k = 1, \dots, m$

$$d_{j,k} \geq d_{i,k} + \delta_{i,k} - M(1 - x_{i,j,k}) \quad \forall i, j = n_1 + 1, \dots, n \tag{9}$$

and $k = 1, \dots, m$

$$d_{i,k} \geq d_{j,k+1} + \delta_{j,k+1} - Mx_{i,j,k} \quad \forall i = 1, \dots, n_1 \tag{10}$$

and $j = n_1 + 1, \dots, n$
and $k = 1, \dots, m$

$$\begin{aligned}
d_{j,k+1} \geq d_{i,k} + \delta_{i,k} - M(1 - x_{i,j,k}) & \quad \forall i = 1, \dots, n_1 \\
& \quad \text{and } j = n_1 + 1, \dots, n \\
& \quad \text{and } k = 1, \dots, m
\end{aligned} \tag{11}$$

Formally, the objective function minimizes the weighted sum of the trains' departure times at their respective destination stations. Since there is no service time at destinations (i.e. $s_{i,t_i} = 0$) these correspond to the trains' actual arrival times at their respective destination stations. Note that this is equivalent to minimizing the total weighted tardiness, as tardiness is just the (positive) difference between actual and planned arrival times and the latter are problem data and thus constant. Constraints (2) ensure that trains can not leave their origin station before their earliest possible departure time. Constraints (3) and (4) constitute the temporal reality of a trains' journey, i.e. a train has to arrive at a station before it can leave this station. Constraints (5)-(11) require that a train has to leave a segment between two stations before another train can enter this segment. Specifically, constraints (5) and (6) consider the case where both trains are outbound. The case of two inbound trains is dealt with by constraints (7) and (9), while the precedence between trains travelling in opposite directions is modelled by constraints (10) and (11).

3 Ant Colony Optimization

Ant Colony Optimization (ACO) was first proposed in [7] as a population-based metaheuristic. Its motivation stems from the underlying metaphor concerning the collective behaviour of real ant colonies leading to the exploitation of rich food sources. More precisely, through a trail laying/trail following mechanism promising (shortest) paths from the nest to a nearby food source are reinforced. An overview of different variants of ACO can be found in [8]. For some basic versions asymptotic convergence results are provided in [9], [10] and [18].

To customize ACO for a particular problem one needs to define a solution construction mechanism describing how to generate feasible solutions, a pheromone model emphasizing how (or what components of) a solution should be memorized and a learning scheme defining how to update the pheromone values. Below we will discuss in detail the implementation of these components in our algorithm.

3.1 Solution Construction

The input of the SLTSP is a planned and typically infeasible timetable specifying the desired departure and arrival times of all trains for their respective journeys. The objective of the SLTSP is to find a feasible and optimal timetable for all trains, such that each train leaves its origin station at or after its planned departure time and each line segment is occupied by only one train at a time. For n trains and m stations any infeasible or feasible solution can be represented as a $n \times m$ matrix of the arrival times of trains in stations. As trains may originate

or terminate at any station along the line not all of the entries in this matrix are necessarily defined.

In our ACO each ant aims at transforming the infeasible planned timetable into a feasible timetable with a minimum total weighted delay of all trains. To this end the solution construction mechanism sequentially removes conflicts between trains. For identifying and removing conflicts we follow most of the existing works (see e.g. [11]) and consider a time scan, where in each step the earliest remaining conflict is considered.

Once the earliest conflict has been identified, its resolution is to hold one of the two involved trains at its entrance station to the congested line segment until the other train has left the segment. Let i and j denote the two trains in conflict and let k_i and k_j be the entrance stations of the two trains to the congested line segment. If both trains travel in the same direction $k_i = k_j$. If trains i and j travel in opposite directions and i is outbound $k_j = k_i + 1$, whereas $k_i = k_j + 1$ if train j is outbound. In a slight abuse of notation we will denote the congested line segment as k below and refer to a conflict by the triple (i, j, k) . Then the decision in ACO is formally based on the following probabilistic rule:

$$\mathcal{P}_i = \begin{cases} \frac{\eta_i \tau_{ijk_i}}{\eta_i \tau_{ijk_i} + \eta_j \tau_{jik_j}} & \text{if conflict } (i, j, k) \text{ is to be resolved} \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

$$\mathcal{P}_j = 1 - \mathcal{P}_i \tag{13}$$

Here \mathcal{P}_i is the probability that the delay between trains i and j in line segment k is resolved by delaying the entry of train i into segment k until train j has left this segment. The pseudo-code of this solution construction mechanism is shown in Figure 1.

```

procedure Construct a feasible solution {
  repeat {
    Find the earliest conflict, let this conflict be  $(i, j, k)$ ;
    Compute the probabilities of delaying train  $i$  or train  $j$  in segment  $k$ 
      by equations (12) and (13), respectively ;
    Draw a random number  $r \in [0, 1]$ ;
    if  $r \leq \mathcal{P}_i$  delay train  $i$ , else delay train  $j$ ;
    Propagate solution and compute the number of remaining conflicts;
  } until no more conflicts exist;
  Return the feasible solution;
}
    
```

Fig. 1. Pseudo-code of the solution construction mechanism

The decision to delay the train i in station k_i depends on the (local) heuristic measures η_i and η_j as well as on the pheromone trails τ_{ijk_i} and τ_{jik_j} which represent the global evaluation of a decision based on the quality of the solutions found

in previous iterations. Thus, the management of the pheromone trails incorporates the learning mechanism into the ACO algorithm. Clearly, higher heuristic values and/or higher pheromone values imply a larger selection probability for trains i and j , respectively.

In our pheromone representation scheme pheromone trails are related to explicit conflicts. More precisely, τ_{ijk_i} shows the learned desirability of delaying train i in station k_i to resolve conflict (i, j, k) leading to a three-dimensional $n \times n \times m$ pheromone matrix. Besides defining the pheromone model we also need to specify the heuristic measure η_i for a train i . This heuristic measure takes into account the train priorities w_i as well as the computation of the estimated delay and is given by

$$\eta_i = \frac{1}{w_i \xi_i^l}, \quad (14)$$

where ξ_i^l is the estimated delay taking into account only the two trains involved in the current conflict. This estimated delay incurred by a train i facing conflict (i, j, k) is given by

$$\xi_i^l = (t_{jk_j} + \delta_{jk_j}) - t_{ik_i} \quad (15)$$

where t_{jk_j} is the departure time of train j in station k_j and δ_{jk_j} is the travel time of train j in the congested segment k it enters after station k_j . Clearly, a smaller delay ξ_i^l will lead to a larger value of the heuristic measure η_i and thus a larger selection probability as mentioned earlier.

Before we turn to the pheromone management let us briefly comment on the inclusion of local search in our algorithm. In some of the related work local search based metaheuristics are used (see e.g. [12]). All the operators are based on reversing the sequence of two or more trains in one or more segments. Essentially this will create new conflicts and the evaluation of such a local search move is possible only after a more or less extensive repair of the solution. Moreover, the logic of the repair is the same as the logic underlying the solution construction, namely to reduce conflicts one by one. In a sense, adding a local search in our algorithm would correspond to a nested implementation of our solution construction mechanism. Since this is computationally expensive we have decided to refrain from using this kind of local search and rather perform more ACO iterations.

3.2 Pheromone Initialisation and Update

In the constructive phase of the ACO algorithm decisions are based on both heuristic information and the pheromone values as described above. At the end of each iteration, that is, once all ants have gone through solution construction, the pheromone update procedure is applied to these pheromone values. The pheromone management used in our algorithm is related to the Hypercube Framework presented in [1] and the MaxMin Ant System (see e.g. [19]) and a

variant of it was first presented in [15]. Formally, the pheromone update rule can be written as

$$\tau_{ijk} := \rho\tau_{ijk} + (1 - \rho)\Delta\tau_{ijk}^* \quad \forall i, j = 1, \dots, n \text{ and } k = 1, \dots, m \quad (16)$$

where $0 \leq \rho \leq 1$ is called the trail persistence and $\Delta\tau_{ijk}^*$ is the amount of reinforcement, which is defined as

$$\Delta\tau_{ijk}^* = \begin{cases} 1 & \text{if train } i \text{ is delayed in conflict } (i, j, k) \text{ in } S^b \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

where S^b is the best solution found up to the current iteration (regardless if it was found in the current iteration or earlier).

The update strategy presented above is a pure elitist strategy, where for edges not belonging to S^b no reinforcement takes place and the pheromone on these edges just evaporates at the rate $(1-\rho)$ towards zero. On the other hand, setting $\Delta\tau_{ijk}^* = 1$ for links that are part of the best found solution implies that the pheromone on these links converges to 1. Together with the fact that at the beginning of the run, the pheromone values are initialised to 1, that is

$$\tau_{ijk} = \tau_0 = 1 \quad \forall i, j = 1, \dots, n \text{ and } k = 1, \dots, m \quad (18)$$

the pheromone values now have a well-defined domain, namely $\tau_{ijk} \in [0, 1]$ and are independent of monotonous transformations of the objective function value.

4 Numerical Analysis

Clearly the focus of our work is on studying the performance of ACO for the SLTSP. For our implementation of ACO we have chosen the following numerical parameter settings: $\rho = 0.975$, $\tau_0 = 1$, the number of ants is 20 and the algorithm was run for a maximum number of 1000 iterations.

Moreover, to get a better understanding of the solution quality obtained with ACO we have implemented two alternative heuristic approaches for comparison, namely a deterministic greedy approach, and a randomized heuristic. Let us briefly describe these heuristics.

Like our ACO, both of these heuristics are based on a sequential resolution of conflicts according to their time of occurrence, i.e. earlier conflicts are resolved first. They also utilize the same measure for resolving a conflict. However, the main difference is in the decision making. The deterministic, greedy approach (referred to as GREEDY hereafter) resolves a conflict according to the following rule:

$$\mathcal{P}_i = \begin{cases} 1 & \text{if conflict } (i, j, k) \text{ is to be resolved and } \eta_i \geq \eta_j \\ 0 & \text{otherwise.} \end{cases} \quad (19)$$

$$\mathcal{P}_j = 1 - \mathcal{P}_i \quad (20)$$

Thus, this algorithm ignores the pheromone and deterministically delays the train with the smaller weighted delay. Due to its deterministic nature, this algorithm produces one unique solution in negligible time.

The second heuristic (referred to as RANDOM hereafter) is based on a randomized decision rule similar to ACO. Specifically this decision rule is given by

$$\mathcal{P}_i = \begin{cases} \frac{\eta_i}{\eta_i + \eta_j} & \text{if conflict } (i, j, k) \text{ is to be resolved} \\ 0 & \text{otherwise.} \end{cases} \quad (21)$$

$$\mathcal{P}_j = 1 - \mathcal{P}_i \quad (22)$$

Compared to ACO we observe that this rule does not utilize the pheromone. Here the term randomized refers to the fact that decisions are not based on a deterministic rule, but rather on a probability distribution underlying the alternatives. While there is some bias in the evaluation of the options this bias is exogeneous to the search and constant over time such that the randomized algorithm will not exhibit any learning or convergence. This is in contrast to the ACO idea, where the learning in form of the pheromones will influence the bias and lead to some kind of convergence. To enable a fair comparison with ACO we will compute the same number of solutions, namely 20000. Further we run the ACO and the RANDOM algorithm 10 times on each instance. All the heuristics were implemented in C and run on an Intel Core2 Duo SP9400 with 2.4 GHz.

Finally, we implemented the model presented in Section 2 in CPLEX 9.1. and ran it on the *small* instances for 3600 seconds with the standard setting on an Intel Core2 Duo Processor with 2.8 GHz.

There is no set of benchmark instances available to test a new algorithm. Thus, to test and compare our approaches we generated a set of 128 instances which differ with respect to several important characteristics of the SLTSP, including number of trains and stations, train priorities, direction of trains, origin-destination characteristics of trains and speed of trains. For a thorough description and analysis of the instances we refer to our working paper [16]. For the purpose of this paper we will focus on the number and direction of trains. Concerning the number of trains we will distinguish between 64 small instances with 10 trains and 64 large instances with 20 trains. Concerning the direction of trains we distinguish between 64 instances with uni-directional traffic and 64 instances where traffic is allowed in both directions. Thus, we obtain 4 subsets with 32 instances each.

Let us first look at the small instances. Table 1 shows for all our tested algorithms the number of proven optima found and the average Relative Percentage Deviations - RPDs - of the best solutions from the best lower bounds returned by

CPLEX after 3600 seconds. Further, for the RANDOM and ACO approaches we also show the RPDs of the average results over the ten runs (termed RPD_{avg}). We do not list the computation times for the heuristics as they are all below 1 second for each run on each instance. However, the average time taken by CPLEX to find its best upper bound, i.e. its best feasible solution is around 265 seconds.

Table 1. Number of proven optima found as well as averages of the best and average RPDs obtained by the different approaches for the different problem classes

Problem class	CPLEX		GREEDY		RANDOM			ACO		
	# of optima	RPD	# of optima	RPD	# of optima	RPD	RPD_{avg}	# of optima	RPD	RPD_{avg}
one-way	28	5.51	13	30.01	28	5.2	5.48	28	5.2	5.75
two-way	23	17.85	1	52.11	22	17.97	19.44	23	17.71	18.42
total	51	11.68	14	41.06	50	11.59	12.46	51	11.45	12.08

From Table 1 we observe that CPLEX is able to find solutions with proven optimality for 51 out of the 64 *small* instances. Further, we see that ACO also finds all these optimal solutions, while the other two algorithms fail to do so. Concerning the distinction between *one-way* and *two-way* instances we observe that all algorithms perform at least slightly worse for the latter instances in terms of the number of optima found and clearly worse in terms of the RPD. However, in terms of the RPD we do not know whether the lower bound or the upper bound (or both) induce this effect.

Overall we find that ACO and also RANDOM have a smaller RPD than CPLEX, i.e. these algorithms provide better upper bounds than CPLEX on at least some of the instances for which optimality is not proven. By looking at the detailed results we indeed found that ACO outperforms CPLEX on *one-way* as well as *two-way* instances, while RANDOM outperforms CPLEX only on the former instances. These results provide a first indication of the strength of ACO.

While the best results provide some insights on the quality of an algorithm, for the randomized algorithms RANDOM and ACO the consideration of average results is more fair when showing comparisons with deterministic results as provided by CPLEX (and the GREEDY heuristics). By looking at the corresponding RPD_{avg} columns we observe that the results obtained by the ACO are quite robust while the variance of the RANDOM approach seems to be larger. To verify this, we performed statistical testing. Specifically we ran one-sided Wilcoxon Signed-Rank Tests to compare ACO and RANDOM with respect to the RPDs of the best (RPD) and average (RPD_{avg}) results on the 95% significance level. While there was no statistically significant difference in the best results, for the average results the test returned $Z = 1.7$ and a p-value of 0.0446, indicating that ACO outperforms RANDOM on average.

To get a better understanding of the relative performance of the ACO and RANDOM variants let us now look at the *large* instances. Given the increased problem size in combination with the much larger number of conflicts found,

Table 2. Overall performance of the different algorithms on *large* instances

Problem class	# of best solutions	RANDOM			time sec.	# of best solutions	ACO			time sec.
		best	avg.	worst			best	avg.	worst	
one-way	12	5.28	8.46	11.24	22.28	28	0.19	1.08	2.75	12.11
two-way	2	12.52	16.45	19.61	32.38	20	0.33	3.67	8.19	35.98
total	14	8.9	12.45	15.42	27.33	48	0.26	2.38	5.47	24.04

these instances can be expected to be more difficult and the observed effects should be more pronounced.

Table 2 shows the overall results of the ACO and RANDOM variants. As we do not have complete CPLEX results for these instances¹ we report for each algorithm the number of best solutions found, the RPDs (measured w.r.t. the best known solution of an instance) of the best, average and worst results over ten runs as well as the average computation times. Due to its already poor performance on the *small* instances we did not run the GREEDY algorithm on the *large* instances.

Table 2 clearly shows the superiority of ACO over the RANDOM heuristic. First, the number of best solutions found is much larger in case of ACO. Second, even the worst RPDs of the ACO is better than the best RPDs of the RANDOM algorithm. Moreover, this is true both overall as well as for the one-way and two-way instances separately. Concerning the comparison between these two groups we again observe the effect that two-way instances seem to be more difficult than one-way instances for both algorithms. However, again the ACO is more robust with respect to these traffic conditions. Finally, the overall computation times of ACO and RANDOM show no systematic difference.

Summarizing, these results show the superiority of the ACO compared with the RANDOM approach and indicate that the pheromone-based learning helps in finding near-optimal solutions quickly.

5 Conclusions and Future Research

In this paper we have presented a new ACO metaheuristic for the Single Line Train Scheduling Problem and found that it not only matches the optimal solutions found by CPLEX for small instances, but it also outperforms CPLEX for those instances where optimal solutions are not known. Further it reaches its solutions in much smaller computation times. For the larger instances we find that the worst solutions returned by ACO are on average better than the best solutions found by a randomized heuristic, showing the importance of the pheromone-based learning.

¹ We ran CPLEX on a selected number of the instances and even after 7200 seconds the gaps were partly above 100% and the best upper bounds were far worse than the results obtained with our heuristics. On a couple of instances CPLEX failed to find a feasible solution at all.

Within the solution construction of ACO an interesting issue for future research is the investigation of different strategies to resolve conflicts, particularly *joint* conflicts of more than two trains. In the current version, in each step one conflict between two trains is resolved ignoring the possibility of additional conflicts between a third (fourth,...) train and one or both of the considered trains in the same track segment. Using a more sophisticated approach for identifying and resolving such group conflicts can be expected to be more expensive in terms of computation time, but may improve the solution quality such that an analysis of this tradeoff should be done.

Acknowledgments. We thank three anonymous referees for their valuable comments on an earlier version of this paper. The second author thanks the National Council of Research-CNPq (Brazil) for the Grant which supports this research.

References

1. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics-Part B* 34, 1161–1172 (2004)
2. Burdett, R.L., Kozan, E.: A disjunctive graph model and framework for constructing new train schedules. *European Journal of Operational Research* (to appear)
3. Cai, X., Goh, C.J.: A fast heuristic for the train scheduling problem. *Computers and Operations Research* 21, 499–510 (1994)
4. Caprara, A., Monaci, M., Toth, P., Guida, P.L.: A lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Applied Mathematics* 154, 738–753 (2006)
5. Carey, M., Lockwood, D.: A model, algorithms and strategy for train pathing. *Journal of the Operational Research Society* 46, 988–1005 (1995)
6. D’Ariano, A., Pacciarelli, D., Pranzo, M.: A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research* 183, 643–657 (2007)
7. Dorigo, M., Maniezzo, V., Coloni, A.: The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B* 26, 29–41 (1996)
8. Dorigo, M., Stützle, T.: *Ant Colony Optimization*. MIT Press/Bradford Books, Cambridge, MA (2004)
9. Gutjahr, W.J.: A graph-based ant system and its convergence. *Future Generation Computing Systems* 16, 873–888 (2000)
10. Gutjahr, W.J.: ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters* 82, 145–153 (2002)
11. Higgins, A., Kozan, E., Ferreira, L.: Optimal scheduling of trains on a single line track. *Transportation Research B* 30, 147–161 (1996)
12. Higgins, A., Kozan, E., Ferreira, L.: Heuristic techniques for single line train scheduling. *Journal of Heuristics* 3, 43–62 (1997)
13. Leal, J.E.: A heuristic approach to the problem of scheduling trains on single lines. Working paper, Department of Industrial Engineering, PUC-Rio (2008)

14. Oliveira, E., Smith, B.M.: A Combined Constraint-Based Search Method for Single-Track Railway Scheduling Problem. In: Brazdil, P., Jorge, A. (eds.) EPIA 2001. LNCS (LNAI), vol. 2258, pp. 371–378. Springer, Heidelberg (2001)
15. Reimann, M.: Combining an Exact Algorithm with an Ant System for Travelling Salesman Problems. Working paper, University of Vienna (2003)
16. Reimann, M., Leal, J.E.: Single line train scheduling with Ant Colony Optimization. Working paper, University of Graz (2011)
17. Sahin, I.: Railway traffic control and train scheduling based on inter-train conflict management. *Transportation Research B* 33, 511–534 (1998)
18. Stützle, T., Dorigo, M.: A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation* 6, 358–365 (2002)
19. Stützle, T., Hoos, H.: The Max-Min Ant System and Local Search for Combinatorial Optimization Problems. In: Voß, S., Martello, S., Osman, I.H., Roucairol, C. (eds.) *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 313–329. Kluwer, Boston (1999)
20. Szpigel, B.: Optimal train scheduling on a single track railway. In: Ross, M. (ed.) *Proceedings of OR 1972*, pp. 343–352. North-Holland, Amsterdam (1973)
21. Tormos, P., Lova, A., Barber, F., Ingolotti, L., Abril, M., Salido, M.A.: A Genetic Algorithm for Railway Scheduling Problems. In: Xhafa, F., Abraham, A. (eds.) *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*. SCI, vol. 128, pp. 255–276. Springer, Heidelberg (2008)
22. Zhou, X., Zhong, M.: Bicriteria train scheduling for high-speed passenger railroad planning applications. *European Journal of Operational Research* 167, 752–771 (2005)

Solving Clique Covering in Very Large Sparse Random Graphs by a Technique Based on k -Fixed Coloring Tabu Search

David Chalupa

Slovak University of Technology, Ilkovičova 3, 842 16 Bratislava, Slovakia
chalupa@fiit.stuba.sk

Abstract. We propose a technique for solving the k -fixed variant of the clique covering problem (k -CCP), where the aim is to determine, whether a graph can be divided into at most k non-overlapping cliques. The approach is based on labeling of the vertices with k available labels and minimizing the number of non-adjacent pairs of vertices with the same label. This is an inverse strategy to k -fixed graph coloring, similar to a tabu search algorithm TabuCol. Thus, we call our method TabuCol-CCP. The technique allowed us to improve the best known results of specialized heuristics for CCP on very large sparse random graphs. Experiments also show a promise in scalability, since a large dense graph does not have to be stored. In addition, we show that Γ function, which is used to evaluate a solution from the neighborhood in graph coloring in $\mathcal{O}(1)$ time, can be used without modification to do the same in k -CCP. For sparse graphs, direct use of Γ allows a significant decrease in space complexity of TabuCol-CCP to $\mathcal{O}(|E|)$, with recalculation of fitness possible with small overhead in $\mathcal{O}(\log \deg(v))$ time, where $\deg(v)$ is the degree of the vertex, which is relabeled.

Keywords: clique covering, tabu search, k -fixed strategy.

1 Introduction

Let $G = [V, E]$ be an undirected graph and let $d(G) = \frac{2|E|}{|V|(|V|-1)}$ be its density. The aim of the (*vertex*) *clique covering problem* (CCP) is to find minimum $k \leq |V|$, such that there are pairwise disjoint classes $V_1, V_2, \dots, V_k \subset V$, which cover the whole vertex set, i.e. $V_1 \cup V_2 \cup \dots \cup V_k = V$, and induce cliques, i.e. $\forall i = 1..k \ d(G(V_i)) = 1$. This minimum k is referred to as the clique covering number and is denoted by ϑ . CCP is known to be NP-hard, with its k -fixed variant being NP-complete [12]. This k -fixed variant of CCP (k -CCP) is formulated as the following decision problem: given a fixed k , is there a clique covering with at most k cliques?

In this paper, we tackle k -CCP by performing *tabu search in the space of vertex labelings* with k available labels. In the representation we use, vertices with the same labels induce subgraphs, which represent cliques in the optimal solution. Therefore, we use local search to minimize a penalty function, which

counts the number of non-adjacent vertices with the same labels. This is a very similar approach to the TabuCol algorithm for graph coloring [9]. Therefore, we call our method *TabuCol-CCP*.

Experimental verification is presented for a set of uniform random graphs, which are very large but sparse, with up to 3×10^4 vertices. The approach is shown to be promising for its ability to improve the results of the currently best specialized heuristics for CCP on this type of graphs. Additionally, since CCP and graph coloring are in a complementary relationship, we present a brief comparison of scalability for TabuCol-CCP and TabuCol to argue that this is a better idea than solving graph coloring for the complement. We show that there is a data structure that allows a decrease in space complexity to $\mathcal{O}(|E|)$, instead of $\mathcal{O}(|V|^2)$, with recalculation possible in $\mathcal{O}(\log \deg(v))$ time, where $\deg(v)$ is the degree of the vertex, which is relabeled. This data structure seems to be crucial for the possibility to further scale this technique for even larger graphs.

The paper is structured as follows. In Section 2, we review the related work and provide the primary motivation for this paper. In Section 3, we describe our technique and its components. In Section 4, we discuss the scalability issue. In Section 5, we present the experimental evaluation of the approach. Finally, in Section 6, we give conclusions of this work.

2 Related Work

Although CCP is a classical problem [12], there is only a limited number of results on it. Theoretical foundations seem to be more developed for the edge clique covering problem, including results in data reductions [8], algorithms for complex networks [1] or special classes of graphs, such as subtree filament graphs [13]. Regarding the vertex CCP, which we study in this paper, there is a relatively recently proposed order-based representation of this problem, where a permutation of graph's vertices is mapped to a clique covering with non-fixed number of cliques. This leads to a minimization problem for the number of cliques [4].

In addition to these theoretical issues, each clique covering with k cliques represents a graph coloring with k colors for the complementary graph. Therefore, graph coloring heuristics are useable for both CCP and k -CCP. However, these algorithms are generally tailored for relatively small instances. For example, the well-known DIMACS benchmark for graph coloring mostly consists of graphs with up to 10^3 vertices [11]. It seems that the largest graph, which is currently studied, *C4000.5*, has 4×10^3 vertices¹. To the best of our knowledge, current research overlooks the scalability issue of these heuristics, which seems to be even more pronounced in k -CCP than in CCP.

In CCP and k -CCP, we may encounter very large graphs (such as samples of real-world complex networks, by which this research has been originally motivated). Furthermore, these graphs often tend to be sparse. Thus, they might lead to very dense complements. This leads to situations, where high computational

¹ According to a relatively current graph coloring library:
<http://www.info.univ-angers.fr/pub/porumbel/graphs/>

demands, which are seemingly the hardest issue already for smaller graphs, are overshadowed by extreme memory demands in larger graphs. In this paper, we aim to adapt the ideas used in graph coloring to solve CCP. A similar issue was addressed in the research on the adaptation of tabu search heuristic STABULUS for maximum independent set to the maximum clique problem [7].

From the *k-fixed algorithms*, the most popular algorithmic strategies in graph coloring include simulated annealing [10] and quantum annealing [16], tabu search [9] and memetic algorithms, which combine local search with problem-specific crossovers [14,15]. From the *non-k-fixed algorithms*, greedy coloring [17] and Brélaz’s saturation-based heuristic [3] can be used to efficiently estimate a suboptimal solution for CCP. Order-based representation for CCP provides solid estimation results without tackling the problem “inversely” through graph coloring [4]. However, to find clique coverings with lower k , one might have to fix k to a constant, since the search space of CCP tends to be often very flat, causing the algorithms such as iterated greedy [4,5] to search on very large plateaus if k is non-fixed [6]. Therefore, in this paper, we come up with a k -fixed technique for solving k -CCP, based on coloring tabu search.

3 The k -Fixed Heuristic Technique for CCP

In this section, we propose the k -fixed technique. Firstly, we formulate the problem in a suitable way. Secondly, we discuss the neighborhood structure and the mutation operator. Finally, we provide a detailed specification of the technique and its components.

3.1 Formulation of the Problem and the Search Space

Let $C = \{1, 2, \dots, k\}$ be the set of k available labels. Then, we have a search space $\Omega = C^V = \{V \rightarrow C\}$ of all possible labelings of vertices in V with labels from C . The solution represents a partitioning of the graph, where each subgraph is induced by the vertices of the same label, similarly as in graph coloring [6].

Regarding the *quality function*, we will consider a penalty function, which counts the number of conflicts. In the coloring problem, a conflict occurs when two vertices, which are adjacent, are labeled with the same color. The objective is then to minimize the number of conflicts to zero. In k -CCP, we minimize the number of vertex pairs, which have the same label but are not adjacent. Suppose that the evaluated solution is $S = \{V_1, V_2, \dots, V_k\}$. Then, the objective is to solve the following minimization problem:

$$\min J(S) = \sum_{V_i \in S} \sum_{v, w \in V_i, v \neq w} (1 - A(v, w)), \quad (1)$$

where v and w are different vertices in class V_i and $A(v, w)$ is an adjacency function, which is 1, when vertices v and w are adjacent. Otherwise, it is 0.

3.2 The Neighborhood and the Mutation Operator

The *neighborhood* in our approach will be defined as a set of all solutions, which are obtained by relabeling a conflicting vertex with each of the $k - 1$ remaining labels. Thus, there are $(k - 1) \times f(S)$ solutions in the neighborhood, where $f(S)$ is the number of vertices, which are suitable for relabeling. Suppose that the current solution is $S = \{V_1, V_2, \dots, V_k\}$ and vertex v is in class V_c . We will say that v is conflicting with w , if both v and w are in the same class V_c but $\{v, w\} \notin E$. If the number of vertices within the class, which are conflicting with v , is positive, then it makes sense to relabel v , thus, possibly resolving a conflict:

$$\sum_{w \in V_c, v \neq w} 1 - A(v, w) > 0. \tag{2}$$

To provide an efficient implementation for the previous ideas, we will work with precomputed values. Let us have a function $\Gamma : V \times C \rightarrow \mathbb{N}$ (we assume that $0 \in \mathbb{N}$), for which $\Gamma(v, c)$ denotes, how many neighbors of v are labeled with c [2]. The number of non-neighbors of v , which are labeled with c , is:

$$\Gamma^*(v, c) = \begin{cases} |V_c| - \Gamma(v, c) - 1 & \text{if } v \in V_c \\ |V_c| - \Gamma(v, c) & \text{if } v \notin V_c \end{cases} \tag{3}$$

Suppose that we have an initial solution S . To compute the values of the Γ function for S , we have to iterate only over the edges of the graph, which can be done in $\mathcal{O}(|E|)$ time. For each vertex, we also have to determine, whether it is conflicting, which, however, takes only $\mathcal{O}(|V|)$ time. To evaluate the neighborhood of S , we consider all conflicting vertices. Let $J(S)$ be the value of the objective function for S . Then, by relabeling a vertex, say v , from label c to label d , we would obtain a new solution S' , for which the conflicts within V_c will be resolved and conflicts within V_d will be introduced for v . Therefore, $J(S') = J(S) - \Gamma^*(v, c) + \Gamma^*(v, d)$, which is a strategy used in graph coloring [2]. By using formula (3), we obtain that:

$$J(S') = J(S) - |V_c| + \Gamma(v, c) + 1 + |V_d| - \Gamma(v, d). \tag{4}$$

When we use this strategy, the complexity of recomputing of the objective function is $\mathcal{O}(1)$. When a move is performed, values in Γ also have to be updated. This takes $\mathcal{O}(|V|)$ steps, but is done less frequently than the fitness recalculation. In addition, we emphasize the fact that directly the Γ function is used to perform the recalculation. We will later show that this allows a sparse data structure to be designed to implement Γ , instead of a large matrix, which would be intractable to store for extremely large graphs.

3.3 The Local Search Algorithm

In this section, we explain, how we use the previous ideas in the k -fixed tabu search technique. This method is called *The TabuCol-CCP* and its basic pseudocode is given in Algorithm 1.

Algorithm 1. The TabuCol-CCP technique

The TabuCol-CCP technique	
Input: graph $G = [V, E]$, the number of cliques k	
Output: output state S	
1	use Brélaz's heuristic to generate initial $S = \{V_1, V_2, \dots, V_k\}$
2	compute matrix Γ , objective value $J(S)$ and the list of conflicting vertices V_{CONFL} for S by Algorithm 2
3	set $S_{BEST} = S$, $J(S_{BEST}) = J(S)$ and $T = \emptyset$, $t_{tabu} = 0$
4	while <i>stopping_criterion</i> is not met
5	find a move $[v, d]$ to be performed on S by Algorithm 3
6	let c be the label such that $v \in V_c$, set $V_c = V_c \setminus \{v\}$, $V_d = V_d \cup \{v\}$ and let the new $S = \{V_1, V_2, \dots, V_k\}$, Γ and V_{CONFL} reflect the change
7	$T = T \cup [v, c]$, set the tabu tenure t_{tabu} for $[v, c]$
8	set T_{old} which contains moves with expired tabu tenure and let $T = T - T_{old}$
9	every ϕ iterations, take the best objective value and worst objective value and modify t_{tabu}
10	if $J(S) < J(S_{BEST})$ then $S_{BEST} = S$, $J(S_{BEST}) = J(S)$
11	return S_{BEST}

Let us first explain the steps of Algorithm 1 on a conceptual level. In the step 1, we generate an initial solution S with an adaptation of Brélaz's graph coloring heuristic, which will be discussed more precisely later. In the step 2, we calculate the Γ function for S , along with the list of conflicting vertices V_{CONFL} and objective value $J(S)$. This is done using a strategy expressed in Algorithm 2. In the step 3, the currently best solution S_{BEST} is set to S , the tabu list T is reset to an empty set and the tabu tenure t_{tabu} , i.e. the number of iterations, for which a move is set to tabu, is initialized to 0. Next, an iterative procedure is performed. In the step 5, we use Algorithm 3 to examine the neighborhood of current solution S and choose a move $[v, d]$, which should be performed. In the step 6, we relabel v with d , thus moving v from class V_c to V_d . In the step 7, we set the move $[v, c]$ tabu for several iterations. In the step 8, we decrease the tabu tenures of current tabu moves and exclude the expired ones. The step 9 is performed only every ϕ iterations, where we modify the tabu tenure t_{tabu} , according to the fluctuation of the objective function during the observation period ϕ . The details of this step will be discussed later. Finally, in the step 10, we check whether the new solution is better than anything else found so far. We note that we stop, when the current objective value $J(S)$ is 0 or the algorithm exceeds some time threshold. At this point, we move to a more detailed explanation of the more complex steps of Algorithm 1.

Generating the Initial State by Brélaz's Heuristic. Brélaz's heuristic is a classical graph coloring algorithm, which we use in a slightly modified fashion. The reason why we use it is that for very large graphs, this strategy tends to generate less conflicts than random assignment of labels. For more details on the original method, the reader may refer to [3]. In Brélaz's heuristic, vertices

are ordered in a single greedy procedure and colored in a way that the first color, which does not induce a conflict, is taken. In graph coloring, this would mean that we choose the first color, with which no neighbor of the vertex is colored. In CCP, we use the first label, with which no non-neighbor of the vertex is labeled. The ordering is determined in the way that vertices with highest current saturation are chosen, where saturation of a vertex (in the context of CCP) is the number of labels, which are used in non-neighbors of the vertex at the particular moment of construction. If there are more vertices with highest saturation, the one with the lowest degree, i.e. the number of adjacent vertices, is taken. The vertices, which remained unlabeled, are labeled randomly with one of the k labels. Both the time and space complexity of Br elaz’s heuristic in this form is $\mathcal{O}(|V|^2)$. However, by using the data structure we describe in Section 4, the space complexity can be decreased to $\mathcal{O}(|E|)$ with runtime $\mathcal{O}(|V|^2\delta)$, where δ is the average degree of graphs’ vertices.

Initialization of Data Structures and the Objective Function. In this paragraph, we explain the details of the step 2 of Algorithm 1. The procedure is given in Algorithm 2. In the step 1, we set all values in Γ to zero. In the step 2, we reset the number of conflicts $J(S)$ and the list of conflicting vertices V_{CONFL} . Then, in the steps 3-8, we incrementally compute the initial values in Γ , $J(S)$ and V_{CONFL} . This is done in the following way. For each vertex $v \in V$, F denotes the number of vertices, which are in a conflict with v . In the step 4, this value is initialized. Then, we iterate over the neighbors of v and in the step 6, we change the corresponding value in Γ . If both v and the neighbor w have the same label, then in the step 7, we exclude w from vertices, which are conflicting with v , thus, decrementing F . In the step 8, if there is at least one vertex in conflict with v , we add v to the list of conflicting vertices V_{CONFL} .

Algorithm 2. Computation of matrix Γ , objective value $J(S)$ and list V_{CONFL}

Computation of matrix Γ and objective value $J(S)$ and list V_{CONFL}

Input: input state S
Output: matrix $\Gamma : |V| \times k$, objective value $J(S)$,
list of conflicting vertices V_{CONFL}

```

1 for all  $v \in V, c \in C$  let  $\Gamma(v, c) = 0$ 
2 set  $J(S) = 0, V_{CONFL} = \emptyset$ 
3 for each  $v \in V$ 
4   set  $F = |V_c|, J(S) = J(S) + |V_c|$ , where  $v \in V_c$ 
5   for each  $w$  such that  $\{v, w\} \in E$ 
6     set  $\Gamma(v, d) = \Gamma(v, d) + 1$ , where  $w \in V_d$ 
7     if  $\exists c [v \in V_c \wedge w \in V_c]$  set  $F = F - 1, J(S) = J(S) - 1$ 
8     if  $F > 0$  set  $V_{CONFL} = V_{CONFL} \cup \{v\}$ 
9 return  $\Gamma, J(S), V_{CONFL}$ 

```

Algorithm 3. The neighborhood exploration strategy

The neighborhood exploration strategy
Input: current state $S = \{V_1, V_2, \dots, V_k\}$ with objective value $J(S)$ and matrix Γ describing its properties, tabu list T , aspiration objective value $J(S_{BEST})$
Output: the move $[v, d]$ to perform

```

1 set  $J(S^*) = \infty$ 
2 for  $i = 1..|V| \times (k - 1)$ 
3   pick uniformly randomly  $v \in V_{CONFL}, v \in V_c$  and  $d \in C$  such that  $v \notin V_d$ 
4    $J(S') = J(S) - |V_c| + \Gamma(v, c) + 1 + |V_d| - \Gamma(v, d)$ 
5   if  $J(S') < J(S_{BEST}) \wedge [v, d] \in T$  return  $[v, d]$ 
6   if  $J(S') \leq J(S) \wedge [v, d] \notin T$  return  $[v, d]$ 
7   if  $J(S') \leq J(S^*) \wedge [v, d] \notin T$  set  $M^* = [v, d], J(S^*) = J(S')$ 
8 return  $M^*$ 

```

The Neighborhood Exploration Strategy. This procedure is performed in the step 5 of Algorithm 1 and is specified by Algorithm 3. Here, $J(S^*)$ denotes the currently best objective value for a solution in the neighborhood of current solution S . In the step 1, this value is reset. In the steps 2-7, the best move $[v, d]$ is chosen from a sample of the neighborhood. In each iteration, the algorithm tries to relabel a conflicting vertex with a new label d . In the step 4, the new objective value is calculated. In the step 5, the algorithm checks, whether the new solution is better than anything else found so far. If yes, the algorithm directly accepts it, even though it might be currently in the tabu list. In the step 6, we handle the case when the move leads to a solution, which is at least as good as the current solution S . If yes, it directly accepts it, too. In the step 7, we handle the case when the solution is currently best from the neighborhood but still not as good as the current solution S . In this case, we only store it as the currently best neighbor M^* but continue with the exploration.

The Tabu Tenure Adjustment. The tabu tenure adjustment is performed in the step 9 of Algorithm 1. It is a very simple procedure but it is controlled by several parameters. The input to this procedure consists of the best objective value $J_{\phi, BEST}$ and worst objective value $J_{\phi, WORST}$, which were recorded during the observation period of ϕ iterations. If the difference $J_{\phi, WORST} - J_{\phi, BEST}$ is lower or equal to a threshold c , then the fluctuation of the objective function indicates cycling and we increment the tabu tenure by a uniformly chosen random number between a and b . Otherwise, we assume that the tabu component is not needed for the moment, thus, we set $t_{tabu} = 0$, until the fluctuation of the objective function does not indicate further cycling. We note that this is almost the same strategy as the FOO tabu scheme known from graph coloring heuristics [2].

4 The Scalability Issue

In this section, we very briefly discuss, how the direct use of Γ function in formula (4) can be used to further improve scalability of TabuCol-CCP.

The standard implementation technique for the Γ function is a matrix of size $|V| \times k$, which obviously requires $\mathcal{O}(k|V|)$ space. Since for sparse graphs, k tends to be close to $|V|$, the space complexity of TabuCol-CCP for this type of graphs behaves practically as $\mathcal{O}(|V|^2)$, which is a problem to store for really large graphs. However, Γ matrix has an interesting property that there are at most $\deg(v)$ non-zero values for vertex v , since there are at most $\deg(v)$ labels that can be possibly used in its neighbors.

Therefore, let us consider the following data structure. For each vertex $v \in V$, we have a list of non-zero values, indexed by the labels. Obviously, this structure requires $\sum_{v \in V} \deg(v) = \mathcal{O}(|E|)$ space, which is much better than $\mathcal{O}(k|V|)$. Furthermore, suppose that these values are sorted according to labels. This opens the possibility to use binary search in this array to verify in $\mathcal{O}(\log \deg(v))$ time, whether there is a non-zero value of $\Gamma(v, c)$ for a particular vertex v and label c or not. For sparse graphs, this might still be an interestingly low computational complexity, although it introduces some overhead. Insertions and deletions require $\mathcal{O}(\deg(v))$ time, however they are not crucial, since they are performed less frequently. We will shortly discuss the impact of this data structure in Section 5.3.

5 Experimental Evaluation

In this section, we present the experimental results of our approach. In the first part, we introduce the experimental settings and the test instances. Then, we give detailed results of our experiments and compare TabuCol-CCP to Brélaz's heuristic and iterated greedy (IG) heuristic, which are currently the most interesting strategies for clique covering in such large graphs. Finally, we give a brief discussion on the scalability of TabuCol-CCP and TabuCol and the impact of the introduced data structure.

5.1 Experimental Settings

In the following experiments, we used TabuCol-CCP with 1 hour time limit to solve instances of clique covering for 5 Erdős-Rényi uniform random graphs² with different values of k . These graphs are considered to be a standard benchmark in many problems, including graph coloring [11]. They are generated by starting with $|V|$ isolated vertices and putting an edge between each pair of vertices with probability $p = \delta$, where δ is the desired density of the graph.

In the version of TabuCol-CCP we evaluated, the period of the objective function observation was set to $\phi = 10^4$, the threshold for tabu incrementation was $c = 2$ and the increment for the tabu tenure was a uniformly chosen integer from $[1, 50]$. These values were determined during a series of preliminary experiments, we believe that better performance can be obtained by further adjusting these parameters. However, we wanted to keep the experimental settings as simple and

² All instances are publicly available at:

<http://www.fiit.stuba.sk/~chalupa/benchmarks/ccp>

general as possible. The maximum tabu tenure was set to $|V|$, to avoid uncontrollable growth of the tabu list. F function was used in the basic matrix variant, the impact of the specialized data structure will be discussed later.

5.2 Detailed Results and Comparison of TabuCol-CCP to Brélaz's Heuristic and Order-Based IG Heuristic

In Table 1, we present detailed experimental results of TabuCol-CCP on five uniform random graphs with up to 3×10^4 vertices and densities 0.1 in the three graphs with up to 10^4 vertices and 0.01 in the case of the two largest graphs. The first two columns of Table 1 contain the names of the graphs, their numbers of vertices and edges and the values of k^* , which are the previously published best results for the graphs in [4], regarding the number of cliques. In the next columns, we have the values of k - the numbers of cliques, which were fixed for the test instance, the success rate for 30 independent runs, the average number of local search iterations in thousands and the average CPU time per run, including unsuccessful runs. The experiments were run on a standard PC machine with an Intel Core i5 3.10 GHz CPU and 4 GB RAM, with a sequential implementation in C++ (g++ compiler was used) being done for a single core.

In the experiments, we began with k equal to the currently best known results, obtained for the instances using the order-based iterated greedy (IG) heuristic. This setting led to a 30/30 success rate in all of the test graphs. Thus, it is highly reliable in reproduction of the current results. Then, by systematically decreasing k and repeatedly performing search in the k -fixed space, we obtained improved solutions. We note that the changes in k were chosen simply according to the experience with previous instances. The values of k , for which TabuCol-CCP was successful, are considerably far away from the initial results obtained by order-based algorithms, which shows much promise of this strategy for this type of graphs. It would probably be interesting to see, how a memetic algorithm or some distributed local search method would perform on instances with such low k . However, this is outside of the aim of this paper.

Last but not least, let us present a brief comparison of TabuCol-CCP to Brélaz's heuristic (BRE) and the order-based IG heuristic with greedy clique covering (IG-GCC). Table 2 shows the numbers of cliques needed to cover graphs by the three approaches. The results of BRE and IG-GCC are taken from [4] (except the largest graph, which is used in this paper for the first time). We can see that the Brélaz's heuristic, although being very fast, is no match for the two stochastic approaches. Although the order-based representation provides a foundation for fast optimization, there seems to be a boundary on k in uniform random graphs, under which it is hard to get using non- k -fixed strategy. In IG-GCC, it seems that although the algorithm reaches very reasonable values of k really quickly, it is hard for it to further decrease k , despite the fact that it is possible. This practically causes that the k -fixed strategy strongly outperforms the non- k -fixed strategy in the quality of results on this type of graphs. In addition, we do not even know, whether the non- k -fixed strategy problems are caused by cycling on very large plateaus or by the fact that the algorithm

Table 1. Detailed results of our heuristic technique TabuCol-CCP on the studied graphs with chosen k , success rate, number of iterations in thousands and average CPU time

G	$ V , E , k^*$	k	succ.	iter. ($\times 10^3$)	CPU
<i>uni</i> f1000_0.1	1000, 49833, 243	243	30/30	231	9 s
		228	30/30	21840	18 m
		227	12/30	56842	48 m
<i>uni</i> f5000_0.1	5000, 1250124, 1066	1066	30/30	72	10 s
		1005	30/30	7117	34 m
		1000	21/30	10091	53 m
		995	1/30	9065	60 m
<i>uni</i> f10000_0.1	10000, 4999336, 2025	2025	30/30	106	31 s
		1950	30/30	3938	42 m
		1940	4/30	4512	59 m
<i>uni</i> f20000_0.01	20000, 2001558, 6387	6387	30/30	2430	29 m
		6300	30/30	3857	51 m
		6280	28/30	4123	56 m
		6260	11/30	4389	59 m
		6255	3/30	4365	60 m
<i>uni</i> f30000_0.01	30000, 4505840, -	9300	30/30	2304	31 m
		9000	3/30	3975	60 m
		8990	1/30	3735	60 m

Table 2. The comparison of the approximations of $\vartheta(G) = \chi(\overline{G})$ for each graph obtained by the Brélaz's heuristic (BRE), the order-based IG with greedy clique covering (IG-GCC) and our TabuCol-CCP heuristic

G	TabuCol-CCP	Brélaz	IG-GCC
<i>uni</i> f1000_0.1	227	299	243
<i>uni</i> f5000_0.1	995	1241	1066
<i>uni</i> f10000_0.1	1940	2326	2025
<i>uni</i> f20000_0.01	6255	7640	6387
<i>uni</i> f30000_0.01	8990	10870	9300

might be stuck. We note that this observation corresponds well with the evidence from the graph coloring domain [5]. However, to be fair, the results of TabuCol-CCP also indicate two flaws. The first one is the running time, which is rather high, although, this is a standard also in graph coloring. The second issue is that TabuCol-CCP did not seem to provide such good results on some other graph classes, including samples from social networks. This might be due to the representation - there might be different k -fixed representations suitable for different graph classes. However, to conclude on the quality and scalability issue, TabuCol-CCP seems to be a promising approach for tackling large sparse random graph, thus, possibly also for other noisy instances.

5.3 Scalability of TabuCol-CCP and TabuCol and the Impact of the Specific Data Structure

At this point, we move on to the discussion on the scalability of TabuCol-CCP and TabuCol. Based on the experiments with TabuCol-CCP and estimates on

memory demands of TabuCol, we can state that both algorithms are useable for *unif20000_0.01*, although this seems to be a marginal case for TabuCol. For *unif30000_0.01*, we encounter a situation, when only demands of TabuCol-CCP would be tractable, Γ matrix would be simply too large with our hardware and software limitations. For larger graphs, also TabuCol-CCP needs to be implemented with the data structure described in Section 4, since Γ matrix grows too fast for practical use. For a graph with 40000 vertices and density 0.01, we obtained with IG that $\vartheta \leq 12139$ and in preliminary experiments, TabuCol-CCP was able to decrease this to at most 11580, which seems encouraging.

To measure the slowdown caused by the fact that recalculation of fitness with the specific data structure needs $\mathcal{O}(\log \deg(v))$ time, we had to consider the smaller graphs from Table 1, for which both Γ as a matrix and the specific data structure were useable. We obtained a slowdown from 9 s to 54 s on (*unif1000_0.1*, $k = 243$), from 10 s to 108 s on (*unif5000_0.1*, $k = 1066$) and from 29 m to 76 m on (*unif20000_0.1*, $k = 6387$). This indicates computational demands, which are still tractable, although a possibility to recalculate fitness in $\mathcal{O}(1)$ time and $\mathcal{O}(|E|)$ space remains to be an interesting open question.

6 Conclusion

We studied a technique for the (vertex) clique covering problem (CCP) using a k -fixed strategy and tabu search known from graph coloring. With our approach, called TabuCol-CCP, we were able to improve the currently best known results, obtained by using the order-based representation of the problem [4], on very large but sparse uniform random graphs with up to 3×10^4 vertices.

We also showed that by adapting a strategy based on Γ neighbor-label function, which is used in graph coloring to recalculate fitness in $\mathcal{O}(1)$ time, we can do the same in k -CCP without modifying the Γ function. This allowed us to design a data structure, which is able to further reduce memory demands of TabuCol-CCP to $\mathcal{O}(|E|)$, with recalculation of fitness done in $\mathcal{O}(\log \deg(v))$ time, where $\deg(v)$ is the degree of the vertex, which is relabeled. This gives a hint, how to further scale the algorithm.

Therefore, these experimental results show a promise not only in quality but also in scalability of this strategy in very noisy instances, such as the uniform random graphs. An open problem is that whether there is a data structure with $\mathcal{O}(|E|)$ space complexity, which would preserve the possibility to do the recalculation of fitness $\mathcal{O}(1)$ in time.

Acknowledgement. The author would like to thank Jiří Pospíchal and the anonymous referees for their very valuable comments, which improved the quality of this work very much. This contribution was supported by Grant Agency VEGA SR under the grant 1/0553/12.

References

1. Behrisch, M., Taraz, A.: Efficiently covering complex networks with cliques of similar vertices. *Theor. Comput. Sci.* 355(1), 37–47 (2006)
2. Blöchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Comput. Oper. Res.* 35(3), 960–975 (2008)
3. Brélaz, D.: New methods to color vertices of a graph. *Commun. ACM* 22(4), 251–256 (1979)
4. Chalupa, D.: On the efficiency of an order-based representation in the clique covering problem. In: Moore, J., Soule, T. (eds.) *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO 2012*, pp. 353–360. ACM, New York (2012)
5. Culberson, J.C., Luo, F.: Exploring the k-colorable landscape with iterated greedy. In: Johnson, D.S., Trick, M. (eds.) *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, pp. 245–284. American Mathematical Society (1995)
6. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. *Comput. Oper. Res.* 33(9), 2547–2562 (2006)
7. Gendreau, M., Soriano, P., Salvail, L.: Solving the maximum clique problem using a tabu search approach. *Ann. Oper. Res.* 41, 385–403 (1993), <http://dx.doi.org/10.1007/BF02023002>
8. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Data reduction and exact algorithms for clique cover. *J. Exp. Algorithmics* 13, 2:2.2–2:2.15 (2009)
9. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. *Computing* 39(4), 345–351 (1987)
10. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Oper. Res.* 39(3), 378–406 (1991)
11. Johnson, D.S., Trick, M.: *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. American Mathematical Society, Boston (1996)
12. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York (1972)
13. Keil, J.M., Stewart, L.: Approximating the minimum clique cover and other hard problems in subtree filament graphs. *Discrete Appl. Math.* 154(14), 1983–1995 (2006)
14. Lü, Z., Hao, J.K.: A Memetic Algorithm for Graph Coloring. *Eur. J. Oper. Res.* 203(1), 241–250 (2010)
15. Porumbel, D.C., Hao, J.K., Kuntz, P.: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Comput. Oper. Res.* 37(10), 1822–1832 (2010)
16. Titiloye, O., Crispin, A.: Quantum annealing of the graph coloring problem. *Discrete Optim.* 8(2), 376–384 (2011)
17. Welsh, D.J.A., Powell, M.B.: An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Comput. J.* 10(1), 85–86 (1967)

Solving the Virtual Network Mapping Problem with Construction Heuristics, Local Search and Variable Neighborhood Descent*

Johannes Inführ and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{infuehr,raidl}@ads.tuwien.ac.at
<http://www.ads.tuwien.ac.at>

Abstract. The Virtual Network Mapping Problem arises in the context of Future Internet research. Multiple virtual networks with different characteristics are defined to suit specific applications. These virtual networks, with all of the resources they require, need to be realized in one physical network in a most cost effective way. Two properties make this problem challenging: Already finding any valid mapping of all virtual networks into the physical network without exceeding the resource capacities is NP-hard, and the problem consists of two strongly dependent stages as the implementation of a virtual network's connections can only be decided once the locations of the virtual nodes in the physical network are fixed. In this work we introduce various construction heuristics, Local Search and Variable Neighborhood Descent approaches and perform an extensive computational study to evaluate the strengths and weaknesses of each proposed solution method.

Keywords: Virtual Network Mapping Problem, Construction Heuristics, Local Search, Variable Neighborhood Descent, Future Internet.

1 Introduction

The Internet has ossified [18]. Core parts of the network protocols have not been updated for more than a decade and the introduction of new services and technology is difficult, time consuming and costly. Improvements to the network protocols, however desirable or necessary, do not see widespread adoption if they could break existing features. Examples include Explicit Congestion Notification [19] or Differentiated Services (a quality of service framework) [3].

The Future Internet research community is currently searching for ways to overcome the ossification of the Internet and network virtualization has been identified as a promising technology to do this [1,2]. With the help of virtualization, changes to the core protocols of the Internet can be deployed in an

* This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT10-027.

incremental and non-disruptive fashion. This idea can be developed even further, if one does not view virtual networks as a necessary crutch to move from one technology to the next, but instead as an integral feature of the network. If multiple virtual networks are present, then each of them can have different properties, different protocols, tailored specifically to a user group. In scientific network testbeds such as GENI [7], PlanetLab [6] or G-Lab [22] network virtualization techniques are already in use to share the underlying network infrastructure (substrate) among different research groups. For a survey on network virtualization, its application and available technologies, see [4].

The Virtual Network Mapping Problem (VNMP) arises in this context. Given are multiple virtual networks (VNs) which need to be realized by using resources present in the substrate. We model the substrate by a directed graph $G = (V, A)$ with node set V and arc set A . The VNs are modeled by the disconnected components of the directed graph $G' = (V', A')$. The set $M \subseteq V' \times V$ defines the allowed mappings between virtual and substrate nodes. By $s(a)$ and $t(a)$, $\forall a \in A \cup A'$ we denote the arc's source and target node, respectively. Each VN node $k \in V'$ requires CPU power $c_k \in \mathbb{N}^+$ (to implement custom protocols, etc.), each arc $f \in A'$ requires bandwidth (BW) $b_f \in \mathbb{N}^+$ and has a maximum allowed delay d_f . Substrate nodes $i \in V$ have an associated CPU power $c_i \in \mathbb{N}^+$, which is used to power the VN nodes mapped to i , but also to route BW. One unit of CPU power is required to route one unit of BW. Substrate arcs $e \in A$ have a BW capacity $b_e \in \mathbb{N}^+$ and a delay $d_e \in \mathbb{N}^+$. The objective is to find a mapping $m : V' \rightarrow V$ of virtual nodes to substrate nodes such that $(k, m(k)) \in M$, $\forall k \in V'$ and the total CPU load on each $i \in V$ caused by mapped virtual nodes and traversing implementations of virtual arcs does not exceed c_i . Furthermore, we have to find for each $f \in A'$ a substrate path $P_f \subseteq A$ leading from $m(s(f))$ to $m(t(f))$ with a delay of at most d_f . The BW capacity of substrate arcs has to be respected by those paths. These are required properties of a valid solution.

We want to implement the virtual networks with as low costs as possible. Every substrate node $i \in V$ has an associated usage cost $p_i^V \in \mathbb{N}^+$ which is paid when at least one VN node is mapped to it. Additionally, every substrate arc $e \in A$ has a usage cost $p_e^A \in \mathbb{N}^+$ which is paid when at least one virtual connection uses it. The sum of substrate node and arc usage costs, the total usage cost C_u , is the objective to be minimized.

As already mentioned, just finding a valid solution to this problem is NP-hard, so we cannot expect efficient heuristic methods to always be able to find valid solutions. For optimization purposes, we want to be able to determine how far solutions are away from validity so we can prefer solutions closer to validity. To do this, every node $i \in V$ can increase its available CPU power by a_i^{CPU} units for the cost of C^{CPU} per unit and every arc $e \in A$ can increase its available BW by a_e^{BW} units for the cost of C^{BW} per unit. The sum of the costs for additional resources in the substrate will be denoted as C_a ; for valid solutions $C_a = 0$. In our experiments we set $C^{\text{CPU}} = 1$ and $C^{\text{BW}} = 5$ to reflect the fact that it is easier to add CPU power to a router than to increase the BW of a data link.

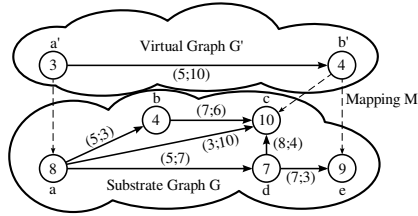


Fig. 1. Example of a VNMP instance

For comparing two solutions, we used lexicographic ordering, i.e., smaller C_a is preferred and if it is equal smaller C_u is preferred.

Figure 1 shows a small VNMP instance. It contains the virtual network graph G' consisting of one VN with two nodes (a' and b'), the substrate graph G (nodes a to e) and the allowed mapping from the virtual network to the substrate nodes (dashed lines). Node labels define the CPU requirements for VN nodes and the available CPU power for substrate nodes. Costs have been omitted for clarity. Arc labels denote bandwidths and delays. Note that in this example, b' actually cannot be mapped to c , even though c offers enough CPU capacity. This is because there is no path from a to c that satisfies the constraints of the virtual connection between a' and b' . Node b cannot be used, because its CPU power of 4 is not enough to route the required BW of 5. The direct connection from a to c does not offer enough BW and the path using d exceeds the delay limit. The only feasible solution to this instance is to map a' to a and b' to e and implement the connection between a' and b' by the path (a, d, e) .

In the following Sections, we will introduce construction heuristics, Local Search and Variable Neighborhood Descent algorithms for solving also larger instances of the VNMP and show that depending on the available run-time, every heuristic can be the best choice. Local Search is the most versatile approach, which depending on configuration can either perform similar to the Construction Heuristics, to Variable Neighborhood Descent or somewhere in-between. We also compare the presented heuristics with the exact method proposed in [15].

2 Related Work

Virtual Network Mapping Problems have received considerable scientific interest in recent years due to their relevance to Future Internet research (e.g. [5,9,11,17,20,21,23,24,25]). The core problem solved in these works is the same: virtual networks have to be realized by means of a physical network. The details however, are always different. This can already be seen when comparing the names of the problems. Typical names include Virtual Network Embedding, Virtual Network Assignment or Network Testbed Mapping. A further area for differences are the resources the virtual networks require. The one demand that is nearly universally considered is bandwidth, but there is no consensus on how

this demand is taken into account. One method is to use traffic bounds to describe a whole range of BW requirements that there has to be a feasible routing for all of them (e.g. [9,17]). Another is to specify the node-to-node communication demand in the form of a traffic matrix (e.g. [23]). If another resource is taken into account, it is the required CPU processing power of each virtual node (e.g. [20,24]). The considered substrate sizes vary between 20 [17] and 100 [25] nodes and are either real or synthetic topologies.

VNMPs have been solved by simulated annealing [21], (quadratic) mixed integer programming [5,9,15,17], approximation algorithms [9], distributed algorithms [11], multicommodity flow algorithms [23,24] or algorithms especially tailored to the considered problem variant [20,25]. To the best of our knowledge, this is the first application and comparison of the trade-off of construction heuristics, Local Search and Variable Neighborhood Descent in the context of virtual network mapping. The VNMP variant solved in this work is very general, since it considers CPU and BW resources, path delays, mapping constraints and the influence of routing overhead on CPU resources. Therefore most algorithms presented in previous work do not apply, with the exception of the exact approach of [15], which we will use for comparisons. Furthermore, we use test instances that are freely available for comparison and designed with a focus on realism, in both size (up to 1000 nodes) and structure.

3 Construction Heuristics

A Construction Heuristic (CH) is used to create solutions to problems by following heuristic rules that guide the construction process towards feasible solutions of high quality. For the VNMP, we can already see that these are conflicting objectives; guiding towards feasibility means spreading resource use across the whole substrate, which causes C_u to be unnecessarily high. Trying to pack VNs densely will most likely lead to high C_a , so some kind of balancing is required. Fortunately, constructing a solution to the VNMP can be split into four different phases that are iterated and in each phase we can focus on different aspects of the solution. The four phases are: selecting a virtual node to map (SVN), selecting a target for the node (TVN), selecting a virtual arc to implement (SVA) and implementing the arc (IVA). Additionally, there can be an emphasis on mapping the nodes (NE) or an emphasis on implementing virtual arcs (AE). With NE, all nodes will be mapped before virtual arcs are implemented. With AE, all implementable virtual arcs will be implemented before a next virtual node is mapped. Since a virtual arc f can only be implemented if $m(s(f))$ and $m(t(f))$ have already been fixed, AE variants will also start by mapping at least two nodes, but then mapping of virtual nodes and implementing virtual arcs will be interleaved instead of strictly sequentially as it is done with NE.

Table 1 lists the considered strategies. Note that for the TVN strategies, only substrate nodes allowed by M are regarded. If a strategy does not find a feasible node, the one with the most free resources is chosen. The SVA strategies only consider implementable virtual arcs. All IVA strategies implement a virtual arc

Table 1. Implemented SVN, TVN, SVA and IVA strategies

Name	Description
SVN1	Selects the next unmapped node of V' .
SVN2	Selects the node with the highest sum of CPU requirement and connected BW.
SVN3	Selects with SVN2 from the VN with highest total CPU and BW requirement that has still unmapped nodes left. Concentrating on one VN when selecting nodes supports AE variants, because virtual arcs become implementable much faster.
SVN4	Selects with SVN2 from the VN with the lowest total sum of allowed delays and unmapped nodes.
TVN1	Maps a virtual node to the first substrate node with enough free CPU capacity.
TVN2	Maps to the first substrate node with enough free CPU capacity to support the CPU requirements and the total connected BW of the virtual node (total CPU load).
TVN3	Maps to the substrate node with the most free CPU capacity. If there are multiple choices, the one with the most free incoming and outgoing BW is used as map target.
TVN4	Maps to the substrate node with enough free resources (w.r.t. total CPU load) and least increase of C_u .
SVA1	Selects the next arc.
SVA2	Selects the arc with the highest BW requirement.
SVA3	Selects arc with the smallest delay.
SVA4	Selects the arc f with the smallest fraction of allowed delay to shortest possible delay between $m(s(f))$ and $m(t(f))$.
IVA1	Arcs have a cost of 0 if they are already used, or their usage cost otherwise. Arcs without enough free BW cost 10^6 .
IVA2-n	The cost of an arc is the sum the fraction of the arcs free BW the virtual arc would use and the fraction of free CPU power the virtual arc would use on the node the substrate arc connects to. This cost is then taken to the power of $n \in \{0.5, 1, 2\}$.

f by finding a Delay Constrained Shortest Path in the substrate from $m(s(f))$ to $m(t(f))$ via the Dynamic Program from [8]. The only difference between the strategies is the calculation of the substrate arc costs, which define the length of a path. If the following strategies define no specific order of nodes or arcs, it is arbitrary.

These strategies result in a total of 512 different construction heuristics, the results of their evaluation can be found in Sect. 7. The strategies were kept simple to keep running times short as the following heuristics build on the best CH variants.

4 Local Search

The basic idea of Local Search (LS) is that a found solution to a problem may be improved by iteratively making small changes. The solutions immediately reachable from a starting solution S are defined by a neighborhood $N(S)$. LS starts with a solution S and replaces it with a better solution from $N(S)$ until no more improvements can be found. For selecting the neighbor, we use the two standard strategies first-improvement (select the first improving solution) and best-improvement (select the best solution from a neighborhood).

Table 2. Implemented neighborhoods for LS

Name	Description
N_1	Removes the implementation of an arc.
N_2	Removes a virtual node and the implementations of its adjacent arcs.
N_3	Removes all virtual nodes and implementations of all virtual arcs of a VN.
N_4	Removes the implementation of all virtual arcs using a specific substrate arc.
N_5	Removes all virtual nodes and the implementation of all arcs using a specific substrate node.
N_6	Like N_2 , but tries mapping the virtual node to all allowed substrate nodes instead of delegating this task to the CH during rebuilding.

The six implemented neighborhoods are listed in Table 2. They all share the common idea that they remove a part of a complete solution (like the implementation of a virtual arc) and then complete the solution again by applying a CH. The neighborhood descriptions will skip this rebuilding step.

For each neighborhood, there is a natural order in which to evaluate the neighbors (e.g., clearing the first substrate node, clearing the second one and so on). However, we might be able to speed up the search process by trying other, more promising, neighbors first. For finding valid solutions, the most promising neighbors are those that might change C_a , e.g., changing the mapping of a virtual node that is mapped to an overloaded substrate node. We will call this strategy `OverloadingFirst`. A more extreme variant of this is `OnlyOverloading`, which only considers the neighbors that `OverloadingFirst` prefers.

5 Variable Neighborhood Descent

The neighborhoods discussed in the previous section can be applied in combination with a variable neighborhood descent (VND) algorithm [10]. A VND utilizes a series of neighborhoods $N_1 \dots N_k$. An initial solution is improved by N_1 until no more improvements can be found, then N_2 is applied to the solution and so on. If N_k fails, VND terminates. If an improved solution is found in some neighborhood, VND restarts with N_1 . We use the neighborhoods of the previous section in two variants: as described and in the `OnlyOverloading` variant, which we will denote with a prime. For example, N'_5 is the neighborhood of all solutions reachable by clearing an overloaded substrate node. Table 3 lists the tested neighborhood configurations.

6 Test Instances

This Section describes the used VNMP instances. The substrates are subgraphs of the NREN (National Research and Education Networks) network [16] which contains European research networks and their interconnects. It's one of the largest freely available networks based on physical network structure (1100 nodes) and has geo-location information embedded which is used for defining meaningful

Table 3. Implemented neighborhood configurations for VND

Name	Description
C_1	$N'_1 N'_2 N'_3 N'_4 N'_5 N'_6 N_1 N_2 N_3 N_4 N_5 N_6$. All neighborhoods, in order of their size.
C_2	$N'_1 N'_2 N'_3 N'_4 N'_5 N'_6$. All OnlyOverloading neighborhoods.
C_3	$N_1 N_2 N_3 N_4 N_5 N_6$. All complete neighborhoods.
C_4	$N_6 N_5 N_4 N_3 N_2 N_1$. Neighborhoods that produce the largest changes first.
C_5	$N'_1 N'_2 N'_3$. Only neighborhoods of C_2 yielding improvements based on preliminary results.
C_6	$N'_3 N'_2 N'_1$. C_5 in reverse order.

Table 4. Properties of the VNMP instances: average number of substrate nodes (V) and arcs (A), virtual nodes (V') and arcs (A'), total usage costs (C) and the average number of allowed map targets for each virtual node ($M_{V'}$)

Size	$ V $	$ A $	$ V' $	$ A' $	C	$M_{V'}$
20	20	40.8	220.7	431.5	1536.0	3.8
30	30	65.8	276.9	629.0	2426.6	4.9
50	50	116.4	398.9	946.9	4298.1	6.8
100	100	233.4	704.6	1753.1	8539.1	11.1
200	200	490.2	691.5	1694.7	17584.2	17.3
500	500	1247.3	707.7	1732.5	44531.8	30.2
1000	1000	2528.6	700.2	1722.8	89958.4	47.2

mapping constraints. The instance set contains substrates of 20 to 1000 nodes, 30 instances of each size. The VN sizes are chosen uniformly at random from $[5, \min(30, 0.3 * |V|)]$. In order to reflect realistic use cases, the VNMP instances contain 10 VNs of each of four different types: Stream, Web, Peer-to-Peer (P2P) and Voice-over-IP (VoIP).

Stream VNs have a tree structure and model video streaming services. They have high BW and CPU requirements but are not delay constrained. Web VNs have a star structure and very low BW and CPU requirements but hard delay constraints. P2P and VoIP VNs have small world structure. P2P VNs have high BW and medium CPU requirements, but no delay constraints. VoIP VNs have medium CPU and BW requirements and moderate delay constraints.

Bandwidth and CPU capacities of the substrates are based on the requirements of a random implementation of all VNs. Table 4 shows the main properties of the instance set, which is available at [14].

7 Results

Each CH, LS and VND variant was tested on the full instance set as described in Section 6. Additionally, each instance was tested with different loads, i.e., reduced numbers of VNs. A load of 0.5 means that only 50% of the VNs of each type were used. Load levels of 0.1, 0.5, 0.8 and 1 were tested, which results in a total of 840 test instances (120 per size). The proposed algorithms have been

run on one core of an Intel Xeon E5540 multi-core system with 2.53 GHz and 3 GB RAM per core. A CPU-time limit of 1000 seconds was applied.

We evaluated each algorithm from two points of view: Capability of finding valid solutions and capability of finding a best solution among all considered algorithms. To evaluate the second aspect, we cannot search for the lowest average C_u values, because higher values might be better if C_a is lower. Therefore we used the following ranking procedure to compare different algorithms: Considering a specific instance, the algorithm that achieves the best solution gets assigned rank 0, the second best rank 1 and so on. Algorithms with the same results share the same rank and no rank is skipped afterwards. The relative rank R_{rel} of an algorithm when solving a particular instance is its rank divided by the highest rank for this instance. Average R_{rel} values can be compared in a meaningful way. For example, an average R_{rel} of 0.1 means that the algorithm in question belongs to the top 10% of all compared algorithms over the compared instances.

7.1 Construction Heuristics

Before we could compare all implemented heuristics, we needed to identify promising CH variants which can be used to generate the initial solution for LS and VND and perform the rebuilding step of the proposed neighborhoods.

Considering the average R_{rel} of each construction heuristic variant over all tested instances, the best construction heuristic (CH1) reached a R_{rel} of 0.093. It used the strategies SVN3, TVN3, SVA4 and IVA1 with AE and was able to find valid solutions to 60.8% of all instances. This strategy is geared towards reducing C_u . For initialization purposes it might be interesting to use a strategy that focuses on finding valid solutions and leave cost reductions to the used neighborhoods, so we changed the IVA to IVA2-1. This variant is denoted with CH2 and is able to find valid solutions to 70.8% of all instances. The results showed that the best CH variants according to R_{rel} exclusively used TVN3, which introduces a strong bias towards validity that might hamper LS and VND during the search for minimal C_u with neighborhoods that remove the mapping of a node. So for the third CH variant (CH3), we changed the TVN of CH1 to TVN4. Both CH1 and CH2 were tested for initialization, all three were tested for rebuilding. This led to 216 LS and 72 VND variants. Now follows their evaluation and comparison.

7.2 Comparing CH, LS and VND

Figure 2 shows the trade-off between low R_{rel} and low run-time for all tested heuristics over all instances. Label (A) marks the best non-dominated construction heuristics. They all use SVN4 and TVN3. They emphasise implementing arcs, so the selected SVA strategy has not a lot of influence. This can be seen here since the two visible configurations are actually multiple configurations using different SVA strategies. The faster but slightly worse clusters use IVA2-1, while the better CH methods use IVA2-2.

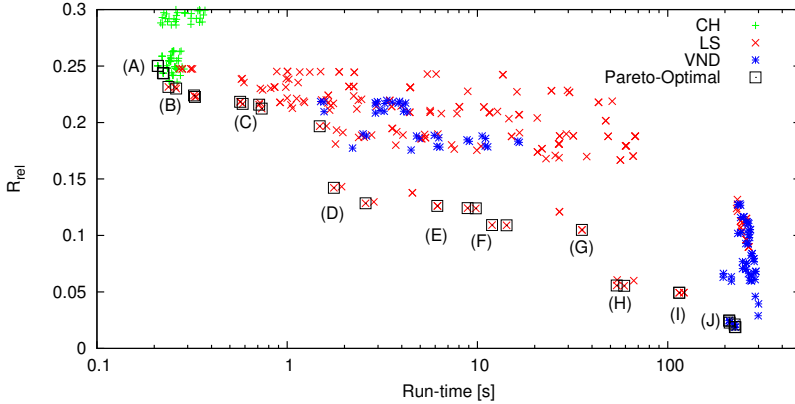


Fig. 2. Pareto-front of the tested heuristics regarding average R_{rel} and run-time over all instances

Label (B) marks the first LS strategies. They use N_2 in the OnlyOverloading variant with first-improvement. Because they use the reduced neighborhoods, they are very fast, even faster than some of the tested CH variants. A marginally better ranking can be achieved by initializing with CH1 instead of CH2. The small visible differences are caused by different rebuilding strategies. The LS variants at (C) use neighborhood N_4 (faster) and N_5 (slower). Otherwise they are equivalent to the better ranked variants at (B).

The run-time jump from (C) to (D) is caused by not using the neighborhoods in the OnlyOverloading variant. Also, starting with the LS configurations at (D), only CH3 is used to rebuild solutions. The effect of using CH3 can be seen comparing the unlabeled LS configuration between (C) and (D) and the faster variant at (D). They are equivalent except for the rebuilding strategy. The variants at (D) use N_3 with first-improvement and OverloadingFirst. Again using CH1 instead of CH2 for initialization causes better ranking but longer run-times. The variants close to the marked ones do not use OverloadingFirst. Variant (E) offers a slight improvement in rank at a high run-time cost by switching to best-improvement. The pattern visible at (D) and (E) is repeated twice with (F) and (G), and (H) and (I). The difference is the used neighborhood, at (F) N_2 is used, at (H) N_5 .

The heuristics at (J) mark the emergence of VND as best solution heuristic. Using VND instead of the best LS variant halves R_{rel} at the cost of doubling the average run-time. The two visible different clusters are caused by the difference between first-improvement (faster) and best-improvement (lower R_{rel}). Both clusters contain VND variants using C_1 and C_3 .

Figure 3 shows the trade-off between solving instances, i.e., just finding a feasible solution and low run-time. Label (A) marks the best non-dominated CHs. They use SVN4, TVN3 and IVA2-1 and AE. Seven percent more instances can be solved when switching to SVN3, marked with (B). The small differences in performance visible at (A) and (B) are caused by different SVA strategies, SVA2 performs better than SVA3.

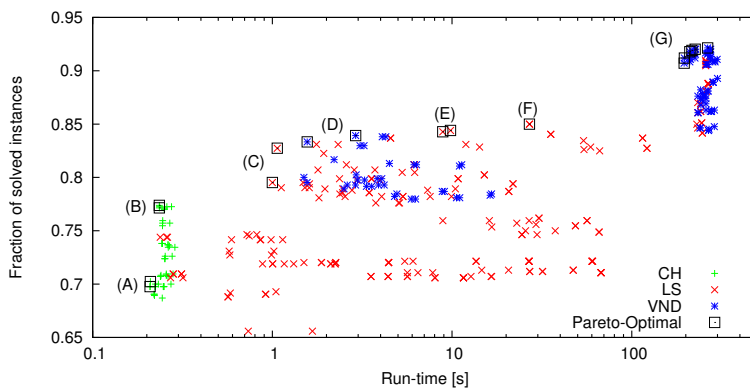


Fig. 3. Pareto-front of the tested heuristics regarding the fraction of solved instances and run-time over all instances

The heuristics at (C) are the first Pareto-optimal LS heuristics. They use N_3 with first-improvement in the OnlyOverloading configuration. Both use CH2 to construct the initial solution. Using CH3 instead of CH1 for rebuilding causes the increase in solved instances. All of the following algorithms use this configuration for initialization and rebuilding.

The first VND variants (using C_6) can be seen at (D). Since they start with the same neighborhood as the LS at (C) but also search other neighborhoods, they perform slightly better than LS. The better but slower variant at (D) uses best-improvement instead of first-improvement.

Once again we can observe the run-time increase caused by considering the complete neighborhoods, this time between (D) and (E). The LS variants at (E) use N_2 with first-improvement and prioritisation (faster) or without (marginally better). The variant at (F) is the same as the faster one at (E), but with best-improvement. The VND configurations C_1 and C_3 can be found at (G). The faster configurations use first-improvement, the others best-improvement.

We further compare the heuristics with the Integer Linear Programming (ILP) approach presented in [15] (with slight modifications to account for differences in the problem definition). For solving the ILP with CPLEX 12.4 [12], we used a time-limit of 10000 seconds and a memory limit of 4 GB. In general, the heuristic methods are able to solve far more instances while requiring only a fraction of the ILP's run-time. On the other hand, the exact approach can solve instances that none of the tested heuristics could and even solved four instances of the largest size to optimality. All instances of size 20 could be solved to optimality with an average run-time of 131 seconds. The best algorithm variants of all classes are compared in Table 5. Among other things, it shows the average relative decrease in C_u required (C_u -Gap) for each algorithm to match the performance of the ILP. This value is only based on instances that could be solved by the considered algorithm. Due to space limitations we cannot show a more detailed analysis here and refer instead to [13].

Table 5. The best algorithms of each class (according to the number of solved instances S or their average R_{rel}), their average runtime t and C_u -Gap over all instances. Bracketed values are the number of instances considered for calculating C_u -Gap.

Algorithm	S	R_{rel}	t [s]	C_u -Gap[%]
CH: SVN4, TVN3, SVA4, IVA1, AE	511	0.236	0.3	24.0 (434)
CH: SVN3, TVN3, SVA1, IVA2-1, AE	650	0.248	0.2	30.8 (512)
LS: N_2 , best-improvement, OverloadingFirst, CH2, CH3	703	0.049	115.0	8.7 (511)
LS: N_6 , best-improvement, OverloadingFirst, CH2, CH1	764	0.096	258.5	18.3 (518)
VND: C_1 , best-improvement, CH2, CH3	773	0.019	226.9	7.1 (519)
VND: C_1 , best-improvement, CH1, CH2	774	0.093	264.7	18.1 (520)
ILP	527	-	2466.0	0.0 (527)

8 Conclusion

In this work, we compared 512 CH, 216 LS and 72 VND algorithms. We could show that for the VNMP, each algorithm class has its application area: CHs for finding solutions fast, VND for finding the best solutions and LS covering the range in-between, depending on the used neighborhoods. For CHs, the most important strategy is the target choice for virtual nodes, so this is a clear area of interest for future improvements. For LS we could see that best-improvement works slightly better than first-improvement, but at a significant run-time cost. Reducing the neighborhood size also reduced the performance, but brought the execution speed into CH territory. VND benefited from the reduced neighborhoods too when searching for valid solutions. For LS, the initialization strategy has a pronounced influence on the result. For the best ranking, CH1 was used while CH2 was the better strategy when comparing the number of found valid solutions. The discussed VND variants produced the best results, but at a high run-time cost. Some fine-tuning with respect to the neighborhood configurations is still necessary.

References

1. Anderson, T., Peterson, L., Shenker, S., Turner, J.: Overcoming the Internet impasse through virtualization. *Computer* 38(4), 34–41 (2005)
2. Berl, A., Fischer, A., de Meer, H.: Virtualisierung im Future Internet. *Informatik-Spektrum* 33, 186–194 (2010)
3. Carlson, M., Weiss, W., Blake, S., Wang, Z., Black, D., Davies, E.: An architecture for differentiated services. IETF, RFC 2475 (1998)
4. Chowdhury, N.M.K., Boutaba, R.: A survey of network virtualization. *Computer Networks* 54(5), 862–876 (2010)
5. Chowdhury, N., Rahman, M., Boutaba, R.: Virtual network embedding with coordinated node and link mapping. In: *INFOCOM 2009*, pp. 783–791 (2009)
6. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.* 33, 3–12 (2003)

7. GENI.net: Global Environment for Network Innovations (2012), <http://www.geni.net>
8. Gouveia, L., Paias, A., Sharma, D.: Modeling and solving the rooted distance-constrained minimum spanning tree problem. *Computers & Operations Research* 35(2), 600–613 (2008)
9. Gupta, A., Kleinberg, J., Kumar, A., Rastogi, R., Yener, B.: Provisioning a virtual private network: a network design problem for multicommodity flow. In: STOC 2001, pp. 389–398 (2001)
10. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. *European Journal of Operational Research* 130(3), 449–467 (2001)
11. Houidi, I., Louati, W., Zeghlache, D.: A distributed virtual network mapping algorithm. In: IEEE International Conference on Communications, ICC 2008, pp. 5634–5640 (2008)
12. IBM ILOG: CPLEX 12.4, <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer>
13. Inführ, J., Raidl, G.R.: Data supplement, <https://www.ads.tuwien.ac.at/projects/optFI-wiki/images/a/a7/DataSupplement.pdf>
14. Inführ, J., Raidl, G.R.: The Virtual Network Mapping Problem benchmark set, <https://www.ads.tuwien.ac.at/projects/optFI/>
15. Inführ, J., Raidl, G.R.: Introducing the Virtual Network Mapping Problem with Delay, Routing and Location Constraints. In: Pahl, J., Reiners, T., Voß, S. (eds.) INOC 2011. LNCS, vol. 6701, pp. 105–117. Springer, Heidelberg (2011)
16. Knight, S., Nguyen, H., Falkner, N., Roughan, M.: Realistic network topology construction and emulation from multiple data sources. Tech. rep., The University of Adelaide (2012)
17. Lu, J., Turner, J.: Efficient mapping of virtual networks onto a shared substrate. Tech. rep., Washington University in St. Louis (2006)
18. National Research Council: Looking Over the Fence at Networks. National Academy Press (2001)
19. Ramakrishnan, K.K., Floyd, S., Black, D.: The addition of explicit congestion notification (ECN) to IP. IETF, RFC 3168 (2001)
20. Razzaq, A., Rathore, M.S.: An approach towards resource efficient virtual network embedding. In: Proceedings of the 2010 2nd International Conference on Evolving Internet, INTERNET 2010, pp. 68–73. IEEE Computer Society (2010)
21. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. *Special Interest Group on Data Communication Comput. Commun. Rev.* 33(2), 65–81 (2003)
22. Schwerdel, D., Günther, D., Henjes, R., Reuther, B., Müller, P.: German-Lab Experimental Facility. In: Berre, A.J., Gómez-Pérez, A., Tutschku, K., Fensel, D. (eds.) FIS 2010. LNCS, vol. 6369, pp. 1–10. Springer, Heidelberg (2010)
23. Szeto, W., Iraqi, Y., Boutaba, R.: A multi-commodity flow based approach to virtual network resource allocation. In: Global Telecommunications Conference, GLOBECOM 2003, vol. 6, pp. 3004–3008. IEEE (2003)
24. Yeow, W.L., Westphal, C., Kozat, U.: Designing and embedding reliable virtual infrastructures. In: Proceedings of the Second ACM Special Interest Group on Data Communication Workshop on Virtualized Infrastructure Systems and Architectures, VISA 2010, pp. 33–40. ACM, New York (2010)
25. Zhu, Y., Ammar, M.: Algorithms for assigning substrate network resources to virtual network components. In: Proceedings of the 25th IEEE International Conference on Computer Communications, INFOCOM 2006, pp. 1–12 (2006)

The Generate-and-Solve Framework Revisited: Generating by Simulated Annealing

Rommel D. Saraiva¹, Napoleão V. Nepomuceno², and Plácido R. Pinheiro²

¹ State University of Ceará (UECE), Fortaleza, Brazil

² Graduate Program in Applied Informatics, University of Fortaleza (UNIFOR),
Fortaleza, Brazil

Abstract. The Generate-and-Solve is a hybrid framework to cope with hard combinatorial optimization problems by artificially reducing the search space of solutions. In this framework, a metaheuristic engine works as a generator of reduced instances of the problem. These instances, in turn, can be more easily handled by an exact solver to provide a feasible (optimal) solution to the original problem. This approach has commonly employed genetic algorithms and it has been particularly effective in dealing with cutting and packing problems. In this paper, we present an instantiation of the framework for tackling the constrained two-dimensional non-guillotine cutting problem and the container loading problem using a simulated annealing generator. We conducted computational experiments on a set of difficult benchmark instances. Results show that the simulated annealing implementation overachieves previous versions of the Generate-and-Solve framework. In addition, the framework is shown to be competitive with current state-of-the-art approaches to solve the problems studied here.

Keywords: Combinatorial Optimization, Hybrid Metaheuristics, Cutting and Packing, Simulated Annealing, Integer Programming.

1 Introduction

Combinatorial optimization is concerned with problems for which we need to find an optimal solution over a well-defined discrete space of potential solutions. Combinatorial Optimization Problems (COPs) arise in many real-world applications, such as vehicle routing, network design, machine scheduling, etc. These problems are usually NP-hard, meaning that there do not exist deterministic polynomial-time algorithms to solve these problems, unless $P = NP$. In practice, it means that often instances of practical interest of COPs cannot be solved to proven optimality in reasonable time. Due to their practical and theoretical relevance, many different solution methods have been proposed for tackling COPs.

In recent years, the hybridization of metaheuristics (e.g., genetic algorithms, simulated annealing, and tabu search) with mathematical programming techniques (e.g., branch-and-bound, constraint programming, and cutting plane algorithms) has originated very powerful methods one should recur to confront hard

COPs, especially because they usually incorporate complementary strengths from different optimization techniques. These methods are commonly referred to as “hybrid metaheuristics” [1].

In this paper, we investigate a novel instantiation of the Generate-and-Solve (GS) optimization framework recently proposed in [2][3]. This framework implements an iterative methodology, which complies directly with the aforementioned hybridization precept, to achieve approximate solutions to hard COPs. The methodology can be summarized as follows: A metaheuristic engine works as a generator of reduced instances of the COP at hand, which are formulated as Integer Linear Programming (ILP) models. These instances, in turn, can be more easily handled by an ILP solver. Besides providing a feasible (optimal) solution to the original problem, the performance measures accomplished by the ILP models are interpreted as score values by the metaheuristic, thus guiding its search for promising reduced instances.

Thanks to the good levels of flexibility and adaptability exhibited by genetic algorithms, this class of heuristics has been the usual choice for the metaheuristic engine of the GS framework [2][3][4][5]. Differently, we investigate an instantiation of the framework for tackling the constrained two-dimensional non-guillotine cutting problem and the container loading problem using a simulated annealing algorithm [6]. We conducted computational experiments on a set of difficult benchmark instances. Results show that the simulated annealing implementation overachieves previous versions of the GS framework.

Moreover, the GS framework is shown to be fairly competitive – in terms of effectiveness – with current state-of-the-art approaches to solve the problems studied here. On the one hand, although these alternative approaches are more time-efficient, they commonly rely on problem-specific knowledge to obtain good solutions. On the other hand, since many COPs can be straightforwardly formulated as integer programs, the conceptual components of the GS framework can be easily customized to cope well with different classes of COPs.

The paper is organized as follows. In Section 2, we present some recent advances in solving the constrained two-dimensional non-guillotine cutting problem and the container loading problem. We then propose an instantiation of the GS framework for tackling these problems in Section 3. Section 4 is devoted to discussing our computational results on benchmark problem instances. Some final remarks and comments on future work conclude the paper with Section 5.

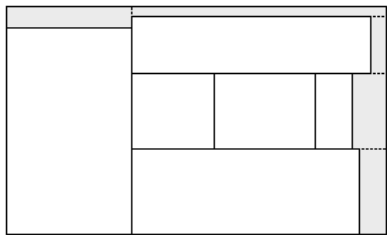
2 Cutting and Packing Problems

According to Wascher [7], Cutting and Packing (C&P) problems can be summarized as follows: Consider two sets of elements, namely a set of large objects and a set of small items, which are defined in a particular number of geometric dimensions. Select some or all small items, group them into one or more subsets and assign each of the resulting subsets to one of the large objects such that (1) all small items of the subset lie entirely within the corresponding large object, (2) the small items do not overlap, and (3) a given objective function is optimized. Special types of C&P problems are characterized by additional properties.

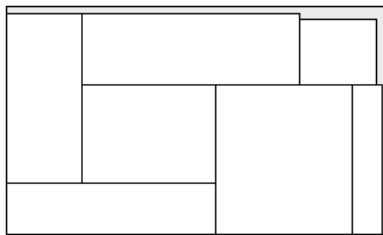
In what follows, we focus on two particular C&P problems belonging to the class of NP-hard combinatorial optimization problems: the constrained two-dimensional non-guillotine cutting problem and the container loading problem.

2.1 The Constrained Two-Dimensional Non-guillotine Cutting Problem

The constrained two-dimensional non-guillotine cutting problem consists of cutting small rectangular pieces of predetermined types from one large rectangular object so as to maximize the value of the pieces cut. The number of pieces of each type that are cut must lie within prescribed limits. Each piece has a fixed orientation and must be cut with its edges parallel to the edges of the large object. The pattern must be a non-guillotine cutting pattern (see Fig.1).



(a) A guillotine cutting pattern.



(b) A non-guillotine cutting pattern.

Fig. 1. The cutting pattern (a) can be obtained by means of successive guillotine cuts, i.e. cuts that run from one edge of a rectangle to its opposite edge. Conversely, the cutting pattern (b) cannot be produced with the use of guillotine cuts.

Several heuristic algorithms to solve this problem have been proposed in the last few years. Beasley [8], for example, introduced a population heuristic algorithm, based on insight from a mathematical formulation of the problem, in which intermediate solutions explicitly state coordinates of the pieces. A penalty in the objective function avoids overlap. The algorithm is capable of efficiently generating good solutions for typical test problems taken from the literature and a number of large randomly generated problems. However, the heuristic is unable to reproduce known optimal solutions for smaller instances.

In [9], Alvarez et al proposed a GRASP algorithm that combines a constructive phase and an improvement phase. In the constructive phase, a solution is built step by step by adding a new piece to a partial cutting pattern. The selection of the piece is based on a greedy function and subjected to a randomization process. In the improvement phase, the solution generated at the constructive phase is used as starting point for a local search procedure that considers different improvement alternatives. The authors presented a tabu search algorithm in [10]. Both GRASP and tabu-search algorithms were able to achieve very impressive results on the data used in [8].

In [11], Egeblad and Pisinger proposed a simulated annealing based approach that makes use of a pair of sequences to represent a cutting pattern. In each iteration, the sequence pair is slightly modified and transformed to a cutting pattern in order to evaluate its objective value. Improving solutions are promptly accepted while non-improving solutions are accepted with probability that decreases over time. This approach is generally able to reproduce the results of exact algorithms with similar running times.

More recently, the application of the GS hybrid framework to solve the constrained two-dimensional non-guillotine cutting problem has provided competitive solutions [3][4][5]. These studies are discussed in more details in Section 3.

2.2 The Container Loading Problem

The container loading problem consists of packing boxes into a container. Given the dimensions of the container and boxes which need to be loaded, the problem aims at finding an arrangement of boxes that optimizes a given objective function - in general, the maximum volume of the loaded boxes. In addition to the geometric constraints, other restrictions can also be considered, such as boxes orientation and cargo stability.

This problem has already been confronted by means of different heuristic approaches. In one of the best known studies on this problem [12], George and Robinson described a wall-building procedure to create layers across the depth of the container. The loading is dictated by a list of priorities that establishes the order of allocation of the boxes and the size of each layer. Similar problem-specific heuristics were then proposed by Loh and Nee [13], Bischoff and Ratcliff [14], and Bischoff et al [15]. The instances presented in [13] are commonly used for benchmarking purposes.

In [16], Gehring and Bortfeldt proposed a genetic algorithm with a stack building heuristic. First a set of stable towers of boxes are generated by the greedy method and, then, the two-dimensional problem of arranging the box towers on the floor of the container is solved by means of the genetic algorithm. The authors also presented a tabu search algorithm with cuboid arrangement heuristic [17] and a wall building heuristic in a genetic algorithm [18]. In [19], Eley designed a similar approach in which initially a greedy heuristic is used to build homogeneous blocks of identically orientated items and, then, a tree search is responsible for improving the solutions provided by the greedy heuristic.

More recently, Liang et al [20] presented a successful hybrid methodology in which first an ant colony optimization algorithm constructs box towers and, after that, a genetic algorithm defines the sequence of arrangement of the towers built in the first phase. Conversely, Yap et al [21] conceived a two-phased ant colony optimization in which a tower building approach is used as the inner heuristic. An instantiation of the GS hybrid framework to solve the container loading problem as well appeared in [2].

3 The Generate-and-Solve Framework

It is well known that exact and heuristic techniques present pros and cons when dealing with hard COPs. In fact, the application of an exact method whenever possible is commonly the procedure of choice. However, as the size and complexity of the optimization problems at hand increase, exact algorithms become prohibitive. In these cases, the use of heuristic components within a solution framework may be very helpful.

Following this general idea, the Generate-and-Solve framework prescribes the integration of two distinct conceptual components: the Generator of Reduced Instances (GRI) and the Solver of Reduced Instances (SRI), as illustrated in Fig. 2. An exact method (e.g., ILP solver) encapsulated in the SRI component is in charge of solving reduced instances of the original problem (i.e., subproblems) that still preserve the original problem's conceptual structure. Thus, a feasible solution to a given subproblem will also be a feasible solution to the original problem. At a higher level, a metaheuristic component (e.g., a genetic algorithm) works on the complementary optimization problem of generating the reduced instance which, when submitted to the SRI, brings about the best feasible solution. The objective function values of the solutions realized by the SRI are in turn used as figure of merit (fitness) of their associated subproblems, thus guiding the metaheuristic search process.

The GS hybrid framework was originally applied to solve the container loading problem [2]. Satisfactory results obtained on the data instances by Bischoff and Ratcliff [14] soon led to an instantiation of the framework to the constrained two-dimensional non-guillotine cutting problem [3]. In these studies, the method of choice for the Generator of Reduced Instances was a genetic algorithm and a LINGO 8.0 DLL for the Solver of Reduced Instances. Although yielding

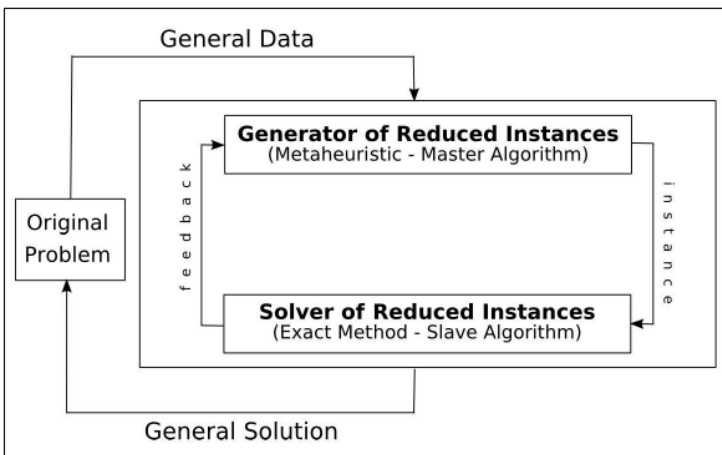


Fig. 2. The Generate-and-Solve framework

high-quality results, the authors reported that some reduced instances could not be solved, or even generated, due to the complexity of the subproblems and/or to computational limitations of the solver.

This fact raised the suspicion that the original version of the GS is hampered by a serious drawback, namely, its tendency to produce subproblems of increasing complexity too prematurely. This drawback is circumvented with the adoption of the density control operator mechanism proposed in [4], whose role is to adaptively control the increase in the size of the generated subproblems so as to allow a much steadier progress towards a better solution. Besides, the authors made use of the IBM ILOG CPLEX solver in a way as to enhance the performance of the SRI. Finally, in order to improve the performance of the GRI component, the utilization of a uniform order-based crossover operator was investigated in [5].

In what follows, we present a new instantiation of the framework which makes use of a simulated annealing engine to generate the reduced instances.

3.1 Problem Formulation

To formulate the constrained two-dimensional non-guillotine cutting problem, we resort to the ILP model proposed in [22]. Consider a set of pieces grouped into m types. For each item type i , characterized by its length l_i , width w_i , and value v_i , there is an associated number of pieces b_i . Consider also a large object that has (L, W) as its length and width dimensions respectively. The pieces should be cut orthogonally from the object. Each 0-1 variable u_{ide} represents the decision of whether to cut or not a piece of type i at the coordinate (d, e) .

$$u_{ide} = \begin{cases} 1, & \text{if a piece of type } i \text{ is put at position } (d, e), \\ 0, & \text{otherwise.} \end{cases}$$

The elements d and e belong, respectively, to the following discretization sets:

$$X = \{x \mid x = \sum_{i=1}^m \alpha_i l_i, x \leq L - \min_{i=1, \dots, m} \{l_i\}, \alpha_i \geq 0, \alpha_i \in \mathbb{Z}^+\}$$

$$Y = \{y \mid y = \sum_{i=1}^m \beta_i w_i, y \leq W - \min_{i=1, \dots, m} \{w_i\}, \beta_i \geq 0, \beta_i \in \mathbb{Z}^+\}$$

To avoid the overlapping of pieces, the incidence matrix g_{idepq} is defined as:

$$g_{idepq} = \begin{cases} 1, & \text{if } d \leq p \leq d + l_i - 1 \text{ and } e \leq q \leq e + w_i - 1, \\ 0, & \text{otherwise.} \end{cases}$$

which has to be computed a priori for each type i , for each coordinate (d, e) , and for each coordinate (p, q) .

Finally, the constrained two-dimensional non-guillotine cutting problem can be formulated as the following standard ILP model.

$$\max \sum_{i=1}^m \sum_{d \in X} \sum_{e \in Y} v_i u_{ide} \tag{1a}$$

$$\text{s.t.} \sum_{i=1}^m \sum_{d \in X} \sum_{e \in Y} g_{idepq} u_{ide} \leq 1 \quad \forall p \in X, \forall q \in Y \tag{1b}$$

$$\sum_{d \in X} \sum_{e \in Y} u_{ide} \leq b_i \quad i = 1, \dots, m \tag{1c}$$

$$u_{ide} \in \{0, 1\} \quad i = 1, \dots, m, \forall d \in X, \forall e \in Y \tag{1d}$$

This model contains $O(m|X||Y|)$ variables and $O(|X||Y|)$ constraints. Since the product of the cardinalities of the discretization sets X and Y may be too large for some problem instances, the number of constraints and variables can easily reach the order of millions, which discourages the use of classical Integer Linear Programming techniques.

Reduced instances of the original problem can be generated by excluding some of the piece types and/or some of the positions demarcated by the discretization sets. In this work, in particular, the reducible structure chosen amounts solely to the coordinates (d, e) from the discretization sets X and Y .

A straightforward model extension to the container loading problem can be found in [2]. It includes the components needed to take into account the height dimension of the container, particularly a new discretization set Z .

3.2 Generator of Reduced Instances

A simulated annealing algorithm has been implemented to account for the GRI task. Each configuration of the algorithm encodes a reduced instance of the original problem. The configuration is represented by a binary structure whose size is determined by the sum of the cardinalities of the discretization sets. Each bit represents one element of a discretization set, as shown in Figure 3. Only elements whose corresponding bits have value of 1 will effectively take part in the reduced instance to be solved by the exact method.

Therefore, the subproblem generated is an ILP model containing all the constraints present in the complete formulation, but only a subset of its decision

Discretization sets:	X = {0, 3, 5, 6, 8, 9} Y = {0, 4, 7, 8}
Binary structure:	x1 x2 x3 x4 x5 x6 y1 y2 y3 y4 1 0 0 1 0 1 1 0 0 1
Discretization subsets:	X' = {0, 6, 9} Y' = {0, 8}

Fig. 3. Binary structure of a configuration

variables. Thus, it can be seen that the subproblem retains the constraint structure of the original problem, along with the property that each solution to the subproblem will also constitute a feasible solution to the original problem. The solution value attained for each subproblem provides the fitness value of the corresponding configuration in the simulated annealing generator.

The initial configuration S_0 is obtained in the following manner: (1) we randomly choose one piece and (2) we set to 1 only the bits whose corresponding elements are multiples of the respective dimension (i.e., length, width, or height) of the selected piece. At each iteration of the simulated annealing algorithm, a random perturbation that flips one correlative bit of each discretization set is performed in the configuration S_{i-1} to obtain the configuration S_i . We then follow the original description of the simulated annealing process [6], where the algorithm replaces the current configuration by the new configuration with a probability that depends on the difference between the corresponding fitness values and a global parameter T , called the temperature. A maximum number of iterations n , dependent on the initial temperature T_0 and the cooling stepsize, and a runtime limit t are used together as stopping criterion.

4 Computational Experiments

Computations were carried out on a desktop machine with a 3.60 GHz Intel i7 CPU and 8GB RAM. We adopt IBM ILOG CPLEX 12.4 as underlying solver of the SRI. The GRI runs a simulated annealing algorithm implemented in Java. After preliminary experiments, we set the initial temperature $T_0 = 500$ which is decreased by 5 in each iteration of the simulated annealing algorithm, leading to a maximum number of iterations $n = 100$.

We conducted a series of experiments on benchmark problems from the literature. We ran the framework 10 times for each problem instance. The computational results are discussed in what follows.

4.1 Results for the Constrained Two-Dimensional Non-guillotine Cutting Problem

For the constrained two-dimensional non-guillotine cutting problem, we set a maximum limit on the execution time of the GS framework to 1800 seconds. In addition, we assume a time limit of 120 seconds of computation for each subproblem submitted to the SRI.

Since classical instances found in [23] can be solved through the application of IBM ILOG CPLEX 12.4 alone, we concentrate our analysis on two classes of instances from the literature commonly referred to as *okp* and *ngcuts*.

In Table 1, we present a comparison of the results achieved with different implementations of the GS framework. GS-SA denotes our work. *Best* and *Average* denote, respectively, the best solution among those produced by all runs of the respective technique and the average solution value. The last instance is the only one for which the best solution found by GS-SA is worst than previous

Table 1. Comparative results of different GS framework implementations

Instance	Nepomuceno et al [3]		Saraiva and Pinheiro [5]		Pinheiro et al [4]		GS-SA	
	Best	Average	Best	Average	Best	Average	Best	Average
okp1	27,360	–	27,539	–	27,539	27,506	27,589	27,513
okp2	21,888	–	22,502	–	22,502	22,235	22,502	22,286
okp3	23,743	–	24,019	–	24,019	23,932	24,019	23,904
okp4	32,018	–	32,893	–	32,893	32,441	32,893	32,304
okp5	27,923	–	27,923	–	27,923	26,601	27,923	26,789
ngcuts1-1	–	–	27,888	–	28,032	27,918	28,032	27,987
ngcuts1-2	–	–	28,946	–	28,946	28,268	28,946	28,405
ngcuts1-3	–	–	27,966	–	27,966	27,932	27,966	27,926
ngcuts1-4	–	–	28,494	–	28,494	28,117	28,494	28,458
ngcuts1-5	–	–	28,677	–	28,677	28,677	28,677	28,677
ngcuts3-1	–	–	–	–	28,315	28,271	28,315	28,041
ngcuts3-2	–	–	–	–	28,046	27,905	28,046	27,942
ngcuts3-3	–	–	–	–	28,877	28,777	28,889	28,848
ngcuts3-4	–	–	–	–	26,604	26,506	26,868	26,447
ngcuts3-5	–	–	–	–	27,787	27,558	27,531	27,396

versions of the framework. In addition, taking into account the maximum limit on the execution time of four hours used in [4], GS-SA presents a remarkable improvement of performance in terms of time efficiency.

GS-SA has noteworthy effectiveness, as evidenced in Table 2 where we present a comparison with some state-of-the-art heuristics, having the *okp* class of problems as target. Except for the instance *okp1*, GS-SA was able to find optimal solutions for all instances of this class. Only Valdes et al [10] attain the optimum for all *okp* instances. It is important to say that these alternative approaches rely on problem-specific knowledge and are usually more time-efficient.

Table 2. Comparative analysis of state-of-the-art algorithms

Instance	Beasley [8]	Valdes et al [9]	Valdes et al [10]	Egeblad and Pisinger [11]	GS-SA	Optimum
		et al [9]	et al [10]	Pisinger [11]		
okp1	27,486	27,589	27,718	27,718	27,589	27,718
okp2	21,976	21,976	22,502	22,214	22,502	22,502
okp3	23,743	23,743	24,019	24,019	24,019	24,019
okp4	31,269	32,893	32,893	32,893	32,893	32,893
okp5	26,332	27,923	27,923	27,923	27,923	27,923
Average	26,161	26,825	27,011	26,953	26,985	27,011

Table 3. Comparative analysis of state-of-the-art algorithms

Instance	Bischoff and Bischoff		Gehring and Bortfeldt		Bortfeldt and Gehring		Liang		Yap	GS-SA
	Ratcliff [14]	et al [15]	Bortfeldt [16]	Gehring [17]	Gehring [18]	Eley [19]	et al [20]	et al [21]		
LN01	62.5	62.5	62.5	62.5	62.5	62.5	62.5	62.5	62.5	
LN02	90.0	89.7	89.5	96.6	89.8	90.8	89.7	87.9	89.9	
LN03	53.4	53.4	53.4	53.4	53.4	53.4	53.4	53.4	53.4	
LN04	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0	
LN05	77.2	77.2	77.2	77.2	77.2	77.2	77.2	77.2	77.2	
LN06	83.1	89.5	91.1	91.2	92.4	87.9	91.4	91.0	87.1	
LN07	78.7	83.9	83.3	84.7	84.7	84.7	84.6	83.8	82.8	
LN08	59.4	59.4	59.4	59.4	59.4	59.4	59.4	59.4	59.4	
LN09	61.9	61.9	61.9	61.9	61.9	61.9	61.9	61.9	61.9	
LN10	67.3	67.3	67.3	67.3	67.3	67.3	67.3	67.3	67.3	
LN11	62.2	62.2	62.2	62.2	62.2	62.2	62.2	62.2	62.2	
LN12	78.5	76.5	78.5	78.5	78.5	78.5	78.5	78.5	78.5	
LN13	78.1	82.3	85.6	84.3	85.6	85.6	85.6	85.6	85.6	
LN14	62.8	62.8	62.8	62.8	62.8	62.8	62.8	62.2	62.8	
LN15	59.5	59.5	59.5	59.5	59.5	59.5	59.5	59.5	59.5	
Average	68.6	69.5	69.9	70.4	70.1	69.9	70.0	69.9	69.7	

4.2 Results for the Container Loading Problem

To cope with the container loading problem, we employ a layer construction component proposed in [2] in which the problem is solved stepwise. At each iteration, a new layer is created across the depth of the container and solved by the hybrid framework. The depth of each layer is set according to the largest dimension among the boxes that are still available. In the last iteration, if the length of the free space becomes smaller than the depth of the layer, we incorporate the residual space to this layer. For each layer, we set a time limit of 1200 seconds on the execution time of the GS framework. Besides, we set a time limit of 180 seconds of computation for each subproblem submitted to the SRI.

Experiments are conducted on a benchmark data set usually referred to as *LN* [13], which consists of 15 problem instances, each with a different number of box type and container's dimensions. We do not find in the literature results to these particular instances obtained by the application of the GS framework.

In Table 3, we present a comparison of the results achieved by some state-of-the-art methodologies to solve the container loading problem. The values in this table represent the percentage of volume utilization. Values in boldface indicates that GS-SA solutions are proven optimal. To sum up, with regard to the average volume utilization, it can be stated that the GS-SA did not achieve the solution quality of some state-of-the-art methodologies when applied to these particular instances. Regardless, GS-SA attained proven optimal solutions for 12 of 15 of these instances and the average percentage of volume utilization is only 0.7% worse compared to the best value found in the literature [17].

5 Conclusion and Future Work

In this work, we presented a novel instantiation of the Generate-and-Solve framework. Briefly, we substitute the metaheuristic engine (*viz.* genetic algorithm) used in previous versions of the framework for a simulated annealing algorithm. We tackled two C&P problems: the constrained two-dimensional non-guillotine cutting problem and the container loading problem.

Computational results on difficult benchmark instances show that the simulated annealing implementation overachieves previous versions of the Generate-and-Solve framework in terms of efficiency and effectiveness. We understand that, without applying recombination such as the typical crossover operator of genetic algorithms, the simulated annealing algorithm prevents the disruption of the building blocks of good solutions.

In addition, the framework is shown to be competitive with current state-of-the-art approaches to solve these problems. This is particularly remarkable because these alternative approaches commonly rely on problem-specific knowledge to obtain good solutions. On the contrary, the GS framework can potentially cope well with different classes of combinatorial optimization problems.

As future work, we shall apply the hybrid framework to other classes of combinatorial optimization problems and investigate other metaheuristics. We also envisage to tackle the container loading problem through a two-phase optimization methodology, such as the prominent approaches by Gehring and Bortfeldt [16] [17], in which first a set of towers of boxes are generated and, then, the two-dimensional problem of arranging the box towers on the floor of the container could be solved by means of the GS framework.

Acknowledgment. This work has been financially supported by CNPq/Brazil via research grant #305844/2011-3. The authors also acknowledge IBM for making the IBM ILOG CPLEX Optimization Studio available to the academic community.

References

1. Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M. (eds.): Hybrid Metaheuristics: An Emerging Approach to Optimization. SCI, vol. 114. Springer, Heidelberg (2008)
2. Nepomuceno, N., Pinheiro, P., Coelho, A.L.V.: Tackling the Container Loading Problem: A Hybrid Approach Based on Integer Linear Programming and Genetic Algorithms. In: Cotta, C., van Hemert, J. (eds.) EvoCOP 2007. LNCS, vol. 4446, pp. 154–165. Springer, Heidelberg (2007)
3. Nepomuceno, N., Pinheiro, P., Coelho, A.L.V.: A Hybrid Optimization Framework for Cutting and Packing Problems: Case Study on Constrained 2D Non-guillotine Cutting. In: Cotta, C., van Hemert, J. (eds.) Recent Advances in Evolutionary Computation for Combinatorial Optimization. SCI, vol. 153, pp. 87–99. Springer, Heidelberg (2008)

4. Pinheiro, P.R., Coelho, A.L.V., de Aguiar, A.B., Bonates, T.O.: On the concept of density control and its application to a hybrid optimization framework: Investigation into cutting problems. *Computers & Industrial Engineering* 61(3), 463–472 (2011)
5. Saraiva, R.D., Pinheiro, P.R.: A novel application of crossover operator to a hybrid optimization framework: Investigation into cutting problems. In: CEC, pp. 1–7. *IEEE* (2012)
6. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* 220(4598), 671–680 (1983)
7. Wäscher, G., Haußner, H., Schumann, H.: An improved typology of cutting and packing problems. *European Journal of Operational Research* 183(3), 1109–1130 (2007)
8. Beasley, J.E.: A population heuristic for constrained two-dimensional non-guillotine cutting. *European Journal of Operational Research* 156(3), 601–627 (2004)
9. Valdes, R.A., Parreño, F., Tamarit, J.M.: A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. *The Journal of the Operational Research Society* 56(4), 414–425 (2005)
10. Valdes, R.A., Parreño, F., Tamarit, J.M.: A tabu search algorithm for a two-dimensional non-guillotine cutting problem. *European Journal of Operational Research* 183(3), 1167–1182 (2007)
11. Egeblad, J., Pisinger, D.: Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers & Operations Research* 36(4), 1026–1049 (2009)
12. George, J.A., Robinson, D.F.: A heuristic for packing boxes into a container. *Computers & Operations Research* 7(3), 147–156 (1980)
13. Loh, T.H., Nee, A.Y.C.: A packing algorithm for hexahedral boxes. In: *Proceedings of the Conference of Industrial Automation*, pp. 115–126 (1992)
14. Bischoff, E., Ratcliff, M.: Issues in the development of approaches to container loading. *Omega* 23(4), 377–390 (1995)
15. Bischoff, E.E., Janetz, F., Ratcliff, M.S.W.: Loading pallets with non-identical items. *European Journal of Operational Research* 84(3), 681–692 (1995)
16. Gehring, H., Bortfeldt, A.: A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research* 4(5-6), 401–418 (1997)
17. Bortfeldt, A., Gehring, H.: Ein tabu search-verfahren für containerbeladeprobleme mit schwach heterogenem kistenvorrat. *OR Spektrum* 20(1), 237–250 (1998)
18. Bortfeldt, A., Gehring, H.: A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research* 131(1), 143–161 (2001)
19. Eley, M.: Solving container loading problems by block arrangement. *European Journal of Operational Research* 141(2), 393–409 (2002)
20. Liang, S., Lee, C., Huang, S.: A hybrid meta-heuristic for the container loading problem. *Communications of the IIMA* 7(4), 73–84 (2007)
21. Yap, C., Lee, L., Majid, Z., Seow, H.: Ant colony optimization for container loading problem. *Journal of Mathematics and Statistics* 8(2), 169–175 (2012)
22. Beasley, J.E.: An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research* 33(1), 49–64 (1985)
23. Beasley, J.E.: OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41(11), 1069–1072 (1990)

Author Index

- Abbasian, Reza 37
Affenzeller, Michael 109
Arbelaez, Alejandro 157
Asta, Shahriar 169
- Beham, Andreas 109
Benlic, Una 61
Bezerra, Leonardo C.T. 85
- Cervante, Liam 25
Chalupa, David 238
Chu, Wenqing 49
Codognet, Philippe 157
Cremene, Marcel 73
Cui, Shanshan 49
- Dréo, Johann 202
Dumitrescu, Dumitru 73
- Eliahou, Shalom 191
Etaner-Uyar, A. Şima 169
- Fonlupt, Cyril 191
Fromentin, Jean 191
- Han, Min 145
Hao, Jin-Kao 61
Hu, Bin 121
- Inführ, Johannes 250
- Kheiri, Ahmed 1
Khoadjia, Mostepha R. 202
- Leal, Jose Eugenio 226
Liu, Chuang 145
Liu, Guang 49
López-Ibáñez, Manuel 85
Lü, Zhipeng 49
- Maenhout, Broos 97
Marinaki, Magdalene 133
Marinakis, Yannis 133
- Marion-Poty, Virginie 191
Mouhoub, Malek 37
- Nagata, Yuichi 179
Nepomuceno, Napoleão V. 262
- Ombuki-Berman, Beatrice 214
Ono, Isao 179
Özcan, Ender 1, 169
- Pallez, Denis 73
Papazek, Petrina 121
Parkes, Andrew J. 169
Pinheiro, Plácido R. 262
Pitzer, Erik 109
- Raidl, Günther R. 121, 250
Rainer-Harbach, Marian 121
Reimann, Marc 226
Robilliard, Denis 191
Rubio-Largo, Álvaro 13
Runka, Andrew 214
- Saraiva, Rommel D. 262
Savéant, Pierre 202
Schoenauer, Marc 202
Shang, Lin 25
Stützle, Thomas 85
Suciu, Mihai 73
- Teytaud, Fabien 191
- Vanhoucke, Mario 97
Vega-Rodríguez, Miguel A. 13
Ventresca, Mario 214
Vidal, Vincent 202
- Xiao, Wei 49
Xue, Bing 25
- Ye, Tao 49
- Zhang, Mengjie 25