Francisco Chicano
Bin Hu
Pablo García-Sánchez (Eds.)

LNCS 9595

Evolutionary Computation in Combinatorial Optimization

16th European Conference, EvoCOP 2016
Porto, Portugal, March 30 – April 1, 2016
Proceedings

Springer

Lecture Notes in Computer Science 9595

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

Editorial Board

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

Francisco Chicano · Bin Hu
Pablo García-Sánchez (Eds.)

Evolutionary Computation in Combinatorial Optimization

16th European Conference, EvoCOP 2016
Porto, Portugal, March 30 – April 1, 2016
Proceedings

Springer

Editors
Francisco Chicano
University of Málaga
Málaga
Spain

Pablo García-Sánchez
University of Granada
Granada
Spain

Bin Hu
Austrian Institute of Technology
Vienna
Austria

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-30697-1 ISBN 978-3-319-30698-8 (eBook)
DOI 10.1007/978-3-319-30698-8

Library of Congress Control Number: 2016932502

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

Preface

Combinatorial optimization is the discipline of decision-making dealing with discrete alternatives. The field is at the interface between discrete mathematics, computing science, operational research, and recently also machine learning, and it includes a diversity of algorithms and hybrid methods. Stochastic local search (metaheuristics), evolutionary, and other nature-inspired algorithms are a family of methods able to provide robust, high-quality solutions to problems of a realistic size in reasonable time. These methods are also relatively simple to design and implement, and offer high flexibility. Many challenging applications in science, industry, and commerce can be formulated as optimization problems. A growing number of them have been successfully solved using the sort of computational methods mentioned, which are the main content of these proceedings.

EvoCOP was held for the first time in 2001, as the first workshop specifically devoted to evolutionary computation in combinatorial optimization. In 2004 it became a conference, and since then it has run annually. This volume contains the proceedings of EvoCOP 2016, the 16th European Conference on Evolutionary Computation in Combinatorial Optimization, which was held in Porto, Portugal, from 30 March to 1 April 2016. EvoCOP is one of the four events of Evostar 2016. The other three are EuroGP (19th European Conference on Genetic Programming), EvoMUSART (5th International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design), and EvoApplications (19th European Conference on the Applications of Evolutionary Computation, formerly known as EvoWorkshops).

Previous EvoCOP proceedings 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, 7832, 8600, 9026). The table on the next page reports the statistics for each conference.

This year, 17 out of 44 papers were accepted after our rigorous double-blind process, resulting in a 38.6 % acceptance rate. We would like to thank the quality and timeliness of our Program Committee members' work, especially since this year's time frame was tighter than usual. Decisions considered both the reviewers, report and evaluation of the program chairs. The 17 accepted papers covered methodology, applications, and theoretical studies. The methods included evolutionary and memetic algorithms, variable neighborhood search, particle swarm optimization, hyperheuristics, matheuristics, and other adaptive approaches. Applications included both traditional domains, such as graph coloring, vehicle routing, the longest common subsequence problem, the quadratic assignment problem, and new(er) domains such as the traveling thief problem, Web service location, and finding short addition chains. The theoretical studies involved fitness landscape analysis, local search and recombination operator analysis, and the big valley search space hypothesis. The consideration of multiple objectives, dynamic, and

EvoCOP	Submitted	Accepted	Acceptance (%)
2016	44	17	38.6
2015	46	19	41.3
2014	42	20	47.6
2013	50	23	46.0
2012	48	22	45.8
2011	42	22	52.4
2010	69	24	34.8
2009	53	21	39.6
2008	69	24	34.8
2007	81	21	25.9
2006	77	24	31.2
2005	66	24	36.4
2004	86	23	26.7
2003	39	19	48.7
2002	32	18	56.3
2001	31	23	74.2

noisy environments was also present in a number of articles. This makes the EvoCOP proceedings an important source for current research trends in combinatorial optimization.

We would like to express our appreciation to the various persons and institutions making this a successful event. First, we thank the local organization team led by Penousal Machado and Ernesto Costa from the University of Coimbra. We extend our acknowledgments to Pablo García-Sánchez from the University of Granada for the excellent website and publicity material. We thank Marc Schoenauer from Inria Paris for his continued assistance in providing MyReview conference management system. Thanks are also due to Jennifer Willies and the Institute for Informatics and Digital Innovation at Edinburgh Napier University, UK, for administrative support and event coordination. Finally, we want to thank the Câmara Municipal do Porto and Turismo do Porto for their support, and the prominent keynote speakers, Richard Forsyth and Kenneth Sorensen.

Special thanks also to Christian Blum, Carlos Cotta, Peter Cowling, Jens Gottlieb, Jin-Kao Hao, Jano van Hemert, Peter Merz, Martin Middendorf, Gabriela Ochoa, and Günther R. Raidl for their hard work and dedication at past editions of EvoCOP, making this one of the reference international events in evolutionary computation and metaheuristics.

March 2016

Francisco Chicano
Bin Hu
Pablo García-Sánchez

Organization

EvoCOP 2016 was organized jointly with EuroGP 2016, EvoMUSART 2016, and EvoApplications 2016.

Organizing Committee

Program Chairs

Francisco Chicano	University of Málaga, Spain
Bin Hu	AIT Austrian Institute of Technology, Austria

Local Organization

Penousal Machado	University of Coimbra, Portugal
Ernesto Costa	University of Coimbra, Portugal

Publicity Chair

Pablo García-Sánchez	University of Granada, Spain

EvoCOP Steering Committee

Christian Blum	Ikerbasque and University of the Basque Country, Spain
Carlos Cotta	University of 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	Hannover University of Applied Sciences and Arts, Germany
Martin Middendorf	University of Leipzig, Germany
Gabriela Ochoa	University of Stirling, UK
Günther Raidl	Vienna University of Technology, Austria

Program Committee

Adnan Acan	Eastern Mediterranean University, Turkey
Enrique Alba	University of Málaga, Spain
Mehmet Emin Aydin	University of Bedfordshire, UK
Thomas Bartz-Beielstein	Cologne University of Applied Sciences, Germany
Matthieu Basseur	University of Angers, France
Maria J. Blesa	Universitat Politècnica de Catalunya, Spain
Christian Blum	Ikerbasque and University of the Basque Country, Spain

Sandy Brownlee	University of Stirling, UK
Pedro Castillo	University of Granada, Spain
Francisco Chicano	University of Málaga, Spain
Carlos Coello Coello	CINVESTAV-IPN, Mexico
Peter Cowling	University of York, UK
Karl Doerner	University of Vienna, Austria
Benjamin Doerr	LIX, Ecole Polytechnique, France
Bernd Freisleben	University of Marburg, Germany
Adrien Goeffon	University of Angers, France
Jens Gottlieb	SAP, Germany
Walter Gutjahr	University of Vienna, Austria
Jin-Kao Hao	University of Angers, France
Emma Hart	Edinburgh Napier University, UK
Richard F. Hartl	University of Vienna, Austria
Geir Hasle	SINTEF Applied Mathematics, Norway
Bin Hu	AIT Austrian Institute of Technology, Austria
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
Frédéric Lardeux	University of Angers, France
Rhyd Lewis	Cardiff University, UK
Arnaud Liefooghe	Université des Sciences et Technologies de Lille, France
José Antonio Lozano	University of the Basque Country, Spain
Gabriel Luque	University of Málaga, Spain
Penousal Machado	University of Coimbra, Portugal
Jorge Maturana	Universidad Austral de Chile, Chile
David Meignan	University of Osnabrück, Germany
Martin Middendorf	University of Leipzig, Germany
Julian Molina	University of Málaga, Spain
Eric Monfroy	University of Nantes, France
Christine L. Mumford	Cardiff University, UK
Nysret Musliu	Vienna University of Technology, Austria
Gabriela Ochoa	University of Stirling, UK
Beatrice Ombuki-Berman	Brock University, Canada
Mario Pavone	University of Catania, Italy
Francisco J.B. Pereira	University of Coimbra, Portugal
Matthias Prandtstetter	AIT Austrian Institute of Technology, Austria
Jakob Puchinger	SystemX-CentraleSupélec, France
Rong Qu	University of Nottingham, UK
Günther Raidl	Vienna University of Technology, Austria
Marcus Randall	Bond University, Australia
Eduardo Rodriguez-Tello	CINVESTAV - Tamaulipas, Mexico
Peter Ross	Edinburgh Napier University, UK
Frédéric Saubion	University of Angers, France

Marc Schoenauer	Inria, France
Patrick Siarry	Université Paris-Est Créteil Val-de-Marne, France
Kevin Sim	Edinburgh Napier University, UK
Jim Smith	University of the West of England, UK
Giovanni Squillero	Politecnico di Torino, Italy
Thomas Stützle	Université Libre de Bruxelles, Belgium
Andrew M. Sutton	University of Postdam, Germany
El-ghazali Talbi	Université des Sciences et Technologies de Lille, France
Renato Tinós	University of Sao Paulo, Brazil
Nadarajen Veerapen	University of Stirling, UK
Sébastien Verel	Université du Littoral Côte d'Opale, France
Takeshi Yamada	NTT Communication Science Laboratories, Japan
Shengxiang Yang	De Montfort University, UK

Contents

A Hybrid Constructive Mat-heuristic Algorithm for the Heterogeneous Vehicle Routing Problem with Simultaneous Pick-up and Delivery

Baris Kececi[1(✉)], Fulya Altiparmak[2], and Imdat Kara[1]

[1] Department of Industrial Engineering, Baskent University, Ankara, Turkey
bkececi@baskent.edu.tr, ikara@baskent.edu.tr
[2] Department of Industrial Engineering, Gazi University, Ankara, Turkey
fulyaal@gazi.edu.tr

Abstract. In this paper, a variant of Vehicle Routing Problem, called Heterogeneous Vehicle Routing Problem with Simultaneous Pick-up and Delivery (HVRPSPD), is considered. The HVRPSPD can be defined as determining the routes and vehicle types on each route in such a way that the pickup and delivery demands of each customer must be performed with same vehicle, while minimizing the total cost. We propose a mathematical model for the problem and some valid inequalities for the model. Since the HVRPSPD is an NP-hard problem, the proposed mathematical model can be used to find the optimal solution for the small-size problems. Therefore we propose a hybrid mat-heuristic approach based on the formulation and Local Search to solve medium and large-size HVRPSPDs. A series of experiments is performed to evaluate the performance of proposed algorithm. Computational results show that hybrid mat-heuristic is computationally efficient to find good quality of initial solutions.

Keywords: Heterogeneous Vehicle Routing Problem · Simultaneous Pick-up and Delivery · Hybrid mat-heuristic · Local search

1 Introduction

The Vehicle Routing Problem (VRP), which has a major importance in the field of transportation, distribution and logistics, was first defined and modelled by Dantzig and Ramser [1] towards the end of 1950s. After the Dantzig and Ramser's study, various models and algorithms have been proposed in the literature to obtain exact and approximate solutions for different types of VRP [2,3]. The interest to the VRP is because it is one of the most important problems in the operational level logistics and also it is in the class of NP-hard problems. The VRP can be defined as the problem of determining the minimum cost routes, in which the vehicles in a fleet should follow, in order to satisfy customer requirements under some operational restrictions. There are various types of the VRP according to the considered operational constraints in the literature.

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 1–17, 2016.
DOI: 10.1007/978-3-319-30698-8_1

One of the variants is the Heterogeneous VRP (HVRP) and the other is the VRP with Simultaneous Pick-up and Delivery (VRPSPD).

In the classical VRP, vehicles in a fleet are considered identical, in other words homogeneous. However, in real life logistics, the vehicles in a fleet may have different properties such as fixed cost (purchasing or rental) of the vehicles, unit variable (transportation) cost between any two customers, vehicle capacities, etc. Besides, according to the customer and/or freight needs, different type of vehicles may be required. That is why to reduce the cost of logistics, the identification of distribution routes as well as the selection of vehicle fleet (i.e. how many vehicle should be bought/rent of each type and which type of vehicle should follow which route) are gaining importance for the companies. This situation particularly requires the consideration of the strategic investment decision in the absence of a current vehicle fleet. There are basically two types of HVRP examined in the literature. The first one, which has unlimited number of vehicles of each type was, first proposed by Golden et al. [4]. In this problem the optimal fleet of vehicles is determined. This problem was originally named in several ways in different studies such as, "The Fleet Size and Mix VRP" by Golden et al. [4]; "The Vehicle Fleet Mix" by Salhi and Rand [5]; "The Fleet Size and Composition VRP" by Gheysens et al. [6]. The second basic type of HVRP was first studied by Taillard [7] and there is limited number of vehicles of each type in a fleet. This case is more realistic and was named in several different ways such as, "The VRP with a Heterogeneous Fleet of Vehicles" by Taillard [7]; "The Heterogeneous Fixed Fleet VRP" by Tarantilis et al. [8]. In addition, it is possible to classify the HVRPs depending on whether the fixed and transportation costs are considered or not and whether the fleet size is limited or not etc. Baldacci et al. [9] give a classification for this problem. We refer the interested readers to the paper of Hoff et al. [10] for an extensive review about this problem and its variants.

Another basic assumption in the classical VRP is that the customers either demand or supply goods. Hence, the vehicles are considered either distribute or collect goods on a route. In the VRPSPD, each customer demands and supplies certain amount of goods at the same time. In case of the customers both demand and supply goods, major economic benefits can be obtained when both activities are performed by a vehicle on the same route rather than by vehicles on different routes. The applications of the VRPSPD can be encountered in the distribution system of grocery store chains, blood banks, etc. Reverse logistics is also another area in which the planning of vehicle routes takes the form of VRPSPD. Companies are facing more often with the management of reverse flow of the products, work in process and/or raw materials. Also there are increasing environmental and social responsibilities and some legal obligations that make the Reverse Logistics Management attractive and mandatory for the companies in addition to its economic return. Particularly on the environmental and economic issues such as collection, disposal and assessment of waste, recycling, reprocessing, re-manufacturing and evaluation of used products; the applications of Reverse Logistics force the companies to use their distribution and logistics

network in a most efficient way. We refer the interested readers to the papers of Berbeglia et al. [11] and Parragh et al. [12] for extensive review about this problem and its variants.

The need for new scientific challenges and an industrial demand for more powerful and versatile routing tools has shifted the focus of VRP research to more complex, general, and larger size variants [10]. Because of the increasing importance of the HVRP and VRPSPD in practical and in scientific researches, in this paper we consider a variant of the VRP called the HVRP with simultaneous pickup and delivery (HVRPSPD). The HVRPSPD includes more real-world aspects of routing problems than the classical VRP by taking into account the heterogeneous vehicle fleet and the simultaneous distribution and collection of goods. Despite the fact that the HVRP and VRPSPD are two important problems in the literature and practice, the HVRPSPD has received little attention from researchers so far. Rieck and Zimmermann [13] address a variant of the VRP faced by less-than-truckload carriers in Europe. The problem includes heterogeneous vehicles, time windows, simultaneous delivery and pick-up at customer locations, and multiple uses of vehicles. They present a vehicle routing model that integrates the real-life VRP and the assignment problem of vehicles to loading bays at the depot. They propose a savings-based solution heuristic combines a multi-start and a local search procedure. Cetin and Gencer [14] consider the VRPSPD with time windows constraints and heterogeneous fleet. They propose a mixed integer programming formulation for the problem based on the model developed by Dethloff [15] for the VRPSPD. Rios-Mercado et al. [16] consider a real-life distribution problem in a company which produces bottled products in Mexico. The problem is minimizing the fixed costs and routing costs while including many complex sub-problems such as; how the trailers should be loaded, which vehicle should pull which trailer, which route should be followed by each vehicle and etc. The problem can be classified as multi-depot, multi-commodity HVRPSPD with time windows. These types of problems are named as Rich VRP that includes many real life features. The problem is modelled as a mixed integer programming formulation and solved by a heuristic algorithm based on GRASP. The performance of algorithm is investigated on the test problems randomly generated with the datums obtained from the company. Furthermore, the solution obtained by the proposed algorithm is compared with the current solution that the company has already been using.

In this paper, we propose a mixed integer programming (MIP) formulation, which is arc-based formulation, for the HVRPSPD. We define some valid inequalities to tighten the MIP formulation in order to increase the solution speed of the formulation. Since the problem is in the class of NP-hard problems, the proposed formulation can be used to obtain optimal solutions for small-sized problems. Hence, we propose a mat-heuristic approach based on the MIP formulation and Local Search (LS) [17] algorithm (called MatH-LS) to solve the medium and large-size HVRPSPDs. We investigate the performance of the MatH-LS on a set of instances derived from the literature and compare it with the proposed arc-based MIP formulation in terms of the solution quality and computation time.

The paper is organized as follows: The problem definition with proposed model and valid inequalities are given in Sect. 2. The detailed description of the MatH-LS algorithm is given in Sect. 3. Section 4 reports the computational results and conclusion follows in Sect. 5.

2 Problem Definition

The HVRPSPD can be defined mathematically as in the following. Let $G = (N, A)$ be a complete directed graph where $N = \{0, ..., n\}$ is the set of nodes and $A = \{(i, j) : i, j \in N, i \neq j\}$ is the set of arcs, respectively. 0 indicates the depot node while the remaining are the customer nodes in N. The fleet is composed by b different types of vehicles, with $B = \{1, ..., b\}$. For each $k \in B$ there are T_k available vehicles, each with capacity Q_k and fixed cost f_k. Each arc $(i, j) \in A$ s associated a non-negative cost $c_{ij} = \theta_k l_{ij}$ where l_{ij} is the distance between the nodes (i, j) with $l_{ij} = l_{ji}$, for each $i, j \in N$ triangular inequality holds (i.e. $l_{ij} + l_{jk} \geq l_{ik}$) and Q_k is the dependent (variable) cost per distance unit of vehicle $k \in B$. Each customer $i \in N$ has delivery (d_i) and pickup (p_i) demands, with $0 \leq d_i, p_i \leq Q_k, \forall k \in B$ and $d_0 = p_0 = 0$. The problem consists in finding the minimum cost feasible routes and determining the type of vehicle on each route such that only one type of vehicle must be used on each route and each customer must be visited by exactly one type of vehicle, each route must begin and end at the depot and the total load on vehicles must not exceed the vehicle capacity.

2.1 Proposed Model

Based on the above definitions, the decision variables of the proposed MIP formulation are given as follows: $x_{ijk} = 1$ iff a vehicle of type k ravels directly from node i to node j; z_{ij} = the total remaining delivery load of vehicle just after it gives the delivery demand of node i, if the vehicle travels directly on arc (i, j), otherwise 0; t_{ij} = the total load picked up by vehicle, if the vehicle travels directly on arc (i, j), after it takes the pickup demand of node i, just after leaving the node i, otherwise 0; y_k = the number of vehicle type k used in the fleet; m = the number of routes in the solution. The proposed arc-based MIP formulation (ABF) is as follows:

$$minimize\ z = \sum_{i \in N} \sum_{j \in N, i \neq j} \sum_{k \in B} c_{ij} x_{ijk} + \sum_{k \in B} f_k y_k \tag{1}$$

subject to;

$$\sum_{j \in N \setminus \{0\}} \sum_{k \in B} x_{0jk} \leq m \tag{2}$$

$$\sum_{i \in N \setminus \{0\}} \sum_{k \in B} x_{i0k} \leq m \tag{3}$$

$$\sum_{i \in N, i \neq j} \sum_{k \in B} x_{ijk} = 1, \forall j \in N \setminus \{0\} \tag{4}$$

$$\sum_{j \in N, i \neq j} x_{ijk} = \sum_{j \in N, i \neq j} x_{jik}, \forall i \in N \setminus \{0\}, \forall k \in B \tag{5}$$

$$z_{ij} + t_{ij} \leq \sum_{k \in B} Q_k x_{ijk}, \forall i, j \in N, i \neq j \tag{6}$$

$$\sum_{j \in N, i \neq j} z_{ji} - \sum_{j \in N, i \neq j} z_{ij} = d_i, \forall i \in N \tag{7}$$

$$\sum_{k \in B} d_j x_{ijk} \leq z_{ij} \leq \sum_{k \in B} (Q_k - d_i) x_{ijk}, \forall i, j \in N, i \neq j \tag{8}$$

$$\sum_{j \in N, i \neq j} t_{ij} - \sum_{j \in N, i \neq j} t_{ji} = p_i, \forall i \in N \setminus \{0\} \tag{9}$$

$$\sum_{k \in B} p_i x_{ijk} \leq t_{ij} \leq \sum_{k \in B} (Q_k - p_j) x_{ijk}, \forall i, j \in N, i \neq j \tag{10}$$

$$z_{i0} = 0, \forall i \in N \setminus \{0\} \tag{11}$$

$$t_{0j} = 0, \forall j \in N \setminus \{0\} \tag{12}$$

$$\sum_{k \in B} y_k \leq m \tag{13}$$

$$y_k \leq T_k, \forall k \in B \tag{14}$$

$$\sum_{j \in N \setminus \{0\}} x_{0jk} = y_k, \forall k \in B \tag{15}$$

$$y_k \geq 0 \text{ and integer}, \forall k \in B \tag{16}$$

$$m \geq 0 \tag{17}$$

$$z_{ij}, t_{ij} \geq 0, \forall i, j \in N \tag{18}$$

$$x_{ijk} \in \{0, 1\}, \forall i, j \in N, \forall k \in B \tag{19}$$

In ABF, the objective function (1) minimizes the total transportation cost and the total vehicle utilization cost. The constraints (2) and (3) satisfy at most m vehicles leave and return back to the depot, respectively. The constraint (4) yields that any node is visited by exactly one type of vehicle and with the constraint (5) it is guaranteed that the same type of vehicle enters and leaves at any node. The constraint (6) prevents the vehicle capacity to be exceeded on any node in a feasible solution. Furthermore, it enforces the auxiliary variables to be zero in case they are not in the solution. The constraint (7) ensures that the auxiliary variables, related with the delivery load, take decreasing values on a feasible vehicle tour and similarly the constraint (9) ensures that the auxiliary variables, related with the pick-up load, take increasing values on a feasible vehicle tour. Equations (7) and (9) avoid the sub-tours together. The constraints (8)

and (10) are bounding constraints of delivery and pickup loads, respectively. They strengthen the model and give tighter formulation. The equalities (11) and (12) initially give zero value to the related variables since a vehicle starts and ends its tour with empty load, respectively. The constraint (13) provides that the total number of vehicles of each type must be equal to at most m in the fleet. The constraint (14) restricts that the number of vehicle of each type in the fleet must be less than or equal to the available number of vehicle of each type. Finally, the constraint (15) ensures that the number of arcs leaving the depot of vehicle type k, should be equal to the number of vehicles of type k in the fleet. The constraints (16), (17), (18) and (19) are the non-negativity and integrality constraints. The ABF has $O\left(n^2\right)$ number of 0–1 integer and continues decision variables, $O\left(b\right)$ number of integer decision variables and $O\left(n^2\right)$ number of constraints.

2.2 Valid Inequalities for the Model

The valid inequalities are the constraints, which are added to the mixed integer mathematical models to tighten their linear programming relaxations. Those inequalities yield all integer feasible solutions; besides, cut some of the relaxed solutions and exclude them from the relaxed solution space. Thus, stronger lower bounds can be obtained by valid inequalities for a problem and this may decrease the solution time of the models. In this study we dwell on three polynomial sized valid inequalities (look at (20) - (24)) to strengthen the ABF.

The first of these is a special case of sub-tour elimination constraints proposed for TSP by Dantzig et al. [18]. In this form of constraints, instead of whole exponential number of sub-tour elimination constraints, only the two-element subsets of constraints are used. The valid inequality (20), which is adapted for HVRPSPD, eliminates only the sub-tours between two customer nodes in any feasible solution.

$$\sum_{k \in B} \left(x_{ijk} + x_{jik}\right) \leq 1, \forall i, j \in N \setminus \{0\}, i < j \tag{20}$$

Another inequality is a covering type inequality, which has been examined in Yaman [19]. In this study the inequalities of the form $\alpha_a a \geq \alpha_0 + \alpha_b b$ are used where α_a, α_b and α_0 are all non-negative values and the Chvatal-Gomory procedure is applied to obtain the valid inequalities. So in our formulation, as described in Yaman [19], we use the inequalities (21),(22) for both delivery and pickup demands, where $Q > 0$. Yaman [19] chooses Q to be Q_1, Q_2,...,Q_b and their greatest common divisor as well, and so we do.

$$\sum_{k \in B} \lceil Q_k / Q \rceil y_k \geq \lceil (\sum_{i \in N} d_i) / Q \rceil \tag{21}$$

$$\sum_{k \in B} \lceil Q_k / Q \rceil y_k \geq \lceil (\sum_{i \in N} p_i) / Q \rceil \tag{22}$$

Let $j \in N \setminus \{0\}$ is any customer visited on a route by the vehicle type $k \in B$. Then the remaining capacity of the vehicle type k should be big enough to get the load remaining after the delivery and pickup operations done for the customer j. To satisfy this relation, the inequality $Q_k - (z_{ij} + t_{ij}) \geq p_j - d_j$ must be yield. In other words the conditional relation of (23) must be satisfied iff $x_{ijk} = 1$ for each customer $j \in N \setminus \{0\}$.

$$Q_k - \sum_{i \in N} (z_{ij} + t_{ij}) \geq p_j - d_j \tag{23}$$

Hence the last inequality, which is a linearisation of the above conditional relation, is valid and can be added to the MIP formulation as shown in (24).

$$\sum_{i \in N} (z_{ij} + t_{ij}) + p_j - d_j \leq \sum_{i \in N} \sum_{k \in B} Q_k x_{ijk}, \forall j \in N \setminus \{0\} \tag{24}$$

3 Proposed Hybrid Mat-heuristic Algorithm

Last few decades have been witnessed the use of model-based heuristics (mat-heuristics) on the solution of combinatorial optimization problems besides the use of heuristic and meta-heuristic. In spite of the development in the computer technologies and in the exact solutions algorithms; however the heuristic and meta-heuristic approaches still keep its importance to find good quality solutions in reasonable time. The existence of powerful software creates new opportunities in the design of heuristic and meta-heuristic algorithms. Hence, a new algorithm class is born, which combines the heuristic and meta-heuristic approaches with mixed integer programming strategies and software infrastructure.

Mathematical model based algorithms, as it is stated in the name, interactively cooperate together with the heuristic (metaheuristic) and mathematical programming techniques. The most important feature of this type of heuristics is the use of information from the mathematical formulation of the problem, within some part of the algorithm [20–24]. Generally the mathematical programming tools are used in some part of the solution procedure. Maniezzo et al. [25] publish a book in consequence with the last workshop about this field of work. The book constitutes of the studies done by the mathematical model based heuristic approaches and their probable usage areas.

In this study we propose a mathematical model-based hybrid heuristic approach to solve the HVRPSPD instances. The heuristic constitutes three phases. The first one is the clustering phase, the second one is the local search phase and the last one is the routing phase. In the clustering phase, the group of customers within each tour is obtained iteratively and the vehicle type on the tour is determined as well. The proposed ABF is used with some modifications in the clustering phase. The main idea in the clustering phase is to solve the relatively and reasonably small part (sub problem) of the original problem at each iteration. The sub problems can be optimally solved by any exact solution approach. At the end of each iteration, according to the exact solution of sub problem,

certain number of decision variables is fixed and a new sub problem is built by adding same number of decision variables. The iterations continue until all the decision variables are fixed. Similar approach is studied by Dondo et al. [26] for the multi-depot HVRP with time windows. The main differences of this study from Dondo et als are the use of mathematical formulation in the clustering phase and the identification of vehicle type on each tour. Consequently the local search phase searches better tours by the help of some neighbourhood search structures. And last the routing phase gives the best order of customers in each tour. Again the proposed ABF but with one vehicle version is used in the routing phase.

3.1 Clustering Phase

In this phase, the cluster of customers on each tour is obtained iteratively and at the end of the iterations the vehicle type on a tour is determined. To be able to build the cluster of customers, one must find m distinct paths at each iteration where each path begins at any node (customer and/or depot) and ends at the depot node. Therefore, the sub problem that has to be solved at each iteration is a Multi-depot Open HVRPSPD (MDOHVRPSPD). The first node visited just after the beginning node on a path is kept as the beginning node of the next iteration. At the end of the iterations, the beginning nodes kept for a path corresponds to the set of customers assigned to the tour.

Modified arc-based formulation (MABF) is used to solve the sub-problem within each iteration. The most significant modification on the formulation is done by rewriting the assignment constraint (2) as *one vehicle must leave each of the m different nodes (either a customer node or a depot node)*, rather than *m vehicles must leave the depot node*. Additionally the following sets and parameters are defined for the MABF:

D is the depots set and N' is the candidate customers set where $D \subset N, N' \subset N$ and $D \cap N' = \phi$. P, R and S sets are defined with the Boolean operations. $P = N' \cup D$, $R = \{0\} \cup N'$ and $S = R \cup D$. The nodes in D have cumulative delivery demand d'_i, and pickup demand p'_i. At the current iteration after the depot set is regenerated with the first visited nodes in the previous iteration; d'_i and p'_i are updated with respect to the preceding nodes. The MABF is as follows with the sets given above and with previously defined parameters and decision variables:

$$minimize\ z' = \sum_{i \in S} \sum_{j \in S, i \neq j} \sum_{k \in B} c_{ij} x_{ijk} + \sum_{k \in B} f_k y_k \tag{25}$$

subject to (13),(14),(16) and;

$$\sum_{j \in R} \sum_{k \in B} x_{ijk} = 1, \forall i \in D \tag{26}$$

$$\sum_{i \in D} \sum_{j \in R, i \neq j} \sum_{k \in B} x_{ijk} + \sum_{j \in N'} \sum_{k \in B} x_{0jk} \leq m \tag{27}$$

$$\sum_{i\in P}\sum_{k\in B} x_{i0k} \leq m \tag{28}$$

$$\sum_{i\in S, i\neq j}\sum_{k\in B} x_{ijk} = 1, \forall j \in N' \tag{29}$$

$$\sum_{j\in R, i\neq j} x_{ijk} = \sum_{j\in S, i\neq j} x_{jik}, \forall i \in N', \forall k \in B \tag{30}$$

$$z_{ij} + t_{ij} \leq \sum_{k\in B} Q_k x_{ijk}, \forall i \in S, \forall j \in R, i \neq j \tag{31}$$

$$\sum_{j\in S, i\neq j} z_{ji} - \sum_{j\in R, i\neq j} z_{ij} = d_i, \forall i \in N' \tag{32}$$

$$\sum_{k\in B} d_j x_{ijk} \leq z_{ij} \leq \sum_{k\in B} (Q_k - d_i) x_{ijk}, \forall i \in S, j \in R, i \neq j, d_i \longleftarrow d'_i \; \forall i \in D \tag{33}$$

$$\sum_{j\in R, i\neq j} t_{ij} - \sum_{j\in S, i\neq j} t_{ji} = p_i, \forall i \in N' \tag{34}$$

$$\sum_{k\in B} p_i x_{ijk} \leq t_{ij} \leq \sum_{k\in B} (Q_k - p_j) x_{ijk}, \forall i \in S, j \in R, i \neq j, p_i \longleftarrow p'_i \; \forall i \in D \tag{35}$$

$$z_{i0} = 0, \forall i \in P \tag{36}$$

$$\sum_{j\in R, i\neq j} t_{ij} = p'_i, \forall i \in \{0\} \cup D \tag{37}$$

$$\sum_{i\in\{0\}\cup D} x_{ijk} = y_k, \forall k \in B \tag{38}$$

$$m \geq 0, z_{ij}, t_{ij} \geq 0, \forall i, j \in S \; and \; x_{ijk} \in \{0, 1\}, \forall i, j \in S, \forall k \in B \tag{39}$$

The significant changes in the MABF with respect to the ABF are done by rewriting the constraints (2) and (12) of ABF. The constraints (26) and (27) of MABF substitute the constraint (2) of ABF; and the constraint (37) of MABF substitutes the constraint (12) of ABF. The changes in the remaining constraints are only on their definition sets; neither on their meanings nor on their functionalities. In MABF, the constraint (26) satisfies only one type of vehicle leaves each depot node in the depots set. According to the constraint (27), the total number of leaving vehicles from depots set D plus the depot $\{0\}$ must not exceed m. In the constraints (33) and (35) the cumulative pickup and delivery demands p'_i and d'_i are used instead of p_i and d_i for the nodes in depots set D. In any iteration, on each path, the total amount of pickup demands through the nodes in the depots set must be known so far. The constraint (12) sets the total pickup demand of the preceding nodes, as the pickup demand of the nodes in the depots set. The MABF allows to trans-pass from the nodes in $\{0\} \cup D$ to any node in R at any iteration. Because the depots set D is empty in the first iteration, to avoid self-pass from the node $\{0\}$ to depot $\{0\}$, the

constraint (40) below must be used only in the first iteration of the algorithm. The constraint (40) is then omitted in the consequent iterations.

$$\sum_{i \in D} \sum_{k \in B} x_{i0k} = 0 \tag{40}$$

The details of the algorithm used in the clustering phase are summarized in the following. Let F_i be the i^{th} cluster of the customers where $i = 1, 2, ..., m$; N^* be the unvisited customers set; N' be the candidate customers set. At the beginning of the iterations, set $m = 1$, $F_1 = \{\}$ and $D = \{0\}$. The customers of the original problem (N) are transferred to N^* $(N^* \longleftarrow N)$. The customers in N^* are ordered farthest to nearest according to their total distance to the nodes $\{0\} \cup D$ and then the first σ customers of N^* is transferred to N'. The MABF is then built and solved optimally by a MIP solver for the current sub-problem. The first β customer(s) visited just after the node(s) in $\{0\} \cup D$ is(are) determined. Each of these customers is then assigned to one F_i for $\forall i = 1, ..., m$. According to the solution of MABF the number of customer clusters, m needs to be increased by one in case of a new path (which starts from the depot node $\{0\}$) is exist. Moreover, these customers substitute the current nodes in D or be added to D in case of a new path is exist in the solution of sub-problem. The nodes in the updated D are copied and kept in order in F_i for $\forall i = 1, 2, ..., m$. The cumulative pickup (p'_i) and delivery (d'_i) demands of the nodes in D are updated. At last, these customers are taken away from the sets of N', N^* and an iteration of the algorithm is completed. In the next iteration, the nodes (with the smallest total distance to the nodes in $\{0\} \cup D$) from the set N^* are transferred to the set N'. The MABF for the next sub-problem is then built and optimally solved. The iterations continue until the set N^* becomes empty. At the end of the iterations, F_i for $\forall i = 1, 2, ..., m$ give the tours for the HVRPSPD. With the last sub-problem solved in the last iteration the vehicle type on each tour $(k_i^*, \forall i = 1, ..., m)$ is already determined.

3.2 Local Search Phase

At the end of the clustering phase the vehicle tours and the vehicle types on each tour are determined. In consequence of the clustering phase, a simple local search is applied to find whether there are better tours in terms of the objective function value. To be able to utilize a local search we use a matrix representation for the solutions of HVRPSPD. In this matrix, each row corresponds to a tour of vehicle where the first element in the row shows the vehicle type and the remaining indicate the customers to be visited sequentially in the tour.

In the local search phase we use three inter-route moving strategies to search the neighbours of a current solution. These are; (1) $shift(1, 0)$: A customer i from route r_1 is transferred to route r_2; (2) $shift(2, 0)$: Two adjacent customers i and j from route r_1 are transferred to route r_2; (3) $k - shift$: A subset of consecutive customers from a route r_1 is transferred to the end of a route r_2. All neighbourhood structures implements the best improvement strategy when

searching a solution space by a moving strategy to select a neighbour of the current solution. Moreover, it accepts only the feasible moves, which do not violate the maximum load constraints.

At each iteration of the local search phase, the algorithm simply chooses one of the three moving strategies randomly. Then if the objective function of the new solution is better than the current one, it accepts the new solution as the current solution. The search phase continues until there is not any improvement in the objective function through the $|N'|$ successive iterations.

3.3 Routing Phase

The routing phase gives the best order of customers within each cluster independent of each other. In this phase each tour is considered as a one vehicle HVRPSPD and the optimal solution is obtained by solving its MIP formulation. The ABF is transformed to a one vehicle HVRPSPD (1HVRPSPD) by setting; $N = \overline{I}, B = \{k^*\}$ and $m = 1$ where $\overline{I}$ is the set of nodes that includes the depot node $\{0\}$ and the customer nodes in the current tour; and k^* is the vehicle type that is assigned to the tour.

The Pseudo Code showing overall steps of the MatH-LS algorithm of all phases is given below.

```
algorithm MatH-LS ()
 initiate (m:=1, F_1:={}, D:={0}, N*:={}, N':={}, N:={}, σ, β);
 read (the problem data);
 set (N* ⟵ N);
 repeat //*Clustering phase*//
  sort (N*, nearest to farthest to the nodes in {0} ∪ D);
  copy (first σ nodes from N* to N');
  set (N* ⟵ N* − N');
  build (the MABF with the parameters of nodes in N');
  solve (the MABF for the sub-problem);
  determine (the first β nodes visited just after
             the nodes in {0} ∪ D);
  if (a new path exists in the solution)
   set (m:=m+1);
  endif
  copy (the determined nodes to DeptoCusFirst{}_i);
  update (F_i ⟵ DeptoCusFirst{}_i for ∀i = 1, ..., m);
  update (d_j := d_j + d'_i; i ∈ {0} ∪ D, j ∈ DeptoCusFirst_i);
  update (p_j := p_j + p'_i; i ∈ {0} ∪ D, j ∈ DeptoCusFirst_i);
  set (D ⟵ DeptoCusFirst);
  set (N' ⟵ N' − DeptoCusFirst);
  set (N* ⟵ N* − DeptoCusFirst);
 until (N* = {});
 do //*Local search phase*//
  set (NList:={shift(1,0), shift(2,0), k − shift});
  choose (a random move ∈ NList);
  find (the best neighbourhood of current solution);
  if (Obj(best neigh.sol.) < Obj(current sol.))
```

```
   set (current sol:=best neigh.sol.);
  endif
 while (# of iters.without improve.in obj.func. < |N'|);
 for i=1 to m //*Routing phase*//
  set (N ⟵ Ī, B ⟵ {k*}, m:=1);
  build (1HVRPSPD for the i^th tour);
  solve (model for the 1HVRPSPD);
  set (the new order of customers in the tour);
 next
 report (the solution and its objective function);
end.
```

4 Computational Results

In order to investigate the performance of MatH-LS, the results of the algorithm are compared with the upper bounds obtained by the MIP formulation on a set of test instances. Since there does not exist any benchmark problems for HVRPSPD, two problem sets for the HVRP are used to generate HVRPSPD test instances. The first HVRP test set was derived by Taillard [7] from the VRP test problems of Golden et al. [4]. This set includes 4 instances with 50 customers, 2 instances with 75 customers and 2 instances with 100 customers. The second HVRP test set was derived by Liu and Shen [27] from the Solomons [28] VRP test problems. This set consists of 18 instances with 100 customers. A HVRPSPD test instances can be easily obtained from a HVRP instance by using a demand separation approach.

In this study, we utilize two demand separation approaches to generate the delivery and pickup demands of customers in each HVRPSPD test instance. The first strategy was proposed by Salhi and Nagy [29]. In the first approach a ratio $r_i = min\{x_i/y_i, y_i/x_i\}$ is calculated and the original demands were split into the pickup and delivery demands according to this ratio. For example, let q_i be the original demand of the customer i. Then the delivery demand is $d_i = r_i q_i$ and the pickup demand $p_i = (1 - r_i)q_i$. We call this type of problems as Type X. Similarly, another problem type that is briefly referred to as Type Y is obtained by shifting each demand of the customer to the next one's demand. In this study, we also proposed another demand separation approach. This approach splits the original demands into the pickup and delivery demands according to the *Golden Ratio*. In mathematics two quantities are in the Golden Ratio if the ratio of the sum of the quantities to the larger quantity is equal to the ratio of the larger quantity to the smaller one. For instance when we divide a line segment $|AB|$ into two parts according to the Golden Ratio, this segment should be divided by a point of C such that the ratio of the bigger part $|CB|$ to the smaller part $|AC|$ is equal to the ratio of the whole line segment $|AB|$ to the bigger part $|CB|$; i.e. $|CB|/|AC| = |AB|/|CB| = \varphi$. φ is an irrational number like π or e and it is equal to $1 + \sqrt{5}/2$ in fractional and is equal to 1.6180339887... in decimal as well. According to these definitions, the original demand q_i of the customer i is

split in to the pickup and delivery demands as $d_i = \lfloor 2q_i/(1+\sqrt{5})\rfloor, p_i = q_i - d_i$ in case i is odd and as $p_i = \lfloor 2q_i/(1+\sqrt{5})\rfloor, d_i = q_i - p_i$ in case i is even. We call this type of problems as Type W and similarly Type Z of problems are generated with shifting each demand of the customer to the next customer as it is explained previously. As a result, 520 ($5 \times 4 \times 26$) small and medium-sized instances are generated by using the first 20, 25, 30, 35 and 40 customers of the problems in each problem set which are generated by using 26 (8+18) original HVRP test problems and 4 different separation procedures (X, Y, W, and Z).

The OPL language and CPLEX 12.6 solver engine are used in coding and solving the MIP formulations, respectively. The proposed MatH-LS algorithm interacting with the CPLEX Concert Technology is coded in C++ programming language in Visual Studio 2010 compiler and all experiments are performed on a computer with Intel Core i5 750 CPU @2,67 @2,67 GHz processor and 2 GB RAM.

The valid inequalities (20), (21), (22) and (24) are adapted for the MABF to strengthen the MABF and used to be able to solve as big sub-problem as possible. Based on our preliminary experiments, we take the cardinality of sub-problem $\sigma = 10$ and keep the first $\beta = 1$ nodes visited just after the node(s) in $\{0\} \cup D$ at each iteration. Each test instance is run 5 times by the proposed algorithm with different random seeds and the computation time of MIP formulation is limited with 2 CPU hours.

In the comparison of MatH-LS and the ABF, following performance measures are used: percentage gap and computation time. The percentage gap is calculated as $100-[(UB-LB)/LB]$ where LB is the LP relaxation bound of the ABF obtained within two hours and UB is the optimal/best solution obtained by the ABF within two hours / the proposed MatH-LS. Table 1 presents the computational results for the MatH-LS and the ABF on HVRPSPD instances obtained by using two demand separation approaches. In the table, the first two columns show the number of customers in a HVRPSPD instance and the demand separation strategy, respectively. Subsequent five columns show the average value of percentage gap, the minimum percentage gap, the maximum percentage gap, the number of problems solved optimally and the average computation time with the MatH-LS, respectively. Finally, the last four columns stand for the average percentage gap of upper bound obtained by ABF, the minimum percentage gap, the maximum percentage gap and the average computation time of ABF. As seen from the Table 1, the proposed algorithm obtains good quality solutions within a very short computational time. The average computation time of the MatH-LS is 48.57 s while the ABF needs 5700 s on average to solve HVRPSPD instances. The average computation time of the MatH-LS changes between 12.33 and 138.17 s. The average percentage gap of the MatH-LS is 22.81 %, while this value is 116.68 % for the MIP formulation. The performance of the ABF quickly degenerates for the instances bigger than 30 customers. The MatH-LS algorithm optimally solves the 78 of 198 instances, which are optimally solved by MIP formulation. Also, it improves upper bounds obtained by MIP formulation for 242 instances. These results show that the MatH-LS is superior to the MIP formulation.

Table 1. Computational results of the proposed MatH-LS algorithm

n	Type	MatH-LS					ABF			
		Gap(%)			Opt	Cpu(s.)	Gap(%)			Cpu(s.)
		Avg.	Min.	Max.			Avg.	Min.	Max.	
20	X	5.38	0.00	27.58	8	16.53	0.28	0.00	3.74	1449.59
	Y	5.49	0.00	21.67	7	13.60	0.28	0.00	5.02	1450.67
	W	4.38	0.00	18.67	12	13.24	0.38	0.00	9.76	749.21
	Z	5.23	0.00	19.45	11	12.33	0.37	0.00	5.79	1203.81
Avg	(Sum)	5.12	0.00	21.84	(38)	13.93	0.33	0.00	6.08	1213.32
25	X	9.35	0.00	29.85	5	18.48	8.70	0.00	167.45	4018.89
	Y	8.87	0.00	27.90	7	17.52	7.63	0.00	159.98	3719.02
	W	7.21	0.00	32.39	7	16.74	5.98	0.00	104.33	3049.56
	Z	9.69	0.00	30.31	7	15.25	7.64	0.00	148.72	3358.76
Avg	(Sum)	8.78	0.00	30.11	(26)	17.00	7.49	0.00	145.12	3536.56
30	X	13.06	0.00	36.22	1	21.95	13.39	0.00	76.44	6529.80
	Y	12.95	0.00	37.77	2	28.12	23.84	0.00	135.80	5945.95
	W	11.11	0.00	28.50	5	22.58	30.33	0.00	267.74	5398.19
	Z	11.56	0.00	41.41	4	18.93	28.03	0.00	267.79	5858.95
Avg	(Sum)	12.17	0.00	35.97	(12)	22.89	23.90	0.00	186.94	5933.22
35	X	26.08	0.06	193.74	1	32.78	35.27	0.00	261.75	7076.00
	Y	16.14	1.12	41.21	0	31.31	37.78	0.00	259.06	7167.35
	W	14.92	0.19	34.35	0	25.42	26.74	0.00	258.78	5721.41
	Z	13.69	0.00	35.03	1	26.26	43.00	0.00	259.24	6727.05
Avg	(Sum)	17.71	0.34	76.08	(2)	28.94	35.70	0.00	259.71	6672.95
40	X	27.66	1.27	184.91	0	41.16	50.27	4.13	404.00	7203.54
	Y	19.28	1.49	50.42	0	46.15	48.84	1.34	417.09	7202.97
	W	15.68	1.23	33.49	0	28.66	44.95	0.00	244.86	6683.07
	Z	11.87	0.52	33.30	0	28.79	42.65	0.00	395.48	5981.91
Avg	(Sum)	18.62	1.13	75.53	(0)	36.19	46.67	1.37	365.36	6767.87
50	X	16.12	5.49	21.00	0	42.90	54.41	17.24	147.34	7203.89
	Y	18.06	5.51	29.12	0	109.39	63.47	6.47	156.65	7202.92
	W	18.82	5.68	24.95	0	83.16	58.68	15.89	156.50	7202.10
	Z	18.31	6.77	27.69	0	62.63	59.97	20.45	148.29	7204.40
Avg	(Sum)	17.83	5.86	25.69	(0)	74.52	59.13	15.01	152.20	7203.33
75	X	36.52	34.59	38.45	0	125.75	400.00[a]	400.00[a]	400.00[a]	7200.00
	Y	35.67	35.23	36.11	0	138.17	400.00[a]	400.00[a]	400.00[a]	7200.00
	W	33.22	33.02	33.42	0	107.77	400.00[a]	400.00[a]	400.00[a]	7200.00
	Z	35.35	32.14	38.55	0	76.05	250.72	101.44	400.00[a]	7200.03
Avg	(Sum)	35.19	33.74	36.63	(0)	111.94	362.68	325.36	400.00[a]	7200.01
100	X	84.35	10.21	316.36	0	91.12	400.00[a]	400.00[a]	400.00[a]	7200.00
	Y	50.12	14.37	125.41	0	95.21	389.13	182.66	400.00[a]	7200.07
	W	63.05	12.21	267.92	0	70.62	412.24	97.33	772.16	7200.04
	Z	70.63	10.73	351.02	0	75.60	388.77	175.36	400.00[a]	7200.01
Avg	(Sum)	67.04	11.88	265.18	(0)	83.14	397.54	213.84	493.04	7200.03
Avg	(Sum)	22.81	6.62	70.88	(78)	48.57	116.68	69.45	251.06	5715.91

[a] This is intentionally given since no UB can be obtained in any instance.

5 Conclusion

In this study, we consider the heterogeneous vehicle routing problem with simultaneous pickup and delivery, HVRPSPD. We propose an arc based MIP formulation and a hybrid mat-heuristic approach to solve the problem. The proposed approach, which constitutes three phases, is based on the MIP formulation and the LS, called MatH-LS. In the MatH-LS, the model based part is used as a constructive structure especially in the clustering phase. The LS part is intended to improve the solution, which is obtained by the model based part, just before the routing phase. With the model based part in the MatH-LS it is easy to construct a feasible solution in terms of the heterogeneous vehicle capacity restrictions without implementing additional feasibility check and repair procedures/mechanisms. We analyse the performance of proposed algorithm in comparison with the MIP formulation on a set of instances adapted from the literature. Computational results indicate that the good quality solutions (23 % in average) are obtained in a reasonable computation time (approximately 49 s). Because of the strict page limitation it is not possible to put further detailed computational results especially in comparison with some other solution approaches. Nevertheless, previous experimentations show that when the simple constructive heuristics used instead of the model based part, the solution quality becomes 41 % in average with the use of Nearest Neighbour Algorithm. Further more, the proposed algorithm can be used to obtain an initial solution for any exact algorithm (i.e. branch and bound, branch and cut, column generation) to shorten the optimization process of exact algorithm.

References

1. Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. Manage. Sci. **6**(1), 80–91 (1959)
2. Toth, P., Vigo, D.: The vehicle routing problem. Society for Industrial and Applied Mathematics (2001)
3. Laporte, G.: Fifty years of vehicle routing. Transp. Sci. **43**(4), 408–416 (2009)
4. Golden, B., Assad, A., Levy, L., Gheysens, F.: The fleet size and mix vehicle routing problem. Comput. Oper. Res. **11**(1), 49–66 (1984)
5. Salhi, S., Rand, G.K.: Incorporating vehicle routing into the vehicle fleet composition problem. Eur. J. Oper. Res. **66**(3), 313–330 (1993)
6. Gheysens, F., Golden, B., Assad, A.: A new heuristic for determining fleet size and composition. In: Gallo, G., Sandi, C. (eds.) Netflow at Pisa. Mathematical Programming Studies, vol. 26, pp. 233–236. Springer, Heidelberg (1986)
7. Taillard, É.D.: A heuristic column generation method for the heterogeneous fleet vrp. Revue française d'automatique, d'informatique et de Recherche Opérationnelle Recherche Opérationnelle **33**(1), 1–14 (1999)
8. Tarantilis, C.D., Kiranoudis, C.T., Vassiliadis, V.S.: A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. Eur. J. Oper. Res. **152**(1), 148–158 (2004)

9. Baldacci, R., Toth, P., Vigo, D.: Exact algorithms for routing problems under vehicle capacity constraints. Ann. Oper. Res. **175**(1), 213–245 (2010)
10. Hoff, A., Andersson, H., Christiansen, M., Hasle, G., Løkketangen, A.: Industrial aspects and literature survey: fleet composition and routing. Comput. Oper. Res. **37**(12), 2041–2061 (2010)
11. Berbeglia, G., Cordeau, J.F., Gribkovskaia, I., Laporte, G.: Static pickup and delivery problems: a classification scheme and survey. Top **15**(1), 1–31 (2007)
12. Parragh, S.N., Doerner, K., Hartl, R.F.: A survey on pickup and delivery models part i: transportation between customers and depot. J. Betriebswirtschaft **58**, 21–51 (2008)
13. Rieck, J., Zimmermann, J.: A hybrid algorithm for vehicle routing of less-than-truckload carriers. In: Sörensen, K., Sevaux, M., Habenicht, W., Geiger, M.J. (eds.) Metaheuristics in the Service Industry. Lecture Notes in Economics and Mathematical Systems, vol. 624, pp. 155–171. Springer, Heidelberg (2009)
14. Çetin, S., Gencer, C.: Heterojen araç filolu zaman pencereli eş zamanlı dağıtım-toplamalı araç rotalama problemleri: Matematiksel model. Int. J. Research Dev. **3**(1), 19–27 (2011)
15. Dethloff, J.: Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up. OR-Spektrum **23**(1), 79–96 (2001)
16. Ríos-Mercado, R.Z., López-Pérez, J.F., Castrillón-Escobar, A.: A GRASP for a multi-depot multi-commodity pickup and delivery problem with time windows and heterogeneous fleet in the bottled beverage industry. In: Pacino, D., Voß, S., Jensen, R.M. (eds.) ICCL 2013. LNCS, vol. 8197, pp. 143–157. Springer, Heidelberg (2013)
17. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations & applications. Elsevier (2004)
18. Dantzig, G., Fulkerson, R., Johnson, S.: Solution of a large-scale traveling-salesman problem. J. Oper. Res. Soc. Am. **2**(4), 393–410 (1954)
19. Yaman, H.: Formulations and valid inequalities for the heterogeneous vehicle routing problem. Math. Program. **106**(2), 365–390 (2006)
20. Boschetti, M.A., Maniezzo, V., Roffilli, M., Bolufé Röhler, A.: Matheuristics: optimization, simulation and control. In: Blesa, M.J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A. (eds.) HM 2009. LNCS, vol. 5818, pp. 171–177. Springer, Heidelberg (2009)
21. Croce, D.F., Salassa, F.: A variable neighborhood search based matheuristic for nurse rostering problems. Ann. Oper. Res. **218**(1), 185–199 (2014)
22. Della Croce, F., Grosso, A., Salassa, F.: A matheuristic approach for the total completion time permutation flow shop problem. Operations Research for Complex Decision Making, pp. 20 (2010)
23. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2005. LNCS, vol. 3562, pp. 41–53. Springer, Heidelberg (2005)
24. Raidl, G.R., Puchinger, J.: Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In: Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M. (eds.) An Emerging Approach to Optimization. Studies in Computational Intelligence, vol. 114, pp. 31–62. Springer, Heidelberg (2008)
25. Maniezzo, V., Stützle, T., Vo, S.: Matheuristics: Hybridizing Metaheuristics and Mathematical Programming. Springer, Heidelberg (2010)
26. Dondo, R., Cerdá, J.: A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. Eur. J. Oper. Res. **176**(3), 1478–1507 (2007)

27. Liu, F.H., Shen, S.Y.: The fleet size and mix vehicle routing problem with time windows. J. Oper. Res. Soc. **50**, 721–732 (1999)
28. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper. Res. **35**(2), 254–265 (1987)
29. Salhi, S., Nagy, G.: A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. J. oper. Res. Soc. **50**, 1034–1042 (1999)

A Property Preserving Method for Extending a Single-Objective Problem Instance to Multiple Objectives with Specific Correlations

Ruby L.V. Moritz[1(✉)], Enrico Reich[2], Matthias Bernt[2], and Martin Middendorf[2]

[1] Intelligent Systems Group, Otto-von-Guericke University Magdeburg, Magdeburg, Germany
ruby.moritz@ovgu.de

[2] Parallel Computing and Complex Systems Group, Institute of Computer Science, University of Leipzig, Leipzig, Germany
enrico.reich@gmx.de, {bernt,middendorf}@informatik.uni-leipzig.de

Abstract. A method is proposed to generate multi-objective optimization problem instances from a corresponding single-objective instance. The user of the method can specify the correlations between the generated the objectives. Different from existing instance generation methods the new method allows to keep certain properties of the original single-objective instance. In particular, we consider optimization problems where the objective is defined by a matrix, e.g., a distance matrix for the Traveling Salesperson problem (TSP) or a flow matrix for the Quadratic Assignment problem. It is shown that the method creates new distance matrices with specific correlations between each other and also have the same average distance and variance of distances as the distance matrix of the original instance. This property is important, e.g., when the influence of correlations between the objectives on the behavior of metaheuristics for the multi-objective TSP are investigated. Some properties of the new method are shown theoretically. In an empirical analysis the new method is compared with instance generation methods from the literature.

Keywords: Multi-objective optimization · Problem instance generation · Traveling salesperson problem

1 Introduction

The empirical analysis of a metaheuristic or other types of algorithms for an optimization problem is usually done on a set of (benchmark) test instances. These have various property values to study how their properties influence the optimization behavior. Considering the *Traveling Salesperson Problem* (TSP) an increased standard deviation of the distances between the cities of a TSP instance increases its difficulty for *Ant Colony Optimization* (ACO) [22].

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 18–33, 2016.
DOI: 10.1007/978-3-319-30698-8_2

Other relevant properties that might influence the difficulty of a TSP instance for an optimization algorithm are the distribution of the distances between the cities, if the cities occur clustered or not, and the size of the convex hull of the cities [16]. The state of such properties might also influence the best choice of parameter values for a metaheuristic (e.g. shown in [19] for ACO).

For applications it is important to have real world test instances or at least test instances that reflect the properties of real world instances. However, for many optimization problems only a relatively small number of real world test instances is available. In those cases it is difficult to find real world instances with specific and varying properties and a method for generating problem instances with such properties and similarities to given real world instances is helpful.

In this paper such a problem instance generation method for *multi-objective optimization problems* (MOOPs) is proposed. An important and characteristic property of MOOPs is if and how the objectives are correlated. Many real world MOOP instances have inter-dependencies between their objectives that influence the characteristics of their fitness landscape [25]. It was shown that correlations between the objectives can influence the correlation between the Pareto optimal solutions [13]. In general, it holds that objectives which are not positively correlated lead to a diverse and large set of Pareto optimal solutions [9]. In contrast, positively correlated objectives decrease the number of Pareto optimal solutions, e.g. for NK-landscapes [23].

It has been demonstrated for various MOOPs that the correlations between objectives can influence the performance of optimization algorithms. If some objectives have a strong positive correlation, a dimensionality reduction can have positive effects on the performance of a metaheuristic [1,5]. To this end, groups of positively correlated objectives can often be aggregated into a single objective [18]. Several studies have demonstrated the influence of the correlation between objectives on the performance of metaheuristics, e.g. [4]. A co-influence of the correlation between objectives, the dimension of the objective space, and the degree of non-linearity on the size of the Pareto set was shown in [24].

Often, a multi-objective problem is an extension of a corresponding single-objective problem. One example is the *multi-objective TSP* (MO-TSP) which is to find a round trip through n given cities that minimizes the traveled distance with respect to multiple $n \times n$ distance matrices. For the MO-TSP it has been demonstrated that correlated objectives influence the optimization behavior of population based ACO (P-ACO) algorithms [17]. Another example is the *multi-objective 0/1 Knapsack problem* which is to find an assignment of a subset of n items to k knapsacks, where each knapsack has a weight capacity limit and the items have knapsack specific weights and profits that are defined by $n \times k$ matrices, such that the total profit is maximized and the weights in the knapsacks satisfy the capacity constraints. The influence of correlated objectives on evolutionary multi-objective algorithms for this problem has been investigated in [6–8]. The performance of MOEA/D is severely degraded by an increase in the number of objectives when they are strongly correlated [7]. Algorithms NSGA-II and SPEA2, on the other hand, perform well on multi-objective problems with strongly correlated objectives [6]. Also, the search behavior of the hypervolume-

based SMS-EMOA algorithm is biased toward the region of the Pareto front for duplicated (i.e. strongly correlated) objectives [8]. The last example is the *Quadratic Assignment Problem* (QAP) where given are n facilities, n locations, a flow matrix $F = [f_{ij}]$ where f_{ij} is the flow from facility i to j, and a distance matrix $D = [d_{ij}]$ where d_{ij} is the distance between locations i and j. The problem is to find an assignment π of the facilities to the locations such that $\sum_{i,j\in[1:n]} f_{ij}d_{\pi(i)\pi(j)}$ is minimized. The *multi-objective QAP* considers multiple flows, where each flow defines one objective. It was shown in [20] that the performance of local search operators (which are a central component of many metaheuristic algorithms) for the multi-objective QAP are strongly influenced by the strength of the correlation between the objectives. ACO algorithms for multi-objective QAPs perform better with 'less aggressive' search strategies (e.g. with iteration-best pheromone update instead of best-so-far pheromone update) when the objectives have a strong positive correlation [14]. On instances with weak or negative correlation between the objectives this does not hold.

As shown by the above examples, various combinatorial MOOP instances are defined mainly by a set of matrices where each objective is represented by one matrix. In the examples there is one distance matrix for each objective of the multi-objective TSP, one flow matrix for each objective of the multi-objective QAP, and one profit matrix for each objective of the multi-objective 0/1 Knapsack problem. In all these cases, the values of the objective functions depend directly on the values of the matrix. Moreover, correlations between the matrices result in a correlation between the respective objectives.

The discussion shows that it would be very helpful for analyzing the optimization behavior of metaheuristics for MOOPs to have the following type of methods for generating MOOP instances. Starting from a given real world single-objective problem instance a method generates new MOOP instances such that:

1. the correlation between the objectives of a MOOP instance can be controlled by the user and
2. some important properties of the single-objective problem instance also hold for a newly generated MOOP instances.

In this paper such an instance generation method for MOOPs where each objective is defined by a matrix is presented. In particular, the method generates new matrices from a given matrix such that the following two properties hold: (i) the correlation between the generated matrices (and the given matrix) can be controlled by the user and (ii) the mean value and the variance of the values of each generated matrix are equal to the corresponding values for the given matrix. Thus, the presented instance generation method is suitable to generate correlated cost matrices for different MOOPs. The new method improves a method that we have presented in [17] which can generate correlated matrices from a given matrix such that property (i) is guaranteed (to a certain extend) whereas property (ii) does not hold.

In the following section related work is described. Our new instance generation method and its properties are described in Sect. 3. Experimental results for the new instance generation method when applied to the TSP are presented in Sect. 4. Conclusions are given in Sect. 5.

2 Related Work

Two common approaches to generate instances of combinatorial MOOPs are to generate (usually uniformly distributed) random cost matrices (e.g. [10]) and to combine single-objective test instances of the same size (as done, e.g. in [9] by combining instances of the TSPLIB [21]). The latter approach creates test instances with a potentially larger practical relevance, but the number of generatable instances is very limited. Both approaches are certainly a useful practice but do not allow to control the correlation between the objectives. The creation of correlated objective functions by using different linear combinations of two random single objective functions using two parameters $\alpha, \beta \in (-1, 1)$ has been suggested in [7,8] and applied to the multi-objective 0/1 Knapsack problem. The practical relevance of this approach is limited due to the use of strict linear (cor)relations. Related problems also occur in multi objectivization approaches which try to avoid local optima by adding objectives to single objective problems such that the global optimum is not affected. Typically subproblems are used as additional objectives, e.g. [11] suggested to use the length of a sub-tour as second objective for the TSP. Also for MOOPs subproblems could define large sets of objectives, but the correlation can not be controlled easily.

The first problem instance generator for MOOPs with correlated objectives was proposed for the multi-objective QAP with k flow matrices that define the different objectives [2,13]. Entries of the flow matrices are defined by a (exponential) function of a random variable. For the first matrix a uniformly random variable X is used and for jth matrix, $j \in [2 : k]$, some entries are generated using a random variable X_j that is correlated with X and the remaining with an independent random variable X'_j. The correlation and the random fraction of the matrix is set with parameters. Several authors have used the QAP generator to study the influence of correlation – mostly for bi-objective problems – on the optimization behavior of metaheuristics and local search operators [4,14,20].

The generation of instances of the multi-objective TSP with correlated objectives was covered by [12]. For each pair of cities (i, j) k distance values $d_h(i, j)$, $h \in [2 : k]$ were created with

$$d_h(i, j) = \alpha \cdot d_{h-1}(i, j) + (1 - \alpha) \cdot rand \tag{1}$$

where the values $d_1(i, j)$ are chosen uniformly at random from $[0, 1]$, $\alpha \in [-1, 1]$ is a "correlation parameter", and $rand$ is a uniform random number from $[0, 1]$. Let us observe here, that (1) has the following potential problems: (i) since the distance values can become > 1 for $\alpha < 0$ even for distance values from $[0, 1]$ for the original matrix it might lead to an uneven influence of different objectives on the behavior of metaheuristics, (ii) the distance values are randomized to a different extent for α and $-\alpha$.

All methods mentioned above generate problem instances that have an inhomogeneous correlation structure, i.e. different strengths of pairwise correlations occur between the objectives. In the method of [2,13] each of the objectives 2 to k has a defined (possibly identical) correlation with the first objective.

But the correlation between pairs of the objectives 2 to k might be different. In the method from [12] each objective h, $h \in [2:k]$ has a defined correlation with objective $h-1$. But there are many different pairwise correlations between objectives i and j with $|i-j| \geq 2$. Also, for each of the problem instances created by the methods of [7] different pairwise correlations occur.

A simple method to create multi-objective TSP test instances with a homogeneous correlation structure was presented in [17]. The method solves some of the possible disadvantages of [12] by using the following equation that differentiates between the cases of positive and negative correlation:

$$d_h(i,j) = \begin{cases} \alpha \cdot d(i,j) + (1-\alpha) \cdot rand & \text{for } 0 \leq \alpha \leq 1 \\ |\alpha| \cdot (1-d(i,j)) + (1-|\alpha|) \cdot rand & \text{for } -1 \leq \alpha < 0. \end{cases} \quad (2)$$

The authors suggested to use normalized real world instances for the original matrix, e.g. from the TSPLIB. The homogeneity is achieved by creating all k objectives on the same original distance matrix which is then discarded. It was shown that the Pearson correlation coefficient of the pairs of matrices depends on the parameter α and the variance of the original distance matrix.

A method to design MOOP instances where the correlation between the objectives is defined by a correlation matrix has been presented in [23]. In particular, NK-landscapes have been investigated, but according to [23] the method can also be applied more generally to other MOOPs. For their empirical investigations the same correlation strength was used for each pair of objectives.

A topic that is related to the generation of MOOP instances is the generation of test instances for dynamic optimization problems. These problems are often single-objective problems, but the objective function changes over time. Here it is interesting to study how the strength of the modification of the objective function, e.g. measured by the correlation between the new function and the old function, influences the optimization behavior of metaheuristics. For the dynamic TSP it has been suggested to create the dynamics by renaming a subset of the cities [15]. The size of the renamed subset influences the extent of the change.

3 Method

In the following we present our method to generate a multi-objective TSP instance from a given distance matrix such that the generated distance matrices are expected to (i) correlated to each other (and to the original matrix) in a user defined way and (ii) have the similar statistical properties as the original matrix, in particular, they have the same expected mean value and variance.

From now, we consider only distance matrices $D = (d_{ij})$ that are symmetric, i.e. $d_{ij} = d_{ji}$ and where all diagonal values are zero, i.e. $d_{ii} = 0$, for $i, j \in [1:n]$. Let D be such a distance matrix. Then, $\bar{d} = 2/(n^2-n) \sum_{i<j} d_{ij}$ is the mean value of all elements in the upper triangular submatrix and $s^2 = 2/(n^2-n-1) \sum_{i \neq j} (d_{ij} - \bar{d})^2$ is the sample variance of all elements in the upper triangular submatrix.

The instance generation method uses matrix D to create k symmetric distance matrices $D_1, \ldots, D_k$ that define the k objectives of a multi-objective TSP instance. Using the parameters $q \in [1:k]$ and $\alpha \in [0,1]$ to determine the correlation structure the method creates a set of distance matrices $\{D_1, \ldots, D_k\}$ such that each matrix D_i with $i \in [1:q]$ is positively correlated with D and each matrix in D_i with $i \in [q+1:k]$ is negatively correlated with D. For $D^+ \in \{D_1, \ldots, D_q\}$ or $D^- \in \{D_{q+1}, \ldots, D_k\}$ the method works as follows:

Step 1: Two random symmetric distance matrices R_1 and R_2 are created by randomly sampling with uniform probability and with replacement from the elements within D. In order to create a symmetric matrix with all diagonal values being zero only values in the corresponding upper triangular matrices are considered and the values in the other half are set accordingly.

Step 2: Matrix D^+ (respectively D^-) is created by:

$$D^+ = \alpha \quad D + (1-\alpha)R_1 + \sqrt{2\alpha(1-\alpha)}(R_2 - \overline{d}) \tag{3}$$

$$D^- = \alpha(2\overline{d} - D) + (1-\alpha)R_1 + \sqrt{2\alpha(1-\alpha)}(R_2 - \overline{d}). \tag{4}$$

Note that different randomized matrices R_1, R_2 need to be used for each D_i, $i \in [1:k]$.

Similar to the methods proposed in [12,17], see also (1) and (2), the first (resp. second) summand in (3) and (4) represent the non-random influence (resp. the random influence) which is controlled by α. The main differences are (i) that the random influence by R_1 is realized via sampling from D (which has the effect that the mean is preserved) and (ii) that a third summand is added (which has the effect that the variance is preserved). Furthermore, the mean $\overline{d}$ of D is used for determining the influencing of the non-random component in (4).

Observe, that the newly generated matrices can contain negative distance values. Therefore, we propose to add the absolute value of the smallest negative value that is contained in the generated matrices (let this value be d_{min}) to all (nondiagonal) values in all distance matrices. Adding d_{min} does not change the correlation and variance and it increases the mean value for all matrices by exactly d_{min}. Hence after such an operation, all matrices are still expected to share the same variance and mean value. In case zero values are undesired a larger constant value can be added, e.g. the sum d_{min} and the minimum of the original matrix which would yield a matrix with the same minimum.

For the following theoretical results we specify the method in terms of random variables. Analogous to the creation of the matrices a set of random variables $\{X_1, \ldots, X_k\}$ can be created such that each $X^+ \in \{X_1, \ldots, X_q\}$ is positively correlated with X and each $X^- \in \{X_{q+1}, \ldots, X_k\}$ is negatively correlated with X. For a given random variable X and independent random variables Z_1, Z_2 that have the same distribution as X we define

$$X^+ = \alpha \quad X + (1-\alpha)Z_1 + \sqrt{2\alpha(1-\alpha)}(Z_2 - E[X]) \tag{5}$$

$$X^- = \alpha(2E[X] - X) + (1-\alpha)Z_1 + \sqrt{2\alpha(1-\alpha)}(Z_2 - E[X]). \tag{6}$$

The relation to our method is as follows. The distance values in the upper triangular submatrix of D can be considered to represent the distribution of a random variable X with $E[X] = \bar{d}$ and $V[X] = s^2$. By sampling R_1 and R_2 from D with replacement the corresponding random variables Z_1, Z_2 are independent to X and from the same distribution as X. Correlation between two random variables X and Y is measured in this paper by the correlation coefficient $\rho(X,Y) = \sigma(X,Y)/\sqrt{V[X]V[Y]}$ where $\sigma(X,Y) = E[XY] - E[X]E[Y]$ is the covariance. The following theorem shows how X^+ and X^- are related to X.

Theorem 1. *For random variables X^+ and X^- as given in (5) and (6), respectively, it holds that: (i) $E[X] = E[X^+] = E[X^-]$, (ii) $V[X] = V[X^+] = V[X^-]$, and (iii) $\rho(X^+, X) = \alpha$ and $\rho(X^-, X) = -\alpha$.*

Proof. By definition $E[Z_i] = E[X]$ and $V[Z_i] = V[X]$, $i \in \{1, 2\}$. Due to space limitations the proof is shown only for $\rho(X^+, X)$. For ease of readability set $\beta := \sqrt{2\alpha(1-\alpha)}$. Since Z_1, Z_2, and X are independent random variables it holds that

$$\begin{aligned}
\rho(X, X^-) &= \frac{E[XX^-] - E[X]E[X^-]}{\sqrt{V[X]V[X^-]}} = \frac{E[XX^-] - E[X]^2}{E[X^2] - E[X]^2} \\
&= \frac{E\left[X\left(\alpha(2E[X] - X) + (1-\alpha)Z_1 + \beta(Z_2 - E[X])\right)\right] - E[X]^2}{X^2 - E[X]^2} \\
&= \frac{\alpha(2E[X]^2 - E[X^2]) + (1-\alpha)E[X]^2 - E[X]^2}{E[X^2] - E[X]^2} = -\alpha.
\end{aligned}$$

The theorem shows that the generated distance matrices are expected to (i) maintain basic characteristics (mean and variance) of a specific initial single objective benchmark instance and (ii) have all the same strength of correlation (potentially with different sign) to the given matrix. Mean and variance of a distance matrix are certainly not the only properties that characterize a TSP instance, however, they are basic properties that are well defined, easily computable, and (potentially) have a strong influence on the behavior of optimization algorithms. The correlation between the generated matrices is covered by the following theorem.

Theorem 2. *For random variables X^+, Y^+ and X^-, Y^- that are generated according to (5) and (6), respectively, it holds that: (i) $\rho(X^+, Y^+) = \rho(X^-, Y^-) = \alpha^2$ and (ii) $\rho(X^+, X^-) = \rho(X^-, X^+) = -\alpha^2$.*

Proof. Note, that for each random variable different random variables Z_i, $i \in \mathbb{N}^+$ are used. These do not appear explicitly in the proof since $E[Z_i] = E[X]$. Because $E[Z_i - E[X]] = 0$ we omit all terms involving the last summands of (5) and (6), respectively. For readability we abbreviate $2E[X] - X$ with $\hat{X}$. Due to space limitations cases $\rho(X^+, Y^+)$ and $\rho(X^+, Y^+)$ are omitted. The following holds

$$\begin{aligned}\rho(X^+, X^-) &= \frac{E[X^+X^-] - E[X^+]E[X^-]}{\sqrt{V[X^+]V[X^-]}} = \frac{E[X^+X^-] - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^2 E[X\hat{X}] + \alpha(1-\alpha)E[X](E[X] + E[\hat{X}]) + (1-\alpha)^2 E[X]^2 - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^2(2E[X]^2 - E[X^2]) + 2\alpha(1-\alpha)E[X]^2 + (1-\alpha)^2 E[X]^2 - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^2(E[X]^2 - E[X^2])}{E[X^2] - E[X]^2} = -\alpha^2.\end{aligned}$$

The theorem shows that for the partition $\{X_1, \ldots, X_q\}$ and $\{X_{q+1}, \ldots, X_k\}$ it holds that two random variables from the same set are positively correlated and two random variables from different sets are negatively correlated. By choosing $q = k$ the special case of homogeneous correlations, which was advocated in [17], is covered. Clearly, also the iterated application of the procedure as suggested in [12] is possible. The corresponding correlations are as follows.

Theorem 3. *Let $X^{+0} := X$ and X^{+i} be the result of applying (5) on X^{+i-1}. Then $\rho(X, X^{+i}) = \alpha^i$ holds.*

Proof. Theorem 1 implies $E[X^{+i}] = E[X]$ and $V[X^{+i}] = V[X]$. We set $\gamma_i = (1-\alpha)Z_{2i-1} + \sqrt{2\alpha(1-\alpha)}(Z_{2i} - E[X])$. Then

$$\begin{aligned}\rho(X, X^i) &= \frac{E[XX^i] - E[X]E[X^i]}{\sqrt{V[X]V[X^i]}} = \frac{E[XX^i] - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{E\left[X\left(\alpha(\ldots\alpha(\alpha X + \gamma_i) + \gamma_{i-1})\ldots + \gamma_1\right)\right] - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{E\left[X(\alpha^i X + \sum_{k=1}^{i}\alpha^{k-1}\gamma_k)\right] - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^i E[X^2] + E[X\sum_{k=1}^{i}\alpha^{k-1}\gamma_k] - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^i E[X^2] + \sum_{k=1}^{i}\alpha^{k-1}E[X\gamma_k] - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^i E[X^2] + \sum_{k=1}^{i}\alpha^{k-1}(1-\alpha)E[XZ_{2k-1}] - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^i E[X^2] + (1-\alpha^i)E[X]^2 - E[X]^2}{E[X^2] - E[X]^2} \\ &= \frac{\alpha^i E[X^2] - \alpha^i E[X]^2}{E[X^2] - E[X]^2} = \alpha^i.\end{aligned}$$

Theorem 3 also holds analogously if (6) is applied in some or all recursive steps (instead of (5)), however, with a sign switch for each such step.

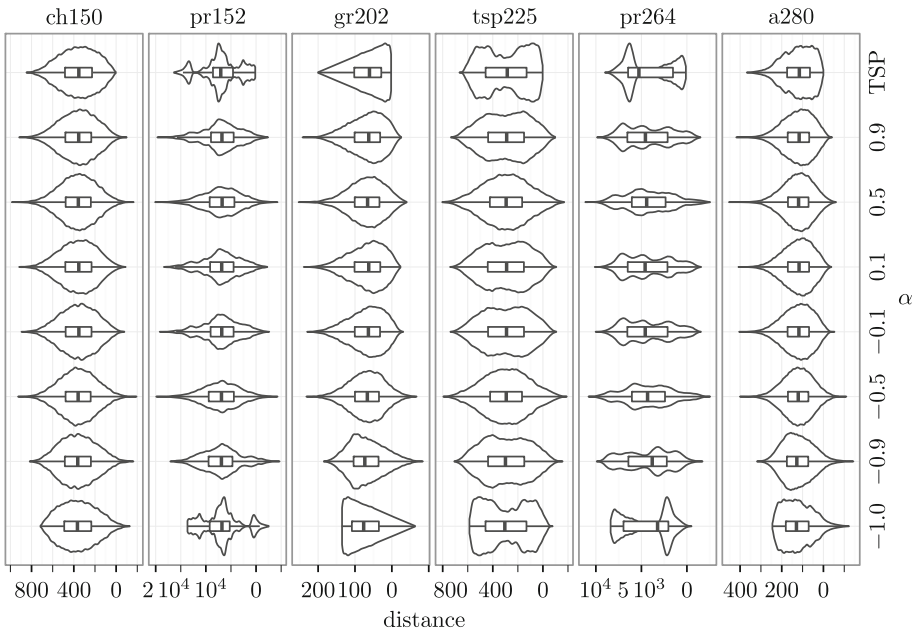


Fig. 1. Violin plot and boxplot of the distance values of six TSPLIB instances (top row) and of the distance matrices that are generated with the new method from these instances with different $\alpha \in \{0.1, 0.5, 0.9\}$ using (3) and (4), the latter indicated by negative values of α.

4 Empirical Analysis

To visualize and complement our theoretical results we performed a series of experiments. For the experiments we used all 85 benchmark instances from the TSPLIB that have at most 7400 cities. A more detailed analysis has been done for the following six of these instances which have been chosen to represent different types of distributions of the distance values that can be found within the instances in the TSPLIB (see topmost row in Fig. 1):

(i) Instances gr202 and a280 represent a negatively skewed distance distribution. For gr202 the number of city pairs decreases almost linearly with an increasing distances, whereas, for a280 this relation is more irregular.
(ii) Instance ch150 represents a distance distribution with a single peak that is similar to a normal distribution (it is only slightly negatively skewed).
(iii) Instances pr152, tsp225, and pr264 represent distance distributions that have more than one pronounced peak. Instance pr264 and pr152 have a non-symmetric distribution with two, respectively three, peaks of different height. Instance tsp225 has two peaks but is more symmetrical and the peaks are less pronounced.

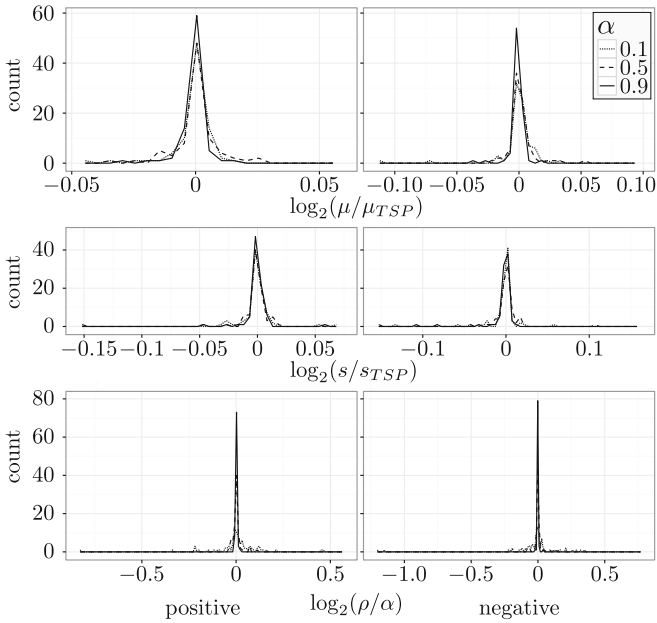


Fig. 2. Distribution of relative mean value (top) and relative standard deviation (middle) of generated matrix with respect to the corresponding value of the original TSPLIB instance over all 85 test instances; also shown is the distribution of the correlation between the generated matrix with the original TSPLIB instance divided by α (bottom for all 85 test instances; shown are the results for the new method; shown are the values for positively (left column) and negatively (right column); $\alpha \in \{0.1, 0.5, 0.9\}$.

Note that for the results that involve a comparison with the results of the methods of [12,17], i.e. (1) and (2), the distance matrices have been normalized.

The result of applying the instance generation method on the selected 85 TSPLIB instances is shown in Fig. 2. For each of the TSPLIB instances and combinations of $\alpha \in \{0.1, 0.5, 0.9\}$ and $q \in \{0, 1\}$ one new matrix was generated, i.e. $k = 1$. The shown distributions of the relative mean values and the relative standard deviations shows that the generated matrices have (with small random fluctuations) the same mean value and the same standard deviation as the corresponding original TSPLIB instances. Note, that the scale of the horizontal axis is logarithmic to base 2 such that 0 corresponds to the fact that the values of the new matrix and the old matrix are equal. The figure also shows that the correlation between the new matrix and the original TSPLIB matrix equals α (with minor random influence). This confirms our theoretical results for the mean, variance, and correlation.

Figure 3 shows the corresponding results when the method of [17], i.e. (2) is applied. The results show a potential disadvantage of this method. The distribution of the distance values in the matrices that is generated with this method is

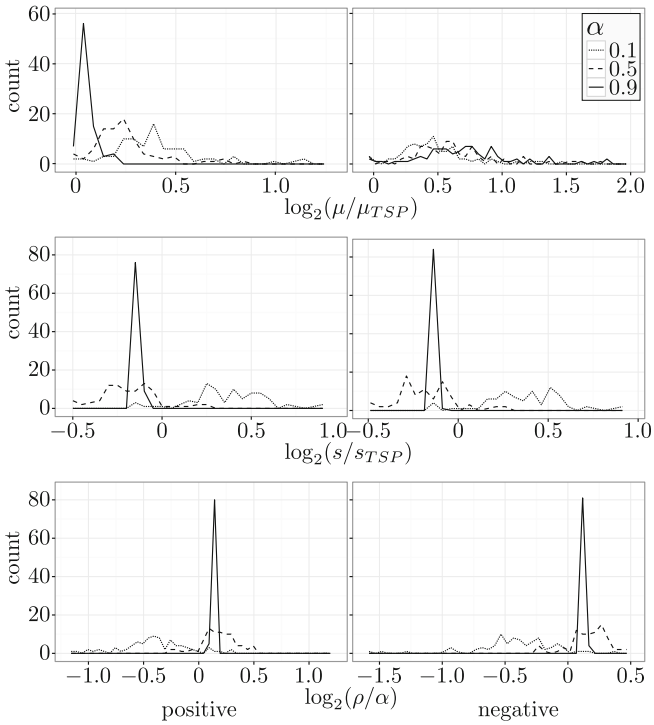


Fig. 3. Distribution of relative mean value (top) and relative standard deviation (middle) relative correlation (bottom) for the 85 test instances from the TSPLIB; shown are the results for the method of [12]; see caption of Fig. 2.

very different compared to the distribution of the values of the original matrices. Their mean values for smaller α approach 0.5 which is the expected value of the random part from (2). Thus, the lower the correlation parameter α is set, the higher the similarity of the resulting distribution to a uniform distribution from 0 to 1. Since most TSPLIB instances have a normalized mean below 0.5 this behavior results in an increase of the mean value for positive correlations and a decrease for negative correlations.

Equation (2) has a notable influence on the standard deviation of the distances in the generated matrices which depends on the values chosen for α. The standard deviation is on average lower for $\alpha \in \{0.5, 0.9\}$ and higher for $\alpha = 0.1$ compared to the original TSPLIB matrix. Moreover, the resulting correlations are virtually never equal to α and they vary strongly, in particular for smaller α. This has been predicted by [17] where it was shown that the observed correlation depends on the variance of the original TSPLIB instance.

As shown, our instance generation method is able to maintain the mean value and variance of the distance distribution of the original matrix. But clearly, other statistical measures might not be preserved by the method. For instance, the

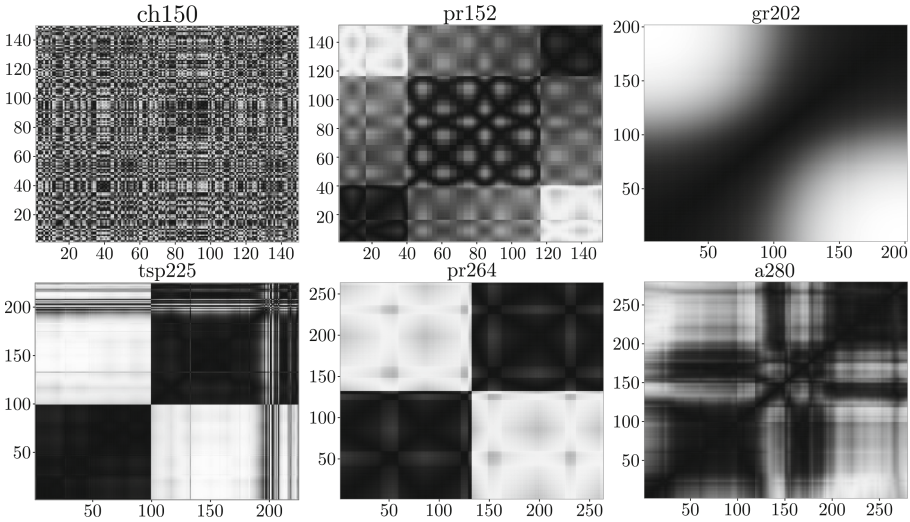


Fig. 4. Correlation $\rho(i, j)$ between all pairs of rows i and j of six TSPLIB distance matrices ; black: $\rho = 1$, white: $\rho = -1$.

median value of the distance distribution might change in the newly generated matrices. This is illustrated in the boxplots in Fig. 1 and can be seen in particular for the asymmetrically clustered instance pr264. With correlation values closer to zero, bulges in the distribution are flattened out. Obviously, this is an effect of the increased random influence on the values in the newly generated matrices. Note, that our instance generation method produces negative distance values, particularly for negative correlations. Since negative distance or cost values could be unrealistic (or impossible) in some application we suggest to deal with them in such cases by a post processing step as described in Sect. 3.

In the following we consider the correlations between the different rows of the distance matrix. More exactly, for an $n \times n$ matrix $M = [m_{ij}]$ the correlation between rows i and j - denoted by $\rho(i, j)$ - is computed over all pairs of values (m_{ih}, m_{jh}) for all $h \in [1 : n] \backslash \{i, j\}$. Note, that the values in row i of the distance matrix are the distances from city i to all other cities. Hence, for a Euclidean TSP instance two cities which are very close to each other have similar distances to all other cities and thus have a high positive correlation between their corresponding rows in the distance matrix. Figure 4 shows the correlations $\rho(i, j)$ for all pairs of rows i and j for each of the six test matrices. Off course, care has to be taken when interpreting the figure because a renumbering of the cities would lead to a reordering of the rows and columns. However, the different characteristics of the six TSPLIB instances can be seen. Instances tsp225, pr264, and pr152 show a clustering of the cities into two (tsp225, pr264) or three (pr152) clusters. Instance gr202 (and to a much lesser extend also instance a280) shows a gradient away from the antidiagonal from positive to negative correlations. Only instance ch150 does not show a clear structure within the correlation plot.

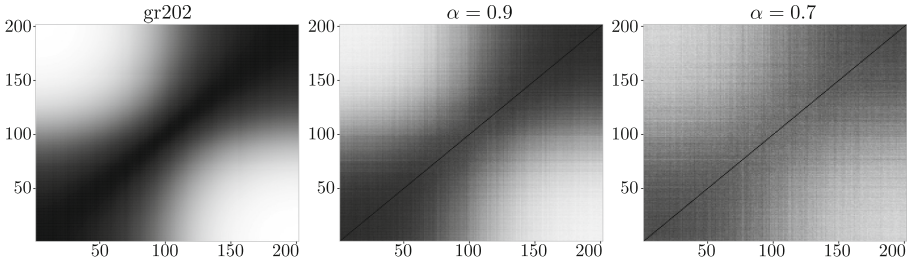


Fig. 5. Correlation $\rho(i, j)$ for all pairs of rows i and j: gr202 (left) and matrices generated from gr202 (middle: $\alpha = 0.9$, right: $\alpha = 0.7$); black: $\rho = 1$, white: $\rho = -1$.

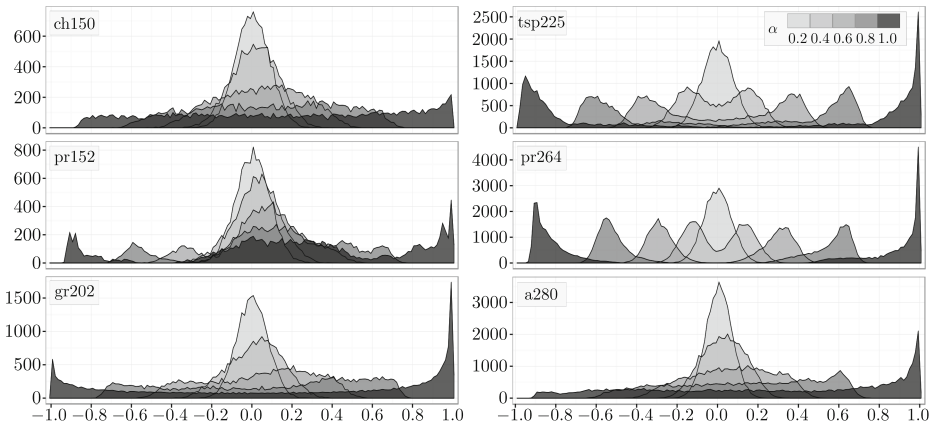


Fig. 6. Distribution of correlations $\rho(i, j)$ between each two rows i and j of matrices generated with $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ for six instances of the TSPLIB; $\alpha = 1.0$ corresponds to the original TSPLIB instance.

In Fig. 5 the correlations between rows in the original TSPLIB matrix and the matrices generated with $\alpha \in \{0.7, 0.9\}$ are shown for gr202. Only results for positively correlated matrices are shown because the results for negative correlation are similar. The general structure of the original matrix is still visible in the generated matrices. Clearly, the random influence introduces considerable noise and reduces the correlation between the rows. This effect is stronger for smaller α.

Indeed, in Fig. 6 it can be seen that the row correlations approach zero for increased random influence, i.e. smaller α. For $0 < \alpha < 1$ the distribution of the row correlations is a mixture of the original TSPLIB instance distribution and the distribution of correlation coefficients of a random sample with correlation zero. The latter distribution was described by Fisher in [3].

Our instance generation method can also provide problem instances where the objectives exhibit different correlations. This can be achieved, for example, by a recursive application of the method. Clearly, each recursive application decreases

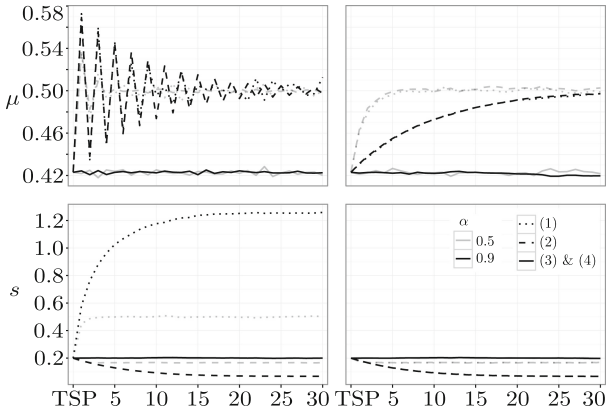


Fig. 7. Mean value and standard deviation (SD) of the distance matrices generated by up to 30 recursive applications of (1), (2), (3) and (4), start matrix is the normalized distance matrix of TSPLIB instance ch150; $\alpha \in \{0.5, 0.9\}$; left: positive correlation; right: negative correlation

the correlation to the original matrix. Other instance generation methods also provide such an option [12,17]. However, a potential disadvantage of these methods is that they change the mean and standard deviation of the generated matrices and therefore important characteristics of the corresponding optimization problems. This effect becomes stronger the more iterations are applied, such that the derived matrices are very different from the original benchmark matrices. This is shown in Fig. 7 where all three methods (i.e. the method from [12] using (1), the method from [17] using (2), and the newly proposed method using (3) and (4)) are compared. It can be seen that for the methods from [12,17] the mean value approaches 0.5 and either increases the standard deviation vastly [12] or reduces it [17]. When we compare the results for positive and negative correlations the compared methods behave asymmetrically. In contrast our method maintains the mean value and standard deviation of the original matrix for positively as well as for negatively for correlated matrices (up to small random influences).

5 Conclusion

We have proposed a new method to generate multi-objective optimization problem instances from a corresponding single-objective instance. The method provides the option to choose specific correlations between the objectives, while maintaining important characteristics of the original single-objective instance. In particular, the proposed method can be applied to optimization problems were the objective is defined by a matrix. Examples are the Traveling Salesperson problem (TSP) which uses a distance matrix and the Quadratic Assignment problem which uses a flow matrix. It was shown that the new method provides the option to choose specific correlations between the generated objectives, while

maintaining the mean value and variance of the given single-objective problem instance. This is not guaranteed by existing instance generation methods that have been proposed in the literature. However, maintaining mean value and variance can be important since these characteristics been shown to be of great influence on the difficulty of TSP instances [22]. Our method was analyzed theoretically and it was experimentally compared for the TSP to other instance generation methods from the literature.

It should be noted the application of the proposed instance generation method to problems other than the TSP but where each objective is controlled by a matrix, e.g. the QAP and the multi objective 0/1 Knapsack problem, requires some minor problem specific modifications. For example, the independent generation of both triangular matrices for the flow matrices of the QAP and the generation of all element of the profit matrices of the Knapsack problem.

References

1. Brockhoff, D., Saxena, D., Deb, K., Zitzler, E.: On handling a large number of objectives a posteriori and during optimization. In: Knowles, J., Corne, D., Deb, K., Chair, D. (eds.) Multiobjective Problem Solving from Nature. Natural Computing Series, pp. 377–403 (2008)
2. Corne, D.W., Knowles, J.D.: Techniques for highly multiobjective optimisation: some nondominated points are better than others. In: Genetic and Evolutionary Computation Conference, pp. 773–780 (2007)
3. Fisher, R.A.: Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. Biometrika **10**, 507–521 (1915)
4. Garrett, D., Dasgupta, D., Vannucci, J., Simien, J.: Applying hybrid multiobjective evolutionary algorithms to the sailor assignment problem. In: Jain, L.C., Palade, V., Srinivasan, D. (eds.) Advances in Evolutionary Computing for System Design. SCI, vol. 66, pp. 269–301. Springer, Heidelberg (2007)
5. Goel, T., Vaidyanathan, R., Haftka, R.T., Shyy, W., Queipo, N.V., Tucker, K.: Response surface approximation of Pareto optimal front in multi-objective optimization. Comput. Methods Appl. Mech. Eng. **196**(4), 879–893 (2007)
6. Ishibuchi, H., Akedo, N., Nojima, Y.: A study on the specification of a scalarizing function in MOEA/D for many-objective Knapsack problems. In: Nicosia, G., Pardalos, P. (eds.) LION 7. LNCS, vol. 7997, pp. 231–246. Springer, Heidelberg (2013)
7. Ishibuchi, H., Akedo, N., Ohyanagi, H., Nojima, Y.: Behavior of EMO algorithms on many-objective optimization problems with correlated objectives. In: 2011 IEEE Congress on Evolutionary Computation (CEC), pp. 1465–1472 (2011)
8. Ishibuchi, H., Yamane, M., Nojima, Y.: Effects of duplicated objectives in many-objective optimization problems on the search behavior of hypervolume-based evolutionary algorithms. In: 2013 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM), pp. 25–32 (2013)
9. Jaszkiewicz, A.: Genetic local search for multi-objective combinatorial optimization. Eur. J. Oper. Res. **137**(1), 50–71 (2002)
10. Jozefowiez, N., Glover, F., Laguna, M.: Multi-objective meta-heuristics for the traveling salesman problem with profits. J. Math. Modell. Algor. **7**(2), 177–195 (2008)

11. Knowles, J.D., Watson, R.A., Corne, D.W.: Reducing local optima in single-objective problems by multi-objectivization. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 268–282. Springer, Heidelberg (2001)
12. Knowles, J.D., Corne, D.W.: Instance generators and test suites for the multiobjective quadratic assignment problem. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 295–310. Springer, Heidelberg (2003)
13. Knowles, J.D., Corne, D.: Towards landscape analyses to inform the design of hybrid local search for the multiobjective quadratic assignment problem. HIS **87**, 271–279 (2002)
14. López-Ibáñez, M., Paquete, L., Stützle, T.: On the design of ACO for the biobjective quadratic assignment problem. In: Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T. (eds.) ANTS 2004. LNCS, vol. 3172, pp. 214–225. Springer, Heidelberg (2004)
15. Mavrovouniotis, M., Yang, S., Yao, X.: A benchmark generator for dynamic permutation-encoded problems. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part II. LNCS, vol. 7492, pp. 508–517. Springer, Heidelberg (2012)
16. Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., Neumann, F.: A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. Ann. Math. Artif. Intell. **69**(2), 151–182 (2013)
17. Moritz, R.L.V., Reich, E., Bernt, M., Middendorf, M.: The influence of correlated objectives on different types of P-ACO algorithms. In: Blum, C., Ochoa, G. (eds.) EvoCOP 2014. LNCS, vol. 8600, pp. 230–241. Springer, Heidelberg (2014)
18. Murata, T., Taki, A.: Examination of the performance of objective reduction using correlation-based weighted-sum for many objective knapsack problems. In: 10th International Conference on Hybrid Intelligent Systems (HIS), pp. 175–180 (2010)
19. Nallaperuma, S., Wagner, M., Neumann, F.: Analyzing the effects of instance features and algorithm parameters for max-min ant system and the traveling salesperson problem. Frontiers in Robotics and AI **2**(18) (2015)
20. Paquete, L., Stützle, T.: A study of stochastic local search algorithms for the biobjective QAP with correlated flow matrices. Eur. J. Oper. Res. **169**(3), 943–959 (2006)
21. Reinelt, G.: TSPLIB-A traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
22. Ridge, E., Kudenko, D.: Determining whether a problem characteristic affects heuristic performance. In: Cotta, C., van Hemert, J. (eds.) Recent Advances in Evol. Comp. SCI, vol. 153, pp. 21–35. Springer, Heidelberg (2008)
23. Verel, S., Liefooghe, A., Jourdan, L., Dhaenens, C.: Analyzing the effect of objective correlation on the efficient set of MNK-landscapes. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 116–130. Springer, Heidelberg (2011)
24. Verel, S., Liefooghe, A., Jourdan, L., Dhaenens, C.: Pareto local optima of multiobjective NK-landscapes with correlated objectives. In: Merz, P., Hao, J.-K. (eds.) EvoCOP 2011. LNCS, vol. 6622, pp. 226–237. Springer, Heidelberg (2011)
25. Xu, Y., Qu, R., Li, R.: A simulated annealing based genetic local search algorithm for multi-objective multicast routing problems. Ann. Oper. Res. **206**(1), 527–555 (2013)

An Evolutionary Approach to the Full Optimization of the Traveling Thief Problem

Nuno Lourenço[1(✉)], Francisco B. Pereira[1,2], and Ernesto Costa[1]

[1] Department of Informatics Engineering, CISUC, University of Coimbra, Polo II - Pinhal de Marrocos, 3030 Coimbra, Portugal
{naml,xico,ernesto}@dei.uc.pt
[2] Polytechnic Institute of Coimbra, Quinta da Nora, 3030-199 Coimbra, Portugal

Abstract. Real-World problems usually consist of several different small sub-problems interacting with each other. These interactions promote a relation of interdependence, where the quality of a solution to one sub-problem influences the quality of another partial solution. The Traveling Thief Problem (TTP) is a recent benchmark that results from the combination of the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP). Thus far, existing approaches solve the TTP by fixing one of the components, usually the TSP, and then tackling the KP. We follow in a different direction and propose an Evolutionary Algorithm that addresses both sub-problems at the same time. Experimental results show that solving the TTP as whole creates conditions for discovering solutions with enhanced quality, and that fixing one of the components might compromise the overall results.

Keywords: Evolutionary algorithms · Combinatorial problems · Traveling thief problem

1 Introduction

Heuristic problem solving, e.g., based on Evolutionary Algorithms (EA), is a successful approach for solving problems for which an analytical solution does not exist, or, when it does, it is computationally intractable. To assess the performance of heuristic-based EAs, researchers usually rely on benchmark problems combined with a sound statistical analysis. Choosing good benchmarks is thus critical, and, over time, discussing which ones should be used has gained relevance in the EA community [13].

Many real world problems comprise a non-linear combination of several sub-problems. The existing interactions between partial solutions impact the quality of the global solution, thus complicating the task of optimization algorithms. Unfortunately, benchmarks adopted by EA researchers tend to ignore this question and the impact of non-linear interactions between problem components is usually outside the discussion of algorithmic effectiveness. The Traveling Thief Problem (TTP) is a recent benchmark [2] that considers the interdependence

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 34–45, 2016.
DOI: 10.1007/978-3-319-30698-8_3

between two well known problems: the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP). The underlying idea behind the TTP is to maximize the profit of a thief that is traveling through a certain number cities stealing items. The thief uses a knapsack with a limited capacity and pays a rent for it, that depends on the time needed to visit all the cities. The interdependence of the TTP emerges from the fact that the speed of the thief depends (non-linearly) on the weight of the items picked so far.

There are a few heuristic approaches to tackle the TTP [3,5,11]. However, most of them seek for solutions by fixing one of the components, while solving the other. The typical approach is to initially set the shortest TSP route for the cities comprising the problem and then solve the remaining KP component. By doing this, a bias towards solutions with small tour lengths is clearly created. Also, it is not guaranteed that the best solutions for the TTP can be found, as large portions of the search space are ignored. The interactions between the two sub-problems ensue that there is a tradeoff between the distance the thief travels and the value that he is able to gather. Since the weights of the items affect the speed of the thief, it is likely that the thief should sometimes slightly increase the tour length, providing that is allows him to pick an heavy, but valuable, item near the end of the tour.

In this paper we present an evolutionary unbiased approach for the TTP, that seeks for complete solutions by simultaneously considering the two sub-problems and the existing interdependence between them. In concrete we rely on an EA where each individual has a tour and a packing plan (items that should be picked at each city). The variation operators modify both components, and a packing heuristic helps creating good packing plans for each individual. The performance of the approach is tested in some TTP benchmarks instances proposed in [11]. Experimental results confirm that it is important to simultaneously take into account both components of the problem.

The remainder of the paper is organized as follows: in Sect. 2 we describe the TTP, whereas Sect. 3 reviews some recent approaches to solve the this problem. Section 4 details our approach to the problem. In Sect. 5 we present and detail the experimental results obtained. Finally, in Sect. 6 we gather the main conclusions and point towards future work.

2 The Traveling Thief Problem

The TTP is a recent benchmark that was created to mimic the interdependence between problems that occur in real-world applications [2]. It is defined as follows: consider a set of cities $N = 1, ..., n$ and a set of items $M = 1, ..., m$, which are distributed among the cities. The distance d_{ij}, with $i, j \in N$ is known. Each city i, except the first one, has a subset of items $M_i = 1, ..., m_i$. Each item k placed in the city i is described by its profit p_{ik} and weight w_{ik}. The thief departs from the first city, visits each city exactly once, and returns to the starting point. Any item may be collected at any city, as long as the total weight of the items in the knapsack do not exceed its capacity W. Additionally the thief has to pay

a rent R for the use of the knapsack for each time unit. When the thief does not have any item in the knapsack it can travel at maximum speed, υ_{max}, whilst when the knapsack is full it travels at a minimum speed υ_{min}. The goal is to find a tour and a packing plan that results in the maximum profit for the thief. Let y_{ik} be a binary variable that is 1 if the item k is picked at city i. The objective function for a given tour $\Pi = (x_1, ...x_n, x_1), x_i \in N$ and a given packing plan $P = (y_{21}, ..., y_{nm_i})$ is:

$$\begin{aligned} Z(\Pi, P) &= \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R * \left(\frac{d_{x_n x_1}}{\upsilon_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\upsilon_{max} - \nu W_{x_i}}\right) \\ \text{s.t.} \sum_{i=1}^{n} \sum_{k=1}^{m_i} w_{ik} y_{ik} &\leq W \end{aligned} \tag{1}$$

where $\nu = \frac{\upsilon_{max} - \upsilon_{min}}{W}$, and W_{xi} is the total weight of collected items that the thief has at city i. The first term of the equation represents the total profit of all picked items, whilst the second term is the total cost of the thief's trip.

Consider the example with 4 cities and corresponding distances depicted in Fig. 1, which was adapted from [11]. Every city, with the exception of the first one, has a set of available items that the thief might choose to pack. As an example, city 2 was two items: $I_{21} : \{profit_{21} = 20, weight_{21} = 2\}$ and $I_{22} : \{profit_{22} = 30, weight_{22} = 3\}$. Specifying that the renting rate $R = 1$, $\upsilon_{max} = 1$, $\upsilon_{min} = 0.1$, and that the maximum capacity of the of the sack is $W = 3$ completely defines the instance.

A possible solution for this instances defines a tour $\Pi : (1, 2, 4, 3, 1)$ and a packing plan $P : (I_{21} = 0, I_{22} = 0, I_{31} = 0, I_{32} = 1, I_{33} = 1, I_{41} = 0)$. In this solution, the thief starts in city 1 and moves to cities 2 and 4 without any items. It then moves to city 3, and, at this moment, the cost of the solution is 15 $(5+6+4)$. In city 3, the thief picks up the items I_{32} and I_{33}, which gives a total profit of 80. When returning to city 1 to complete the tour, the knapsack has a weight of 2, thus reducing the velocity of the thief, corresponding to a traveling cost of 15. All in all, the final fitness value is $Z(\Pi, P) = 80 - 15 - 15 = 50$.

3 Review of the Literature

The TTP problem is a benchmark to study the interdependence between different sub-problems and it was proposed by Bonyadi *et al.* in [2]. In concrete, the TTP results from the combination of the TSP and KP. The authors propose a method to create instances of the TTP, so that researchers can compare the results of different approaches. Additionally, the work shows a simple experiment, which studies how the two problems are connected. In concrete they created a simple instance of the TTP, and separately solved the TSP and KP parts to optimality. Then the best solutions found for each sub-problem are combined, and it is shown that this combination does not correspond to the best solution for the TTP.

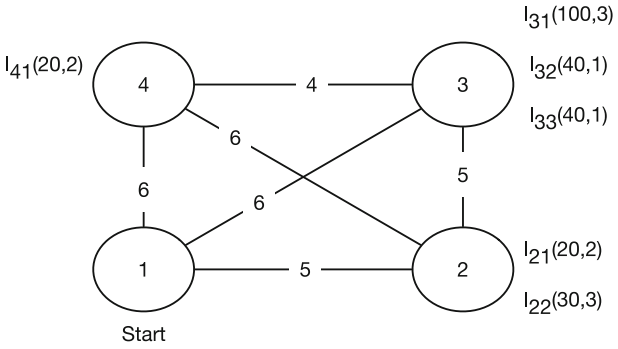


Fig. 1. Example of a TTP instance with 4 cities and 6 items. Adapted from [11]

Later, Polyakovskiy *et al.* in [11] proposes an ensemble of benchmark instances and heuristic methods to tackle the TTP problem. Regarding the instances, they considered the TSPLIB[1] [12] dataset as a starting point. For the KP part of the TTP they created several sets of items following the recommendations of [8]. To generate the TTP instances they considered different combinations of the TSPLIB instances with the KP dataset[2]. Although the creation of a comprehensive dataset was the main goal of the work, it also suggests some simple heuristics to solve the TTP problem. Although the heuristics consider the two sub-problems independently. Firstly they solve the TSP problem to obtain a good tour, disregarding the KP part of the problem. Once a good tour is found, it is kept fixed for the remaining part of the optimization. Then a local search algorithm is used to create a packing plan that achieves a good objective value for the TTP. This work was extended by Faulkner *et al.* [5] and a set of new heuristics were proposed. However the underlying idea is the same: find a good solution for one of the components and then fix it. Although, fixing one of the components of the TTP and neglecting the dependence that exist between the two sub-problems, might prevent the discovery of best solution for the problem.

The work of Mei *et al.* [9] focus on solving large TTP instances with low resource consuming heuristics. They analyze the computational complexity of several different algorithms, and propose a new two-stage local search procedure to create packing plans. The main idea of the algorithm is to first prioritize the insertion of items. Then, as more items are added they check how the addition of a item worsen the thief's speed. They compute some thresholds to decide whether it is worth to add item or not. They incorporate the proposed heuristic into a memetic algorithm, where several initial tour solutions are generated and optimized by the Lin-Kerninghan heuristic (LK) [1]. The algorithm iteratively combines the different solutions to create new packing plans. The approach was

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.
[2] The TTP instances are available at: http://cs.adelaide.edu.au/~optlog/CEC2014COMP_InstancesNew/.

applied to different instances of the TTP, and they show that it was able to outperform previous heuristics when used to solve large instances.

The work presented in [3] puts forward two methods to solve the TTP problem. The first is called Density-based Heuristic (DH). The DH starts by using the LK heuristic to create a tour as short as possible for the TSP. Then it computes the profit that the thief would get if only one item was picked, after completing the tour. They do this for all available items. After it iteratively adds the items to the packing plan. An item is only packed if it does not worsen the overall profit of the TTP.

The second method attempts to solve the TTP by decomposing it in two sub-problems and trying two solve them in parallel. Each component communicates with each other, and from time to time the algorithm tries to combine the solutions of the sub-problems to create an overall approximated solution for the TTP.

4 Evolutionary Approach

The approaches presented in the previous section are the state-of-art for the TTP problem. Despite their relative success there is margin for progress, mostly because they fix one of the TTP's sub-problems. Our approach tries to overcome this by tackling both sub-problems at the same time. For each new tour generated by the optimization algorithm, the packing plan is rearranged.

Our proposal relies on an Evolutionary Algorithm (EA) to search for good solutions for the TTP. The underlying idea behind EA is the simulation of evolution by natural selection of a population of artificial individuals via application of selection, variation operators, and reproduction. These components are guided by a fitness function that evaluates each individual, measuring the quality of the solution it represents. In our approach each individual is composed by a tour and by a packing plan. The tour is a permutation of integers that represents the order in which each city should be visited by the thief. Note that each tour starts and ends in the first city of the instance (city 0). The packing plan is a binary string, that indicates if an item should be picked at a certain city. Fig. 2 shows a simple example of a solution for a TTP instance with 5 cities and 2 items per city.

The EA starts by randomly generating a population of tours for the TTP. Then, for each generated tour, we rely on the packing heuristic described in [11] to create a valid packing plan for the specific tour of the individual. The

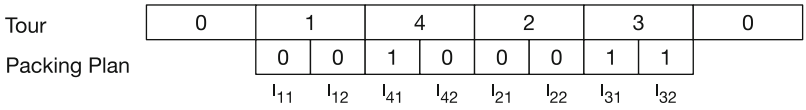


Fig. 2. Example of an EA individual for a TTP instance with 5 cities and 2 items per city

heuristic calculates a score that estimates how profitable an item is, according to a certain tour. Consider that an item I_{x_ik} can be picked at a city x_i, and that d_{x_i} and t_{x_ik} are the total traveling distance and the total traveling time with I_{x_ik} being carried until the end of the tour, respectively. The score of each item is computed has follows:

$$score_{x_ik} = p_{x_ik} - R * t_{x_ik} \tag{2}$$

where

$$t_{x_ik} = \frac{d_{x_i}}{\upsilon_{max} - \nu w_{x_ik}} \tag{3}$$

The $score_{x_ik}$ is the total profit that the thief would obtain if only the item I_{x_ik} is picked during the whole tour. Using this we iteratively select the items that have the highest score. We stop when there are no more items to add, or the knapsack is full. Then each solution is evaluated using the objective function defined by in Eq. 1.

After being evaluated, each solution is improved by a straightforward local search procedure (Algorithm 1), which tries to refine the packing plan over a limited number of iterations. In this procedure we randomly select an item in the current plan and flip its status: if the item is in the knapsack it is removed, else it is added. If this flip results in a better solution to the problem, we keep the new packing plan, else, it is discard. We repeat this until an improvement is found or the maximum number of flips is reached.

Tournament selection chooses the individuals that undergo reproduction. We rely on two variation operators to create new solutions, by modifying existing ones. The first is an adaptation of the Partially Mapped Crossover (PMX) [6]. The second is the Inversion mutation operator for permutations [4]. The recombination operator creates offspring by exchanging the information about the tours as well as the items that the thief picked at each city. When switching cities in the tour, the operators also move the items that were picked at that city (Fig. 3).

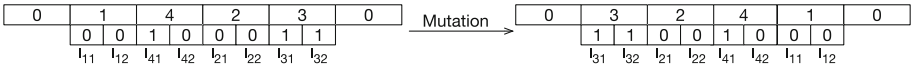


Fig. 3. Example of the application of the mutation operation

It is known that the PMX operator alone is not particularly effective, partially because it does not transmit common edges between the parents to the offspring. However using it together with inversion mutation yielded better results than using, for instance, edge recombination. This is in accordance with the observations made by [7]: using inversion mutation (2-opt) in solutions that have been already improved by other methods can result in poorer overall results than using inversion mutation on worst solutions.

Having a new tour, the packing plan is corrected/improved to take into account the new routes that the thief has to travel. We rely again on the packing

heuristic described above to select the most promising items and/or to remove the items that are in excess. Subsequently the new solutions are evaluated, and the packing plan undergoes a new iteration of the local search. Algorithm 2 outlines the proposal described.

Algorithm 1. Local Search

```
function LOCALIMPROVEMENT(ind)
    tempInd ← ind
    i ← 0
    while i < MaximumLocalImprovementIterations do
        posToInvert ← randomInteger(0, length(tempInd.packingPlan))
        if tempInd.packingPlan[posToInvert] = 1 then
            tempInd.packingPlan[posToInvert] ← 0
        else
            tempInd.packingPlan[posToInvert] ← 1
        end if
        evaluate tempInd using Eq. 1
        if tempInd.Fitness > ind.Fitness then
            ind ← tempInd
            return ind
        end if
    end while
    return ind
end function
```

5 Experiments

In this section we report the results attained by our approach, and compare it with an implementation of the DH heuristic described in [3]. We conducted the experimental study using instances with a number of cities ranging from 51 to 100. Each instance is composed by a group of cities, a set of items, the maximum (v_{max}) and minimum (v_{min}) velocity at which the thief can travel, the maximum capacity W of the knapsack, the renting ratio R, and the number of items available per city. In the instances considered every city (except the first) has the same number of items, which are either 1 or 3 items. For this study we selected instances whose items have their profit strongly correlated to their weight. The higher the correlation between weight and profit, the more time consuming is the KP to solve [8,10].

The experimental parameters for the EA are described in Table 1. The algorithm is composed by 100 individuals that are evolved for $2.5 * 10^5$ function evaluations. Contrary to previous approaches to this problem, we use the number of evaluations instead of time as stop criterion, as this simplifies the reproducibility of results. We performed 30 independent runs of the EA on each

Algorithm 2. Evolutionary Algorithm for the TTP

```
create population Pop randomly
evaluations ← 0
for all ind ∈ Pop do
    ind.PackingPlan ← packingHeuristic(ind.Tour)
    evaluate ind using Eq. 1
    ind ← localImprovement(ind)
end for
while evaluations < maximumNumberEvaluations do
    Parents ← parentSelection(Pop)
    Offspring ← AdaptedPMX(RecombinationRate, Parents)
    Offspring ← InversionMutation(MutationRate, Offspring)
    for all ind ∈ Offspring do
        ind.PackingPlan ← packingHeuristic(ind.Tour)
        evaluate ind using Eq. 1
        ind ← localImprovement(ind)
    end for
    Pop ← selectSurvivors(Pop, Offspring)
end while
return best Solution discovered
```

instance considered in this study. We do not present a statistical analysis comparing the results of the EA and the DH heuristic, since for this second method we only have access to the best solutions found.

5.1 Results

A summary of the optimization results is presented in Table 2. The first column identifies the instance, with the number indicating how many cities the thief has to visit (e.g., eil51 has 51 cities). The second column (*Items per City*) shows the

Table 1. Parameter Settings

Parameter	Value
Runs	30
Population Size	100
Evaluations	$2.5 * 10^5$
Parent Selection	Tournament with size 5
Replacement	Generational
Recombination Operator	Partially Mapped Crossover
Mutation Operator	Inversion
Recombination Rate	0.9
Mutation Rate	0.1
Local Search Iterations	#cities * #items

Table 2. Results obtained using the Evolutionary Approach proposed in this work and the DH heuristic

Instance	Items Per City	Algorithms		
		DH	EA	
			Best	MBF
eil51	1	2591.95	**4706.54**	4088.90 (± 309.95)
	3	9163.66	**10115.55**	8247.07 (± 1067.26)
eil76	1	4264.13	**5506.92**	4502.23 (± 737.26)
	3	10583.40	**12040.77**	8871.19 (± 1805.64)
kroA100	1	**7095.97**	6442.41	4392.21 (± 1035.68)
	3	**25923.00**	22325.30	17989.43 (± 2757.97)

number of items that are available in each city. The remaining columns contain the results obtained by the DH heuristics and by the EA proposed in this paper. The outcomes of the EA include the fitness of the best solution found (column *Best*) and the average and standard deviation of the best solutions found in the 30 runs (Mean Best Fitness - column *MBF*).

An overview of the results reveals that the EA is able to find promising solutions and is competitive with existing state-of-the-art approaches for the TTP. In concrete, it discovered clearly better solutions than DH in 4 of the selected instances. For the two larger instances (kroA100 with 1 and 3 items), it found solutions whose quality is not worse than 9 % and 13 %, respectively. The difficulties in the larger instances are not unexpected and are probably related to the increasing size of the search space. The DH heuristic breaks the problems in two independent components: it firstly finds an high quality tour and fixes it. Then it tries to create the best packing plan that fits into that tour. By doing this, DH narrows the search space to a region of solutions that are based on (near) optimal tours. Although this simplifies the task of creating a packing plan, there is no guarantee that the best solution for the compete problem is found. On the contrary, the EA is performing its search in the space of all possible TTP solutions. Increasing the number of cities and/or the number of items results in larger search spaces. Since the number of evaluations is kept fixed for all instances considered in the study, it is likely that there is degradation in the performance of the algorithm, since it does not have time to effectively sample the search space and discover the region(s) where the promising solutions are.

The results from the last column of Table 2 show that the number of items available per city impact the variance of the results: for all considered instances, more items lead to a higher standard deviation. This is another piece of evidence that confirms the decreased performance of the EA when dealing with larger instances.

5.2 Solution Analysis

Our approach is based on the argument that solving one of the sub-problems (namely the TSP) and then solving the other provides compromises the quality

of the global solution and prevents a meaningful exploration of the search space. Results presented in the previous section revealed that solving the TTP as a whole is advantageous and creates conditions for the discovery of enhanced solutions. Here we analyse some features of the solutions discovered by the EA to gain a better insight into the optimization behavior of this approach. Specifically, we are interested in comparing the tour that provides the best fitness for the TTP, with the best tour for the corresponding TSP problem. If they are different, it supports the claim that non-linear components of an optimization problem should not be solved independently.

The comparison of the tours is performed as follows. We take the permutation set that represent the best TSP tour for each instance and compute a set G_1 with the number of transitions. A transition is a tuple (x_i, x_{i+1}), that indicates a direct move from city x_i to city x_{i+1}. The same procedure is followed to create the set G_2 with all transitions from the best tour found by the EA. Finally we created a set $G_3 = G_1 \cap G_2$, containing all transitions that are shared between G_1 and G_2. Table 3 present the percentage of transitions that are different between the best TSP tour and the best TTP tour found by the EA. The results show that the TTP tours tend to be substantially different from the TSP tours. The percentage of different transitions ranges from 31.4 % to 71.1 %. Moreover, in all cases but one, the difference exceeds 50 %. These values indicate that the tours are substantially different, and that fixing the tour might compromise the discovery of effective solutions for the TTP.

Table 4 compares the distances of the best TTP tours with the optimal distance of the TSP tour. The column *TSP Optimal Distance* represents the best known solution for the TSP, whereas column *TTP Distance* represents the distance of the tour associated with the best TTP solution. A brief perusal show that the TTP tours are longer, suggesting that sometimes it is worth to delay the pick of a valuable item that is in a particular city. This might result in slightly longer tours, but in lower carrying cost.

The results presented in this section help to justify why it is important to solve the TTP problem as a whole. Solving one of the sub-problems and then solving the other compromises the interdependence between components, leading to poorer global results. The study presented here is a first step towards a better

Table 3. Percentage of transitions that are different between the TSP best known tour and the tour belonging to the best TTP solution found by the EA

Instance	Items Per City	#Different Edges (in %)
eil51	1	58.8
	3	31.4
eil76	1	71.1
	3	51.3
kroA100	1	69.0
	3	52.0

Table 4. Difference between the distance of the TSP tours and the TTP tours.

Instance	TSP Optimal Distance	Items Per City	TTP Distance	Absolute Difference (in %)
eil51	426	1	495	16.2
		3	470	10.3
eil76	538	1	695	29.2
		3	641	19.1
kroA100	21282	1	26691	25.4
		3	24847	16.8

understanding of the non-linear interconnections occurring in the TTP. It still has some limitations, as only a subset of instances were considered and the analysis focuses just on the influence of relying in fixed TSP optimal tours. A logical next step if to study the impact of fixing the set of items, while trying to evolve a tour.

6 Conclusions and Future Work

In this paper we proposed an unbiased approach to the Traveling Thief Problem (TTP). The TTP is a new benchmark that results form the combination of two well-known problems, the Traveling Salesman Problem and the Knapsack Problem. The benchmark is created in such a way that it promotes a relation of interdependence between the two sub-problems. This means that the solution of one sub-problem influences the quality of the solutions for the other sub-problem. Based on this, solving one sub-problem alone, even to optimality, might result in a inferior performance, when considering thew global optimization scenario.

Almost all of the existing approaches described in the literature solve the TTP problem by fixing one of its sub-problems, thus creating a bias towards some solutions. These approaches create a reasonably good TSP tour (most of the times they use the optimal solution), fix it, and try to find the set of items the give the maximum profit for that tour.

Our approach follows a different direction. We proposed an Evolutionary Algorithm (EA) that solves the TTP as a whole. The EA evolves a population of individual solutions, where each solution is a tour and a packing plan. During the evolutionary process both components are modified, keeping the synergy that exists between the sub-problems of the TTP. The results obtained confirm the potential of the approach, as the EA was able to find good quality solutions in most of the instances used. We also performed an analysis on the structure of the best solutions found. Specifically, we compared the optimal TSP tours with the tours proposed by the EA to the corresponding TTP problem and verified that, in all but one case, they differ in more than 50 % of the transitions. This is an important result for it shows that fixing one of the sub-problems might compromise the discovery of global effective solutions.

More experiments are still needed to fully comprehend the relation between the two problems. In the near future we will focus our efforts in understanding how the distribution of items through the cities affects the quality of the

results. Also, we intended to address the scalability issues, in order to solve larger instances. Finally, we plan to develop alternative representations that create a more symbiotic relationship between the two sub-problems.

Acknowledgments. This work was partially supported by Fundação para a Ciência e Tecnologia (FCT), Portugal, under the grant SFRH/BD/79649/2011.

References

1. Applegate, D., Cook, W., Rohe, A.: Chained lin-kernighan for large traveling salesman problems. INFORMS J. Comput. **15**(1), 82–92 (2003)
2. Bonyadi, M.R., Michalewicz, Z., Barone, L.: The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 1037–1044. IEEE (2013)
3. Bonyadi, M.R., Michalewicz, Z., Roman Przybyoek, M., Wierzbicki, A.: Socially inspired algorithms for the travelling thief problem. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation. pp. 421–428. ACM (2014)
4. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg (2003)
5. Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, pp. 385–392. ACM (2015)
6. Goldberg, D.E., Lingle, R.: Alleles, loci, and the traveling salesman problem. In: Proceedings of the first International Conference on Genetic Algorithms and their Applications pp. 154–159 (1985)
7. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: a case study in local optimization. Local Search Comb. Optim. **1**, 215–310 (1997)
8. Martello, S., Pisinger, D., Toth, P.: Dynamic programming and strong bounds for the 0–1 knapsack problem. Manage. Sci. **45**(3), 414–424 (1999)
9. Mei, Y., Li, X., Yao, X.: Improving efficiency of heuristics for the large scale traveling thief problem. In: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (eds.) SEAL 2014. LNCS, vol. 8886, pp. 631–643. Springer, Heidelberg (2014)
10. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Heidelberg (1996)
11. Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: A comprehensive benchmark set and heuristics for the traveling thief problem. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, pp. 477–484. ACM (2014)
12. Reinelt, G.: Tspliba traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
13. White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaśkowski, W., OReilly, U.M., Luke, S.: Better gp benchmarks: community survey results and proposals. Genet. Program. Evolvable Mach. **14**(1), 3–29 (2013)

Construct, Merge, Solve and Adapt: Application to the Repetition-Free Longest Common Subsequence Problem

Christian Blum[1,2(✉)] and Maria J. Blesa[3]

[1] Department of Computer Science and Artificial Intelligence, University of the Basque Country, San Sebastian, Spain
christian.c.blum@gmail.com
[2] IKERBASQUE, Basque Foundation for Science, Bilbao, Spain
[3] ALBCOM Research Group, Computer Science Department, Universitat Politécnica de Catalunya - BarcelonaTech, Barcelona, Spain
mjblesa@cs.upc.edu

Abstract. In this paper we present the application of a recently proposed, general, algorithm for combinatorial optimization to the repetition-free longest common subsequence problem. The applied algorithm, which is labelled Construct, Merge, Solve & Adapt, generates sub-instances based on merging the solution components found in randomly constructed solutions. These sub-instances are subsequently solved by means of an exact solver. Moreover, the considered sub-instances are dynamically changing due to adding new solution components at each iteration, and removing existing solution components on the basis of indicators about their usefulness. The results of applying this algorithm to the repetition-free longest common subsequence problem show that the algorithm generally outperforms competing approaches from the literature. Moreover, they show that the algorithm is competitive with CPLEX for small and medium size problem instances, whereas it outperforms CPLEX for larger problem instances.

Keywords: Hybrid algorithm · Combining metaheuristics with ILP solvers · Repetition-free longest common subsequence problem

1 Introduction

The problem that is often encountered when applying exact solvers to combinatorial optimization problems is that they are not applicable to problem instances

This work was supported by project TIN2012-37930-C02-02 (Spanish Ministry for Economy and Competitiveness, FEDER funds from the European Union) and project SGR 2014-1034 (AGAUR, Generalitat de Catalunya). Additionally, Christian Blum acknowledges support from IKERBASQUE. Our experiments have been executed in the High Performance Computing environment managed by RDlab (http://rdlab.cs.upc.edu) and we would like to thank them for their support.

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 46–57, 2016.
DOI: 10.1007/978-3-319-30698-8_4

of realistic sizes. However, for smaller problem instances, exact solvers are often surprisingly efficient. This is because the operations research community has invested a lot of time, effort and expertise into the development of exact solvers. Prime examples are general-purpose mathematical programming solvers such as CPLEX and Gurobi. Therefore, recent research efforts have focused on ways in which efficient exact solvers can be used within heuristic frameworks even in the context of large problem instances. One of the most recent examples of these research efforts is an algorithm labelled Construct, Merge, Solve & Adapt (CMSA) [1,2]. This algorithm works as follows. At each iteration, solutions to the tackled problem instance are generated in a probabilistic way. The solution components found in these solutions are then added to a sub-instance of the original problem instance. Subsequently, an exact solver such as, for example, CPLEX is used to solve the sub-instance to optimality. Moreover, the algorithm is equipped with a mechanism for deleting seemingly useless solution components from the sub-instance. This is done such that the sub-instance has a moderate size and can be solved rather quickly to optimality.

In this work we apply the CMSA algorithm to the so-called repetition-free longest common subsequence problem [3]. This problem, which is NP-hard, is a special case of the well-known longest common subsequence problem. The repetition-free longest common subsequence problem seems to be well-suited for being solved with CMSA, because the standard integer linear programming (ILP) model for the problem can only be solved to optimality in the context of rather small problem instance. Both the number of variables and constraints in this model (which is outlined later in Sect. 2) are exponential in the input parameters of the problem. The obtained results show that, indeed, the application of CMSA obtains state-of-the-art results, especially in the context of large problem instances.

The remaining part of the paper is organized as follows. In Sect. 2 we provide a technical description of the repetition-free longest common subsequence problem. Moreover, we describe the standard ILP model for this problem. Next, in Sect. 3, the application of CMSA to the tackled problem is outlined. Finally, Sect. 4 provides an extensive experimental evaluation and Sect. 5 offers a discussion and an outlook to future work.

2 Repetition-Free Longest Common Subsequence Problem

The longest common subsequence (LCS) problem is a string problem with numerous applications, for example, in computational biology [4–6]. A problem instance (S, Σ) consists of a set $S = \{s_1, s_2, \ldots, s_n\}$ of n input strings over a finite alphabet Σ. The goal consists in finding the longest possible subsequence of all strings in S. A string t is a subsequence of a string s, if t can be produced from s by deleting zero or more characters. For example, *dga* can be produced from *adagtta* by deleting the first two occurrences of letter *a* and the two occurrences of letter *t*. Apart from applications in computational biology, the LCS

problem finds applications, for example, in data compression and file comparison [7,8]. Moreover, note that the LCS problem was shown to be NP-hard [9] for an arbitrary number n of input strings.

In this work we consider a restricted version of the LCS problem, the so-called repetition-free longest common subsequence (RFLCS) problem. Given exactly two input strings x and y over a finite alphabet Σ, the goal is to find a longest common subsequence with the additional restriction that no letter may appear more than once. This problem was introduced in [3] as a comparison measure for two sequences of biological origin. In the same paper, the authors proposed three heuristics for solving this problem. Other algorithms from the literature for solving the RFLCS problem include a Beam-ACO approach [10] and an evolutionary algorithm [11]. Among these techniques, the Beam-ACO approach can be regarded as the current state-of-the-art method.

The RFLCS problem can be stated in terms of an integer linear program (ILP) in the following way. First, let us denote the length of x by l_x and the length of y by l_y. Furthermore, the positions in x and y are numbered from 1 to l_x, respectively from 1 to l_y. The letter at position i of x is denoted by $x[i]$, and the letter at position j of y is denoted by $y[j]$. The set Z of binary variables that is required for the ILP model is composed as follows. For each combination of $i = 1, \ldots, l_x$ and $j = 1, \ldots, l_y$ such that $x[i] = y[j]$, set Z contains a binary variable $z_{i,j}$. Moreover, we say that two variables $z_{i,j}$ and $z_{k,l}$ are *in conflict*, if and only if either $i < k$ and $j > l$ or $i > k$ and $j < l$. Finally, for each letter $a \in \Sigma$, set $Z_a \subset Z$ contains all variables $z_{i,j}$ such that $x[i] = y[j] = a$. The RFLCS problem can then be rephrased as the problem of selecting a maximal number of non-conflicting variables from Z provided that, among all variables representing a letter $a \in \Sigma$, at most one variable is chosen. Given these notations, the ILP is stated as follows.

$$\textbf{max} \sum_{z_{i,j} \in Z} z_{i,j} \tag{1}$$

subject to:

$$\sum_{z_{i,j} \in Z_a} z_{i,j} \leq 1 \quad \text{for } a \in \Sigma \tag{2}$$

$$z_{i,j} + z_{k,l} \leq 1 \quad \text{for all } z_{i,j} \text{ and } z_{k,l} \text{ being in conflict} \tag{3}$$

$$z_{i,j} \in \{0, 1\} \quad \text{for } z_{i,j} \in Z \tag{4}$$

Hereby, constraints (2) ensure that each letter from the alphabet is chosen at most once, and constraints (3) ensure that selected variables are not in conflict.

3 Application of CMSA to the RFLCS Problem

The application of CMSA to the RFLCS problem is pseudo-coded in Algorithm 1. Note that, in the context of this algorithm, solutions to the problem and sub-instances are both subsets of the complete set Z of variables. If a solution S

Algorithm 1. CMSA for the RFLCS problem

```
1: input: strings x and y over alphabet Σ, values for parameters n_a and age_max
2: S_bsf := NULL, Z_sub := ∅
3: age[z_{i,j}] := 0 for all z_{i,j} ∈ Z
4: while CPU time limit not reached do
5:    for i = 1, ..., n_a do
6:       S := ProbabilisticSolutionConstruction(Z)
7:       for all z_{i,j} ∈ S and z_{i,j} ∉ Z_sub do
8:          age[z_{i,j}] := 0
9:          Z_sub := Z_sub ∪ {z_{i,j}}
10:      end for
11:   end for
12:   S'_opt := ApplyILPSolver(Z_sub)
13:   if |S'_opt| > |S_bsf| then S_bsf := S'_opt
14:   Adapt(Z_sub, S'_opt, age_max)
15: end while
16: output: S_bsf
```

contains a variable $z_{i,j}$, this means that this variable must be given value one in order to produce the corresponding solution. The main loop of CMSA is executed while the CPU time limit is not reached. It consists of the following actions. First, the best-so-far solution S_{bsf} is initialized to NULL, and the restricted problem instance (Z_{sub}) to the empty set. Then, at each iteration a number of n_a solutions is probabilistically constructed in function ProbabilisticSolutionConstruction(Z) in line 6 of Algorithm 1. The variables contained in these solutions are added to Z_{sub}. The age of a newly added variable $z_{i,j}$ (age$[z_{i,j}]$) is set to 0. After the construction of n_a solutions, an ILP solver is applied to find the best solution S'_{opt} in the sub-instance Z_{sub} (see function ApplyILPSolver(Z_{sub}) in line 12 of Algorithm 1). In case S'_{opt} is better than the current best-so-far solution S_{bsf}, solution S'_{opt} is stored as the new best-so-far solution (line 13). Next, sub-instance Z_{sub} is adapted, based on solution S'_{opt} and on the age values of the variables. This is done in function Adapt(Z_{sub}, S'_{opt}, age$_{\text{max}}$) in line 14 as follows. First, the age of each variable in Z_{sub} is increased by one, and, subsequently, the age of each variable in $S'_{\text{opt}} \subseteq Z_{\text{sub}}$ is re-initialized to zero. Finally, those solution components from Z_{sub} whose age has reached the maximum component age (age$_{\text{max}}$) are deleted from Z_{sub}. The motivation behind the aging mechanism is that variables which never appear in an optimal solution of Z_{sub} should be removed from Z_{sub} after a while, because they simply slow down the ILP solver. On the other side, components which appear in optimal solutions seem to be useful and should therefore remain in Z_{sub}.

In the following we will describe in detail the remaining component of the algorithm: the probabilistic construction of solutions in function ProbabilisticSolutionConstruction(Z). Such a solution construction starts with an empty solution $S = \emptyset$, and the first step consists in generating the set of variables that serve as options to be added to S. More specifically, the initial set C is generated in

order to contain for each letter $a \in \Sigma$ the variable $z_{i,j} \in Z_a$ (if any) such that $i < k$ and $j < l$, $\forall z_{k,l} \in Z_a$. Moreover, options $z_{i,j} \in C$ are given a weight value $w(z_{i,j}) := \frac{i}{l_x} + \frac{j}{l_y}$, which is a known greedy function for longest common subsequence problems. At each construction step, exactly one variable is chosen from C and added to S. For doing so, first, a value r is chosen uniformly at random from $[0, 1]$. In case $r \leq d_{\text{rate}}$, where d_{rate} is a parameter of the algorithm, the variable $z_{i,j} \in C$ with the smallest weight value is deterministically chosen. Otherwise, a candidate list $L \subseteq C$ of size $\min\{l_{\text{size}}, |C|\}$ containing the options with the lowest weight values is generated and exactly one variable $z_{i,j} \in L$ is then chosen uniformly at random and added to S. Note that l_{size} is another parameter of the solution construction process. Finally, the set of options C for the next construction step is generated. This is done such that C only contains variables that represent letters that are not already represented by one of the variables in S. Moreover, being $z_{i,j}$ the last variable that was added to S, C contains for each non-represented letter $a \in \Sigma$ the variable $z_{r,s} \in Z_a$ (if any) with the lowest weight value $w(z_{r,s})$ calculated as $w(z_{r,s}) := \frac{r-i}{l_x-i} + \frac{s-j}{l_y-j}$. The construction of a complete (valid) solution is finished when the set of options is empty.

4 Experimental Evaluation

We implemented the proposed algorithm in ANSI C++ using GCC 4.7.3, without the use of any external libraries. The ILP models, both the ones of the original RFLCS instances and the ones of sub-instances within CMSA, were solved with IBM ILOG CPLEX v12.1 in one-threaded mode. The experimental evaluation has been performed on a cluster of PCs with Intel(R) Xeon(R) CPU 5670 CPUs of 12 nuclei of 2933 MHz and at least 40 Gigabytes of RAM. In the following we first describe the set of benchmark instances that we generated to test the CMSA algorithm. Then, we describe the tuning experiments that were performed in order to determine a proper setting for the parameters. Finally, an exhaustive experimental evaluation is presented.

4.1 Problem Instances

Two sets of problem instances were adopted from [10]. These sets were generated with the same procedure as described in [3]. The first set (henceforth called Set1) consists for each combination of input sequence length $n \in \{32, 64, 128, 256, 512\}$ and alphabet size $|\Sigma| \in \{n/8, n/4, 3n/8, n/2, 5n/8, 3n/4, 7n/8\}$ of exactly 10 problem instances. The second set of instances (henceforth called Set2) is generated on the basis of alphabet sizes $|\Sigma| \in \{4, 8, 16, 32, 64\}$ and the maximal repetition of each letter $rep \in \{3, 4, 5, 6, 7, 8\}$ in each input string. For each combination of $|\Sigma|$ and rep this instance set consists of 10 randomly generated problem instances. In addition, we generated an extension of Set2 consisting of larger problem instances. More specifically, we generated for each combination of $|\Sigma| \in \{128, 256\}$ and $rep \in \{3, 4, 5, 6, 7, 8\}$ ten problem instances. All the

results to be shown in the forthcoming sections are averages over the 10 problem instances of each type.

4.2 Tuning of CMSA

There are several parameters involved in CMSA for which well-working values must be found: (n_a) the number of solution constructions per iteration, (age_{max}) the maximum allowed age of solution components, (d_{rate}) the determinism rate, (l_{size}) the candidate list size, and (t_{max}) the maximum time in seconds allowed for CPLEX per application to a sub-instance. The last parameter is necessary, because even when applied to reduced problem instances, CPLEX might still need too much computation time for solving such sub-instances to optimality. In any case, CPLEX always returns the best feasible solution found within the given computation time.

We decided to make use of the automatic configuration tool irace[12] for the tuning of the five parameters. In fact, irace was applied to tune CMSA separately for each alphabet size, which—after initial experiments—seems to have more influence on the behavior of the algorithm than the length of the input strings. In the context of SET1 we randomly generated two tuning instances for each combination of string length and alphabet size, whereas for SET2 (and its extension) we randomly generated two tuning instances for each combination of alphabet size and number of repetitions.

The tuning process for each alphabet size was given a budget of 200 runs of CMSA, where each run was given a computation time limit of $l_x/10$ CPU seconds for instances of SET1 (remember that for instances of SET1 it holds that $l_x = l_y$) and $(|\Sigma| * reps)/10$ CPU seconds for instances of SET2 and its extension. Finally, the following parameter value ranges were chosen concerning the five parameters of CMSA:

- $n_a \in \{10, 30, 50\}$
- $\text{age}_{\text{max}} \in \{1, 5, 10, inf\}$, where *inf* means that solution components are never removed from Z_{sub}.
- $d_{\text{rate}} \in \{0.0, 0.3, 0.5, 0.7, 0.9\}$, where a value of 0.0 means that the selection of the next variable to be added to the partial solution under construction is always done randomly from the candidate list, while a value of 0.9 means that solution constructions are nearly deterministic.
- $l_{\text{size}} \in \{3, 5, 10\}$
- $t_{\text{max}} \in \{0.5, 1.0, 5.0\}$ (in seconds) for instances of SET1 and SET2, and $t_{\text{max}} \in \{1.0, 10.0, 100.0\}$ for the instances of the extension of SET2.

The tuning runs with iraceproduced the configurations of CMSA as shown in Table 1.

4.3 Experimental Results

Three algorithms were included in the comparison. Beam-ACO is currently a state-of-the-art method for the RFLCS problem [10], CPLEX refers to the application of CPLEX to the complete problem instances, and CMSA is the algorithm

Table 1. Results of tuning CMSA with irace.

(a) Tuning results for the seven alphabet sizes of Set1.

$\|\Sigma\|$	n_a	age_{max}	d_{rate}	l_{size}	t_{max}
$n/8$	30	5	0.7	5	0.5
$n/4$	10	1	0.7	10	1.0
$3n/8$	30	1	0.3	10	5.0
$n/2$	50	1	0.3	3	5.0
$5n/8$	30	5	0.7	10	5.0
$3n/4$	30	5	0.5	5	5.0
$7n/8$	30	5	0.0	10	5.0

(b) Tuning results for the seven alphabet sizes of Set2 and its extension.

$\|\Sigma\|$	n_a	age_{max}	d_{rate}	l_{size}	t_{max}
4	10	*inf*	0.9	10	1.0
8	10	10	0.9	5	0.5
16	50	*inf*	0.7	3	0.5
32	50	*inf*	0.5	10	5.0
64	10	10	0.5	5	1.0
128	30	1	0.5	5	10.0
256	10	1	0.7	3	10.0

proposed in this work. The results of Beam-ACO for the instances of Set1 and Set2 were taken from [10], where Beam-ACO was applied once to each problem instance with a computation time limit of 5 CPU seconds per run, a beam width of 10, and a determinism rate of 0.9. Note that the low computation time limit of 5 CPU seconds was adopted in [10], because Beam-ACO always produced its best results during the first seconds of a run. For the application to the larger problem instances that were generated as an extension of Set2 Beam-ACO was applied with the same parameter values for beam with and determinism rate, but with the same computation time limit as CMSA. In particular, CMSA was applied to each problem instance with a computation time limit of $l_x/10$ CPU seconds for instances of Set1 (remember that for instances of Set1 it holds that $l_x = l_y$) and $(|\Sigma| * reps)/10$ CPU seconds for instances of Set2 and its extension. The stand-alone application of CPLEX to each problem instances was given more computation time, namely, 600 CPU seconds for each run, regardless of the instance/alphabet size. Moreover, a memory limit of 2 GB were used for each application of CPLEX.

The numerical results are presented in Table 2 concerning Set1, Table 3 concerning Set2, and Table 4 concerning the extension of Set2. Each table row presents the results averaged over 10 problem instances of the same type. The results of Beam-ACO and CMSA are provided in two columns each. The first one (with heading **result**) provides the result of the corresponding algorithm averaged over 10 problem instances, while the second column (with heading **time (s)**) provides the average computation time necessary for finding the corresponding solutions. The same information is given for CPLEX. However, in this case we also provide the average optimality gaps (in percent), that is, the average gaps between the upper bounds and the values of the best solutions when stopping a run.

The results allow to make the following observations:

– First of all, CPLEX is able to provide optimal solutions for all instances of 29 out of 35 instance types (that is, table rows) concerning Set1, and for all instances of 27 out of 30 instance types concerning Set2. This means, on one side, that the instances of these two benchmark sets are, in their majority,

Table 2. Experimental results concerning the instances of Set1.

$\|\Sigma\|$	n	Beam-ACO		CPLEX			CMSA	
		result	**time (s)**	**result**	**time (s)**	**gap (%)**	**result**	**time (s)**
	32	4.0	< 0.1	4.0	0.1	0.0	4.0	< 0.1
	64	8.0	< 0.1	8.0	0.8	0.0	8.0	< 0.1
$n/8$	128	16.0	< 0.1	16.0	9.6	0.0	16.0	< 0.1
	256	31.9	< 0.1	n.a.	n.a.	n.a.	31.8	0.1
	512	62.3	1.8	n.a.	n.a.	n.a.	60.4	13.3
	32	7.9	< 0.1	7.9	0.1	0.0	7.9	< 0.1
	64	14.3	< 0.1	14.4	0.3	0.0	14.4	< 0.1
$n/4$	128	25.3	0.2	25.9	17.2	0.0	25.7	1.1
	256	42.4	0.7	41.6	495.1	8.6	42.6	3.6
	512	68.0	0.8	n.a.	n.a.	n.a.	68.7	7.1
	32	8.7	< 0.1	9.0	< 0.1	0.0	9.0	< 0.1
	64	14.4	< 0.1	14.8	0.2	0.0	14.8	< 0.1
$3n/8$	128	25.1	< 0.1	25.3	3.1	0.0	25.3	< 0.1
	256	39.7	0.2	40.1	133.5	0.0	40.1	1.5
	512	59.4	1.3	7.0	36.2	> 100.0	59.5	3.2
	32	8.8	< 0.1	8.8	< 0.1	0.0	8.8	< 0.1
	64	14.5	< 0.1	14.6	0.1	0.0	14.6	< 0.1
$n/2$	128	23.4	< 0.1	23.4	1.0	0.0	23.3	< 0.1
	256	34.1	0.2	34.3	30.5	0.0	34.1	0.3
	512	53.1	0.6	14.9	207.5	> 100.0	53.1	5.9
	32	7.9	< 0.1	7.9	< 0.1	0.0	7.9	< 0.1
	64	13.7	< 0.1	13.7	< 0.1	0.0	13.7	< 0.1
$5n/8$	128	21.1	< 0.1	21.1	0.5	0.0	21.1	< 0.1
	256	31.1	0.2	31.2	10.4	0.0	31.2	1.6
	512	47.8	0.3	47.9	308.3	0.0	47.8	2.9
	32	7.8	< 0.1	7.8	< 0.1	0.0	7.8	< 0.1
	64	13.1	< 0.1	13.3	< 0.1	0.0	13.3	< 0.1
$3n/4$	128	19.1	< 0.1	19.1	0.2	0.0	19.1	< 0.1
	256	30.0	< 0.1	30.1	4.3	0.0	30.1	1.3
	512	44.7	0.5	44.8	115.5	0.0	44.8	1.3
	32	7.6	< 0.1	7.6	< 0.1	0.0	7.6	< 0.1
	64	12.2	< 0.1	12.2	< 0.1	0.0	12.2	< 0.1
$7n/8$	128	18.5	< 0.1	18.5	0.2	0.0	18.5	< 0.1
	256	27.2	< 0.1	27.2	2.2	0.0	27.1	0.1
	512	40.7	0.3	40.9	59.4	0.0	40.8	4.3

not very difficult to be solved. On the other side, there seems to be a kind of phase transition between instances that can be solved to optimality quite easily, and instances that are difficult to be solved. In three out of six instance types of Set1 which CPLEX cannot solve to optimality within the allocated CPU time, the allocated memory is not sufficient, and for other two instance types the average optimality gap is greater than 100 %.

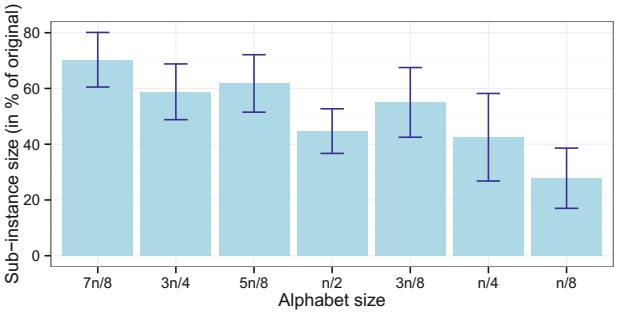
Table 3. Experimental results concerning the instances of SET2.

$\|\Sigma\|$	n	Beam-ACO		CPLEX			CMSA	
		result	**time (s)**	**result**	**time (s)**	**gap (%)**	**result**	**time (s)**
4	3	3.4	< 0.1	3.4	< 0.1	0.0	3.4	< 0.1
	4	3.8	< 0.1	3.8	< 0.1	0.0	3.8	< 0.1
	5	3.8	< 0.1	3.8	< 0.1	0.0	3.8	< 0.1
	6	3.8	< 0.1	3.8	< 0.1	0.0	3.8	< 0.1
	7	3.9	< 0.1	3.9	< 0.1	0.0	3.9	< 0.1
	8	4.0	< 0.1	4.0	< 0.1	0.0	4.0	< 0.1
8	3	5.9	< 0.1	5.9	< 0.1	0.0	5.9	< 0.1
	4	6.7	< 0.1	6.7	< 0.1	0.0	6.7	< 0.1
	5	6.8	< 0.1	7.0	< 0.1	0.0	7.0	< 0.1
	6	7.3	< 0.1	7.3	< 0.1	0.0	7.3	< 0.1
	7	7.6	< 0.1	7.7	< 0.1	0.0	7.7	< 0.1
	8	7.5	< 0.1	7.5	< 0.1	0.0	7.5	< 0.1
16	3	9.6	< 0.1	9.6	< 0.1	0.0	9.6	< 0.1
	4	11.1	< 0.1	11.1	< 0.1	0.0	10.9	< 0.1
	5	13.7	0.2	13.8	< 0.1	0.0	13.6	0.2
	6	13.0	< 0.1	13.2	0.1	0.0	13.1	< 0.1
	7	14.5	< 0.1	14.7	0.3	0.0	14.7	< 0.1
	8	14.7	< 0.1	15.2	0.6	0.0	15.1	0.3
32	3	16.1	< 0.1	16.1	< 0.1	0.0	16.1	< 0.1
	4	19.2	< 0.1	19.2	0.4	0.0	19.2	< 0.1
	5	20.6	0.1	20.9	1.3	0.0	20.9	< 0.1
	6	24.0	0.5	24.4	5.8	0.0	24.4	0.2
	7	24.9	< 0.1	25.8	9.4	0.0	25.8	2.8
	8	26.8	0.4	27.4	32.2	0.0	27.4	1.5
64	3	24.8	< 0.1	24.9	1.8	0.0	24.9	0.3
	4	30.1	0.1	30.3	8.7	0.0	30.3	0.9
	5	34.5	0.2	34.8	70.5	0.0	34.7	1.8
	6	38.4	0.4	38.8	231.4	1.7	39.0	8.4
	7	42.3	0.4	42.8	435.7	5.2	44.0	6.0
	8	45.1	0.9	35.4	413.1	53.1	45.7	17.0

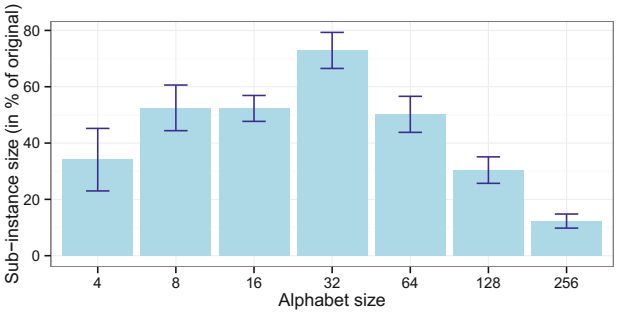
- Concerning SET1, both Beam-ACO and CMSA provide (near-)optimal solutions and both outperform CPLEX once the average optimality gaps start to increase. However, no clear trend about the superiority of CMSA over Beam-ACO (or the other way around) is noticeable.
- Concerning SET2, the performance of Beam-ACO and CMSA is comparable for instances of alphabet sizes $|\Sigma| \in \{4, 8, 16\}$, both providing (near-)optimal solutions. However, starting from alphabet size $|\Sigma| = 32$, CMSA outperforms Beam-ACO. This becomes even more clear in the case of the extension of SET2, consisting of larger problem instances. In the context of these instances, CMSA outperforms both CPLEX and Beam-ACO.

Table 4. Experimental results for larger problem instances.

$\|\Sigma\|$	n	Beam-ACO		CPLEX			CMSA	
		result	**time (s)**	**result**	**time (s)**	**gap (%)**	**result**	**time (s)**
128	3	38.3	< 0.1	38.4	50.1	0.0	38.4	0.1
	4	44.3	< 0.1	45.3	296.1	0.0	45.2	3.8
	5	52.6	1.8	23.3	85.8	> 100.0	53.7	1.3
	6	58.6	< 0.1	18.1	78.3	> 100.0	61.2	9.2
	7	66.3	5.3	n.a.	n.a.	n.a.	68.7	26.0
	8	73.7	6.9	n.a.	n.a.	n.a.	75.8	32.1
256	3	53.6	< 0.1	7.10	102.2	> 100.0	53.6	0.7
	4	66.8	0.9	0.10	143.2	> 100.0	67.0	12.1
	5	79.4	0.6	n.a.	n.a.	n.a.	81.0	7.6
	6	90.2	22.1	n.a.	n.a.	n.a.	92.1	35.0
	7	99.4	16.5	n.a.	n.a.	n.a.	102.2	47.7
	8	109.0	23.1	n.a.	n.a.	n.a.	111.3	62.7



(a) Sub-instance size for instances of Set1.



(b) Sub-instance size for instances of Set2and larger instances.

Fig. 1. Graphical presentation of the sizes of the sub-instances in percent with respect to the size of the original problem instances.

Finally, we studied the (average) size of the sub-instances that are generated (and maintained) within CMSA in comparison to the size of the original problem instances. These sub-instance sizes are provided in a graphical way in Fig. 1a for instances of SET1, and in Fig. 1b for instances of SET2 and its extension. Note that these graphics show the sub-instance sizes averaged over all instances of the same alphabet size. In both cases, the x-axis ranges from small alphabet size (left) to large alphabet sizes (right). Interestingly, when the alphabet size is rather small, the tackled sub-instances in CMSA are rather large (up to ≈70 % of the size of the original problem instances). With growing alphabet size, the size of the tackled sub-instances decreases. This is more clearly visible in the context of instances of SET1. However, this trend also becomes clear starting from alphabet size 32 in the context of instances of SET2. The reason for this trend is as follows. As CPLEX is very efficient for problem instances based on rather small alphabet sizes, the parameter values of CMSA are chosen during the tuning process of irace such that the sub-instance sizes become quite large. On the contrary, with growing alphabet size, the parameter values chosen during tuning lead to smaller sub-instances, simply because CPLEX is not so efficient anymore when applied to sub-instances that are not much smaller than the original problem instances.

5 Discussion and Future Work

CMSA is a new, general, algorithm for combinatorial optimization which is based on a simple, but apparently successful, idea: the generation of sub-instances based on merging the solution components found in randomly constructed solutions, and their subsequent solution by means of an exact solver. Moreover, the considered sub-instances are dynamically changing due to adding new solution components at each iteration, and removing existing solution components on the basis of indicators about their usefulness. In this work, the CMSA algorithm has been applied to the repetition-free longest common subsequence problem. The general picture of the results, in comparison to CPLEX, is similar to the one observed in earlier applications of CMSA to the minimum common string partition problem and a minimum weight arborescence problem in [1]. CMSA is generally competitive with CPLEX for small to medium size problem instances, whereas it outperforms CPLEX with growing problem instances size. In our opinion, this algorithm is quite appealing, especially for the following reasons:

- CMSA can be applied to any problem for which a constructive heuristic and an exact solver are known.
- In comparison to other metaheuristics, CMSA can generally be implemented with few lines of code.
- When using an ILP solver for solving sub-instances, CMSA allows to make use of the valuable operations research expertise that has gone into the development of the ILP solver, without the need of being an expert in operations research.

Finally, note that the idea behind CMSA is similar, in some sense, to the basic idea of large neighborhood search (LNS) [13]. However, while exact solvers in LNS are used to search the best solution in a large neighborhood of the current solution which is generally obtained by a partial destruction of the current solution, exact solvers in the context of CMSA are applied to sub-instances of the original problem instances.

Concerning future work, we first plan to extend the conducted experimental study to even larger problem instances. Second, we intent to study the incorporation of potentially valuable knowledge about, for example, the reduced costs of variables, in order to develop a more sophisticated—and hopefully more effective—mechanism for the removal of variables from the considered sub-instances.

References

1. Blum, C., Pinacho, P., López-Ibáñez, M., Lozano, J.A.: Construct, merge, solve and adapt a new general algorithm for combinatorial optimization. Comput. Oper. Res. **68**, 75–88 (2016)
2. Blum, C., Calvo, B.: A matheuristic for the minimum weight rooted arborescence problem. J. Heuristics **21**(4), 479–499 (2015)
3. Adi, S.S., Braga, M.D.V., Fernandes, C.G., Ferreira, C.E., Martinez, F.V., Sagot, M.F., Stefanes, M.A., Tjandraatmadja, C., Wakabayashi, Y.: Repetition-free longest common subsquence. Discrete Appl. Math. **158**, 1315–1324 (2010)
4. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
5. Smith, T., Waterman, M.: Identification of common molecular subsequences. J. Mol. Biol. **147**(1), 195–197 (1981)
6. Jiang, T., Lin, G., Ma, B., Zhang, K.: A general edit distance between RNA structures. J. Comput. Biol. **9**(2), 371–388 (2002)
7. Storer, J.: Data Compression: Methods and Theory. Computer Science Press, MD (1988)
8. Aho, A., Hopcroft, J., Ullman, J.: Data Structures and Algorithms. Addison-Wesley, Reading (1983)
9. Maier, D.: The complexity of some problems on subsequences and supersequences. J. ACM **25**, 322–336 (1978)
10. Blum, C., Blesa, M.J., Calvo, B.: Beam-ACO for the repetition-free longest common subsequence problem. In: Legrand, P., Corsini, M.-M., Hao, J.-K., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) EA 2013. LNCS, vol. 8752, pp. 79–90. Springer, Heidelberg (2014)
11. Castelli, M., Beretta, S., Vanneschi, L.: A hybrid genetic algorithm for the repetition free longest common subsequence problem. Oper. Res. Lett. **41**(6), 644–649 (2013)
12. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Technical report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
13. Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research and Management Science, vol. 146, pp. 399–419. Springer, US (2010)

Deconstructing the Big Valley Search Space Hypothesis

Gabriela Ochoa(✉) and Nadarajen Veerapen

Computing Science and Mathematics, University of Stirling, Stirling, Scotland, UK
goc@cs.stir.ac.uk

Abstract. The big valley hypothesis suggests that, in combinatorial optimisation, local optima of good quality are clustered and surround the global optimum. We show here that the idea of a single valley does not always hold. Instead the big valley seems to de-construct into several valleys, also called 'funnels' in theoretical chemistry. We use the local optima networks model and propose an effective procedure for extracting the network data. We conduct a detailed study on four selected TSP instances of moderate size and observe that the big valley decomposes into a number of sub-valleys of different sizes and fitness distributions. Sometimes the global optimum is located in the largest valley, which suggests an easy to search landscape, but this is not generally the case. The global optimum might be located in a small valley, which offers a clear and visual explanation of the increased search difficulty in these cases. Our study opens up new possibilities for analysing and visualising combinatorial landscapes as complex networks.

Keywords: Fitness landscapes · Local optima networks · Funnels · Traveling salesman problem

1 Introduction

In the mid 1990s, it was conjectured that the search space of travelling salesman instances had a "globally convex" or "big valley" structure, in which local optima are clustered around one central global optimum [3]. This globally convex structure has subsequently been observed in other combinatorial problems such as the NK family of landscapes [7,8], graph bipartitioning [13], and flowshop scheduling [21]. Under this view, there are many local optima but they are easy to escape from, with the coarse level gradient leading to the global optimum (see Fig. 1). This hypothesis has become generally accepted and has inspired the design of modern search heuristics.

We argue that this view of combinatorial search spaces is not complete. We challenge the existence of a single valley, and present compelling and visual evidence of examples where the big valley de-constructs into several valleys, also called 'funnels' in the study of energy surfaces in theoretical chemistry [9,14]. The multi-funnel concept implies that local optima are organised into clusters, so that a particular local optimum largely belongs to a particular cluster.

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 58–73, 2016.
DOI: 10.1007/978-3-319-30698-8_5

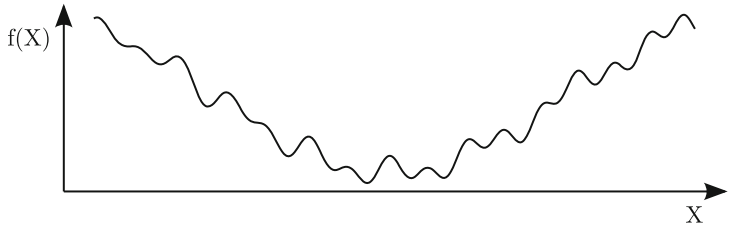


Fig. 1. Depiction of the 'big-valley' structure.

Using the travelling salesman problem (TSP) as a case study, we found that this decomposition into clusters does not only occur near the global optimum as has been observed recently [6,19]. It occurs earlier on in the search process, even among local optima tours with relatively high costs. This finding improves our understanding of search difficulty in combinatorial optimisation. It explains why, when using current local search heuristics, random restarts are generally required to consistently find globally optimal solutions. When trapped in a sub-optimal funnel, a local search heuristic will not be able to escape even with relatively large random perturbations. This insight will foster research into more informed escaping and *tunnelling* mechanisms [6,17,24].

We use the local optima networks model to analyse and visualise the big valley deconstruction. Local optima networks compress the whole search space into a graph, where nodes are local optima and edges are transitions among them with a given search operator [18,20,23]. Local optima are key features of fitness landscapes as they can be seen as obstacles for reaching high quality solutions. The local optima networks model emphasises the number, distribution and most importantly, the connectivity pattern of local optima in the underlying search space. They are therefore an ideal tool for modelling and visualising the big valley structure.

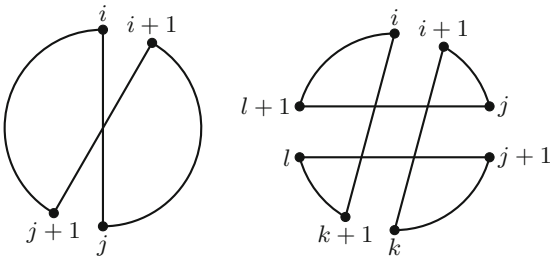
We propose a new and effective sampling procedure for extracting the network data of large instances. Local optima and escape edges are collected from several runs of *Chained Lin-Kerninghan*, a state-of the-art TSP heuristic [12]. This data is gathered to construct the local optima networks.

The remainder of this article is organised as follows. The next section gives an overview of Chained Lin-Kerninghan. Section 3 defines the local optima network model considered, and describes the procedure for extracting the data and constructing the networks. Section 4 describes the TSP instances studied. Section 5 presents the analysis and visualisation of the extracted local optima networks. Finally, Sect. 6 summarises our main findings and suggests directions of future work.

2 Chained Lin-Kernighan

Lin-Kernighan (LK) [10], is a powerful and well-known heuristic for solving the TSP. For about two decades, it was the best local search method, and nowadays it

is a key component of state-of-the-art TSP solvers. LK search is based on the idea of k-changes: take the current tour and remove k different links from it, which are then reconnected in a new way to achieve a legal tour. A tour is considered to be 'k-opt' if no k-change exists which decreases its length. Figure 2a illustrates a 2-change move. LK applies 2, 3 and higher-order k-changes. The order of a change is not predetermined, rather k is increased until a stopping criterion is met. Thus many kinds of k-changes and all 3-changes are included. There are many ways to choose the stopping criteria and the best implementations are rather involved. Here, we use the implementation available in the Concorde software package [1], which uses *do not look* bits and candidate lists.



(a) 2-change (b) Double-bridge

Fig. 2. Illustration of tours obtained after 2-change and double-bridge moves.

The overall tour-finding strategy using LK-search was to repeatedly start the basic LK routine from different starting points keeping the best solution found. This practice ended in the 1990 s with the seminal work of Martin, Otto and Felten [12], who proposed the alternative of *kicking* (perturbing slightly) the LK tour and reapplying the algorithm. If a better tour is produced, we discard the old LK tour and keep the new one. Otherwise, we continue with the old tour and kick it gain. This simple yet powerful strategy is nowadays best known as *iterated local search* [11]. It was named *Chained Lin-Kernighan* (Chained LK) by Applegate et al. [2], who also provided an improved implementation to solve large TSP instances.

The kick or escape operator in Chained-LK is a type of 4-change, named *double-bridge* by Martin et al. [12] (drawn in Fig. 2b). It consists of two improper 2-changes, each of which is a 'bridge' as it takes a legal, connected tour into two disconnected parts. The combination of both bridges, must then be chosen in order to produce a legal final tour.

3 Local Optima Networks for TSP

To construct the networks we need to define their nodes and edges. The definition is closely linked to the methodology for extracting the network data, which

is based on a number of runs of the Chained-LK algorithm described above. Clearly, a full enumeration of the local optima for TSP instances of non-trivial size becomes unmanageable. Therefore, the networks are based on a sample of high-quality local optima in the search space. We first provide some basic definitions, below, before describing the sampling algorithm.

3.1 Definitions

Definition 1. A tour is a *local optimum* if no tour in its neighbourhood is shorter than it. The neighbourhood is imposed by LK-search, which considers variable values of k. The local optimality criterion is, therefore, rather stringent. Only a small proportion of all possible tours are LK-optimum. The set of local optima is denoted by LO.

Definition 2. Edges are directed and based on the double-bridge operator. There is an *escape* edge from local optimum LO_i to local optimum LO_j, if LO_j can be obtained after applying a double-bridge kick to LO_i followed by LK-Search. The set of escape edges is denoted by E_{esc}.

Definition 3. The *local optima network, LON*, is the graph $LON = (LO, E_{esc})$ where nodes are the local optima LO, and edges E_{esc} are the escape edges.

3.2 Gathering Network Data

To extract the network data, we instrumented the Chained-LK implementation of Concorde (see Algorithm 1). We simply store, in LO, every unique local optima obtained after an LK application, and create and store, in E_{esc}, an edge between the starting and end optima after a double-bridge move.

A hundred independent runs of Chained-LK are executed. We chose to use two different starting mechanisms, one producing "better" solutions, the other "worse" solutions, to have a broader picture of the search space. Half of the runs start from a relatively good solution, built using the Quick-Borůvka method. The latter is the default initialisation for Concorde's Chained-LK and is based on the minimum-weight spanning tree algorithm of Borůvka [15]. The other half starts from a random solution.

Each run performs n kicks, where n is the size of the tour (number of cities). The default kicking procedure in Concorde is used: the edges involved in the double bridge are selected using random walks along connected vertices.

Since nodes and edges are collected from a combination of several runs, it is possible that each of them is found more than once. Therefore, weights could be associated to edges indicating the number of times they were encountered. We recorded such weights, but chose to analyse unweighted networks. Future work will consider this information in the analysis.

Data: I, a TSP instance
Result: LO, the set of local optima,
E_{esc}, the set of edges between local optima
$n \leftarrow$ numberOfCities(I); $LO \leftarrow \{\}$; $E_{esc} \leftarrow \{\}$
for $i \leftarrow 1$ **to** 100 **do**
  $s \leftarrow$ initialSolution()
  $s \leftarrow$ LK(s)
  $LO \leftarrow LO \cup \{s\}$
  for $k \leftarrow 1$ **to** n **do**
    $s_{start} \leftarrow s$
    $s_{end} \leftarrow$ applyKick(s)
    $s_{end} \leftarrow$ LK(s_{end})
    $LO \leftarrow LO \cup \{s_{end}\}$
    $E_{esc} \leftarrow E_{esc} \cup \{(s_{start}, s_{end})\}$
    if *fitness*(s_{end}) < *fitness*(s_{start}) **then** $s \leftarrow s_{end}$
  end
end

Algorithm 1. Local optima network sampling for 100 runs of Chained-LK.

4 Selected TSP Instances

Our study considers four TSPLIB [22] instances of a few hundred cities belonging to different types (see Table 1). By exploring and comparing the local optima networks of instances of similar size, we aim to discover structural differences distinguishing the hard from the easy to solve instances.

Table 1. Selected TSP instances

Property	Instances			
	att532	u574	rat575	gr666
Cities	532	574	575	666
Edge Weight Type	ATT	EUC-2D	EUC-2D	GEO
Description	US cities	Drilling problem	Rattled grid	World cities
Optimum	27686	36905	6773	294358
Concorde run time (s)	8.9	3.8	18.9	6.5
Concorde B&B nodes	5.2	1.7	17.7	3.2
Chained-LK success rate	0.06	0.47	0.01	0.04

The types of edge weights are as follows. EUC-2D refers to the Euclidean distance of points in a 2D plane rounded to the nearest integer. ATT refers to a pseudo-Euclidean distance: the sum of the squares is divided by 10 and the square root of this value is then rounded to an integer. GEO refers to the integer geographical distance computed from latitude and longitude coordinates on the surface of a sphere representing an idealised Earth.

The bottom portion of Table 1 gives information on the solving difficulty of each instance. Specifically, we report the mean run time and the mean number of branch-and-bound nodes required by Concorde (interfaced with IBM ILOG CPLEX 12.6) to solve the instances to optimality on a 3.4 GHz Intel Core i7-3770 CPU across 10 runs. Although Concorde is an exact solver, the means are computed since randomised heuristics, including Chained-LK, are used. This leads to different execution times and branch-and-bound trees. The last row reports the success rate of the 100 Chained-LK runs used for extracting the network data (described in Sect. 3.2). By success rate, we mean the ratio of runs that found at least one global optimum. According to this information, the easiest instance to solve is u574 (by far) and the hardest is rat575.

5 Results

When extracting local optima networks from large instances, it is important to decide which sample to consider. We chose here to analyse two sets: (i) the whole set of local optima collected with the procedure described in Sect. 3.2, and (ii) the subset containing the best 10 % local optima according to fitness. For each TSP instance in Table 1, we consider the two sets and construct the local optima networks as defined in Sect. 3.

Results are presented in the following two subsections, which conduct a network analysis and a fitness distance correlation analysis, respectively.

5.1 Network Analysis and Visualisation

Over the years, an extensive set of tools – mathematical, computational, and statistical – have been developed for analysing and understanding networks [16]. We select here a set of network features (see Table 2) which we consider relevant to search dynamics.

Table 2. Main local optima network features.

Feature	Description
nv	Total umber of vertices (local optima)
ne	Total number of edges
n_{go}	Number of different global optima
nc	Number of connected components (or clusters)
c_{go}	Cluster containing the global optimum (or optima), where the clusters are ordered by decreasing size.

We argue that the decomposition into clusters (connected components in our study) is one of the most relevant features impacting search. Indeed, we hypothesise that the notion of multiple funnels, originally studied in theoretical

chemistry [9,14], and more recently also in combinatorial optimisation [6,17], is captured by the connected components in the networks studied. Specifically, funnels correspond to connected components. Once trapped in a connected component, it is not easy for the search process to hop to another component. There are no connections among components with the underlying escaping mechanism according to our sampling procedure. Some connections may exist, but of low probability. We, therefore, explore in detail the connected components decomposition of the studied networks.

Table 3 reports the main network features for each instance and local optima sample. All instances have more than one global optima, and all decompose into several clusters. Indeed, several components are found on both samples, indicating that the deconstruction occurs not only among solutions near the global optima, but early on in the search process (solutions with higher costs). The last row in the table (c_{go}) shows that global optima are not always found in a large connected component.

Table 3. Network metrics (as described in Table 2) for the four TSP instances and the two local optima samples: *all* and *best 10 %*.

	att532		u574		rat575		gr666	
	all	*best 10 %*	*all*	*best 10 %*	*all*	*best 10 %*	*all*	*best 10 %*
nv	35,512	3,678	37,780	3,842	41,536	4,805	46,021	4,611
ne	37,730	4,435	40,161	4,660	44,643	5,842	47,939	5,039
n_{go}	2	2	4	4	2	2	2	2
nc	6	7	8	5	69	47	53	25
c_{go}	1	2	1	1	60	8	4	2

It is interesting to note that for the hardest instance studied, rat575 (see the bottom of Table 1 for an indication of search difficulty), the global optima were not found in any of the 5 largest connected components. They are located in cluster number 60 when considering the whole sample and cluster 8, when considering the best 10 % local optima. On the other hand, for the easiest instance, u574, the global optima are found in the largest connected component for both samples. Table 4 reports sizes (as percentages) of the 5 largest connected components for each instance and local optima sample. Bold fonts indicate the component containing the global optima. As mentioned before, global optima are not found in the top 5 connected components of instance rat575, they are located in the 8^{th} component, which contains only 2.67 % of the local optima sample.

A useful approach to explore the structure of networks is to visualise them. Software for analysing and visualising networks is currently available in various languages and environments. Here we use the R statistical language together with the igraph package [4]. The graph layout algorithm used is the Fruchterman and Reingold method [5], which is based on exploiting analogies between the

Table 4. Sizes (as percentages) of the top 5 connected components for the four TSP instances and the two local optima samples: *all* and *best* 10 %. Bold fonts highlight the connected component containing the global optima. For instance rat575, the global optima are located in the 8^{th} component, which contains only 2.67 % of the local optima sample

	att532		u574		rat575		gr666	
	all	*best 10 %*	*all*	*best 10 %*	*all*	*best 10 %*	*all*	*best 10 %*
c_1	**93.62**	50.33	**58.56**	**85.84**	17.55	33.47	17.20	29.75
c_2	2.17	**48.15**	15.97	11.63	4.93	7.62	7.12	**13.42**
c_3	1.12	1.20	10.30	1.54	4.89	3.79	7.05	12.67
c_4	1.04	0.19	8.72	0.60	2.03	3.33	**6.22**	6.66
c_5	1.04	0.08	3.20	0.39	2.01	2.93	3.90	4.49

relational structure in graphs and the forces among elements in physical systems. The heuristic is concerned with drawing graphs according to some generally accepted aesthetic criteria such as (a) distribute the vertices evenly in the frame (a circle in this case), (b) minimise edge crossings, (c) make edge lengths uniform, and (d) reflect inherent symmetry [5].

In order to have manageable images, we plotted the networks corresponding to the subset containing the best 10 % local optima. We also pruned some of the nodes of degree one, and removed self-loops for improved visibility. Figure 3 shows the local optima for instances att532 and u574, and Fig. 4 shows the networks for rat575 and gr666. Nodes are LK-search local optima and edges represent escape transitions according to double-bridge moves.

We decorated the network images according to the two most relevant features impacting search dynamics: fitness and connectivity. The fitness of a solution is reflected by its node size, with size inversely proportional to tour cost (so the best solutions are larger in size). The connected components are distinguished with different colours: red shows the largest connected component, blue the 2^{nd} largest and so on, as indicated in the legends of Figs. 3 and 4. Global optima nodes are highlighted with a yellow outline.

The networks show strikingly different structures. In instance att532 (Fig. 3, top), the two largest connected components (red and blue) show similar sizes, with the remaining components having small sizes (see also Table 4 for percentages). The two global optima are located in the blue component. In instance u574 (Fig. 3, top), the largest component (red) clearly dominates, containing the four global optima and many good local optima as indicated by the node sizes. This is consistent with the fact that u574 is the easiest instance to solve, as indicated in the bottom of Table 1.

In the first two instances considered (Fig. 3), the two largest components (red and blue) dominate the network images. This is not the case for instances rat575 and gr666 (Fig. 4), where the smaller connected components are more visible. In rat575, the global optima are not found in the top 5 components. Instead, the

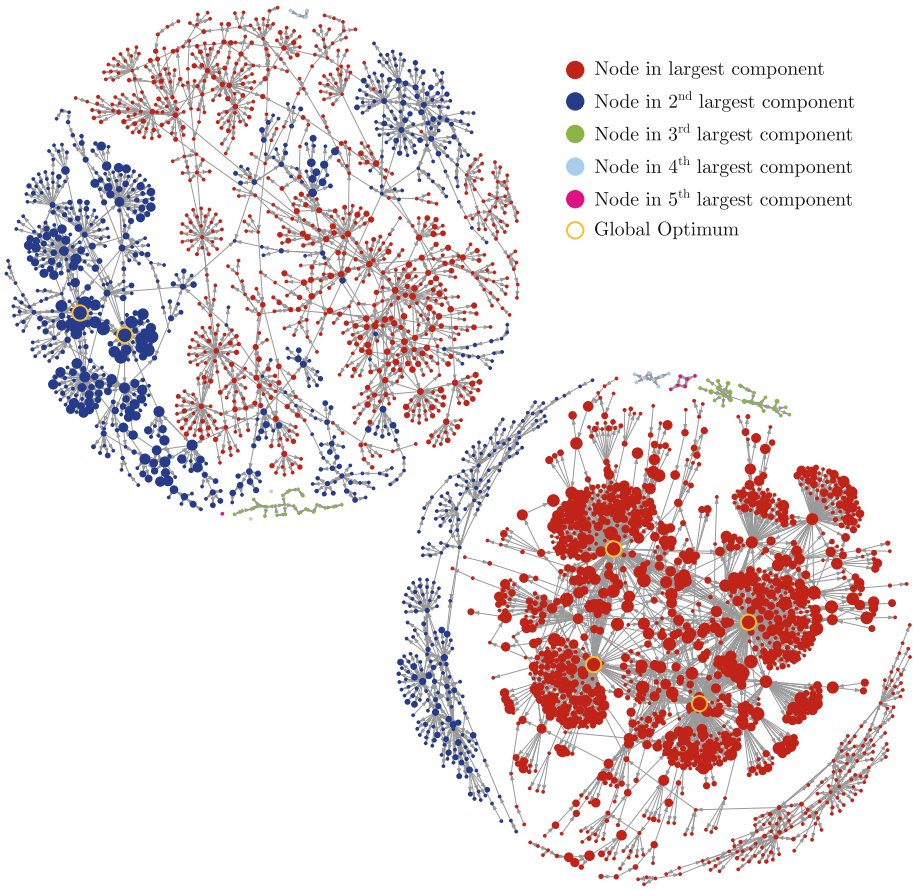


Fig. 3. Local optima networks for att532 (top) and u574 (bottom). Nodes are LK-search local optima, and edges represent escape transitions according to double-bridge moves. Node colours identify connected components as indicated in the legend, while node sizes are inversely proportional to tour cost (so the best solutions are larger in size). Global optima nodes are highlighted with a yellow outline (Color figure online).

two global optima are located in component number 8, visualised in dark green at the bottom right of the network plot. This is a small component containing only 2.67 % of the local optima sample: this provides a clear visual indication of the increased search difficulty of this instance. For the gr666 instance, the global optima are located in the 2^{nd} largest connected component (blue) visualised at the bottom left portion of the image. This component contains 13.42 % of the local optima, suggesting an easier to search instance despite having a larger number of cities.

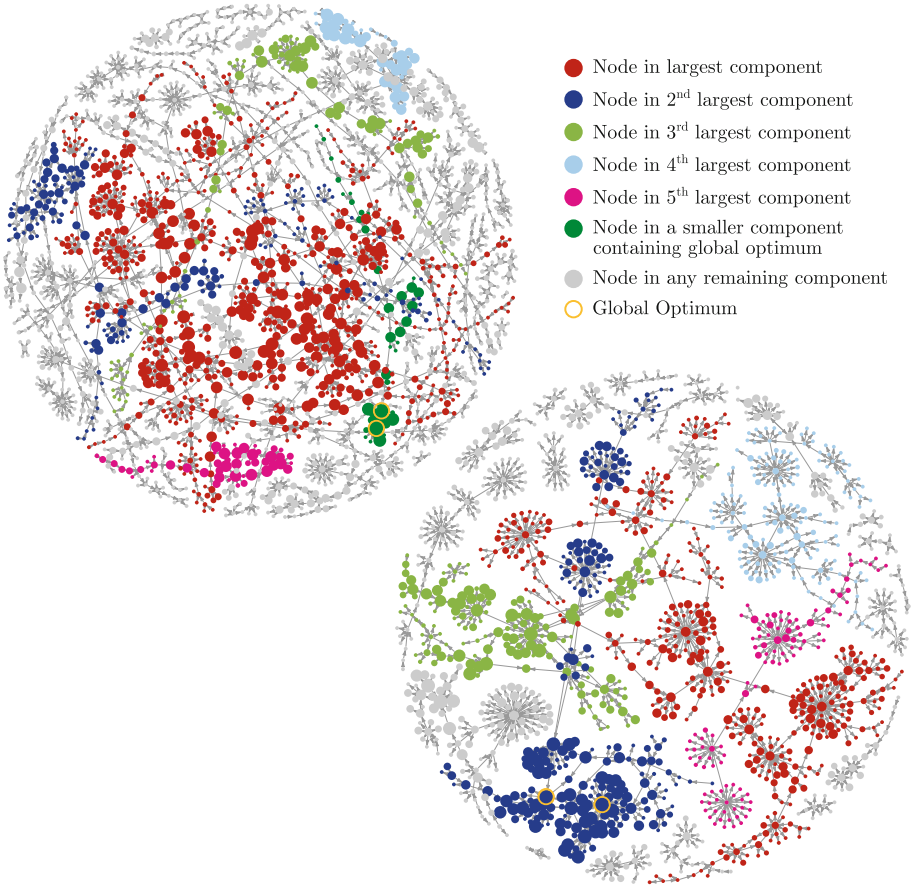


Fig. 4. Local optima networks for rat575 (top) and gr666 (bottom). Nodes are LK-search local optima, and edges represent escape transitions according to double-bridge moves. Node colours identify connected components as indicated in the legend, while node sizes are inversely proportional to tour cost. Global optima nodes are highlighted with a yellow outline (Color figure online).

This study only considers instances where the number of connected components was less than the number of runs. Yet, the maximum number of components that could be discovered by the sampling method corresponds to the number of runs if no local optimum is repeated in any two runs. It is nevertheless possible that some instances actually have many more components than this number. It is also important to note that the sampling mechanism, including the parameter values for the number of kicks and runs, introduces a bias generating an approximation of the search space and not the complete picture.

5.2 Fitness-Distance Analysis

While the network analyses provide insight into the connected nature of the search space, it is also useful to examine the landscapes through more traditional tools. In particular, we now look at the relationship between fitness and bond distance [3]. The latter is defined as the difference in the number of common edges, or bonds, between two tours. It is computed by subtracting the number of common edges from the number of cities. We specifically consider the distance between a single randomly chosen global optimum and the other local optima. Let us note that the global optima for each instance share the overwhelming majority of their edges. The bond distances between the global optima are 2 for att532, $\{2, 4, 6\}$ for u574, 3 for rat575 and 13 for gr666. It is thus logical for them to appear within the same component.

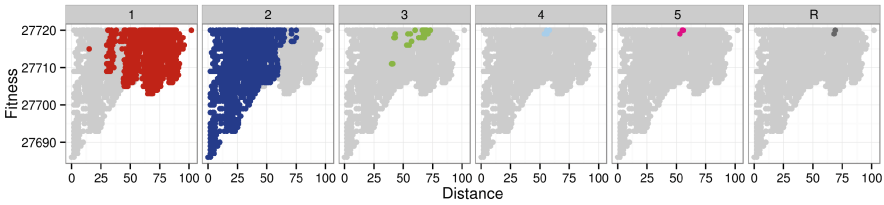
Figure 5 presents the fitness-distance plots for the *best 10 %* sub-sampling that is represented in the local optima networks and reuses the same colour-component correspondence. Each of the 5 largest components is plotted in a separate facet. When the global optima are found in a smaller component, the points of the latter have their own facet. Any remaining components are grouped in one final facet. Each plot also displays all the local optima across components in the background.

Figure 6 provides a similar view of the local optima but considers (almost) complete samples. Points with fitness above the 95^{th} percentile are not plotted because they are very spread out and thus interfere with visualisation. Components, however, are computed with respect to all the points in the *all* sample. Points in common with those in Fig. 5 are highlighted with the same colour scheme, with the aim of exploring the correspondence of clusters between the two studied samples.

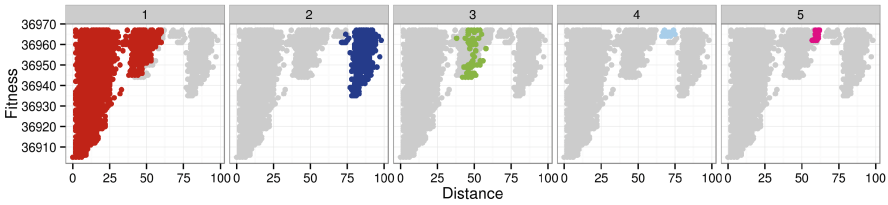
In the *best 10 %* sampling, smaller components containing a few solutions are artefacts of the arbitrary threshold and actually form part of larger distinct clusters. For att532, even though this is not visible due to points overlapping, all the *best 10 %* components are indeed at the bottom of a single massive component in the *all* sampling.

We can observe that there is relatively little overlap between components when the solution fitness is close to the best fitness (Fig. 5). This no longer the case when the *all* sampling is observed. The components not containing the global optima in instances att532 and u674, and to a lesser extent in gr666, are relatively far away from the one with the global optima both in terms of fitness and bond distance. In contrast, the best fitness values in the components of rat575 are all within 4 units of the global optimum. Let us note that, for this instance, the distribution of points appears to consist of distinct layers. This is simply because the range of fitness values is very small and all values are integers.

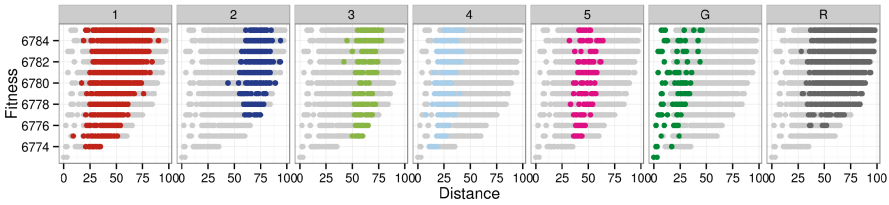
From Fig. 6, it can be seen that the presence of distinct components does not match the big valley hypothesis, but rather that there are multiple distinct funnels. On some instances, looking at the bottom of these components, or funnels, reveals further splits into basins within funnels (Fig. 5).



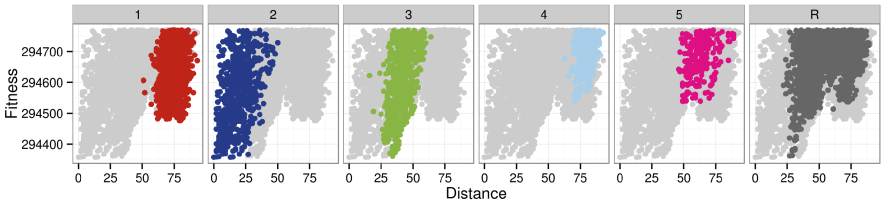
(a) att532



(b) u574

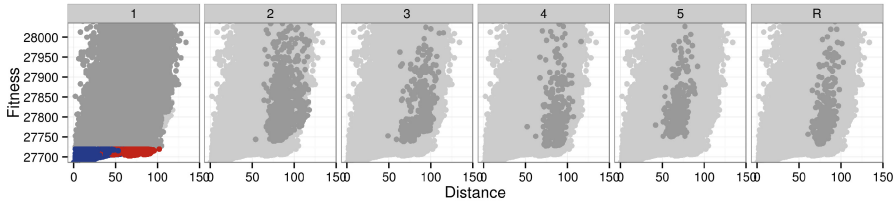


(c) rat575

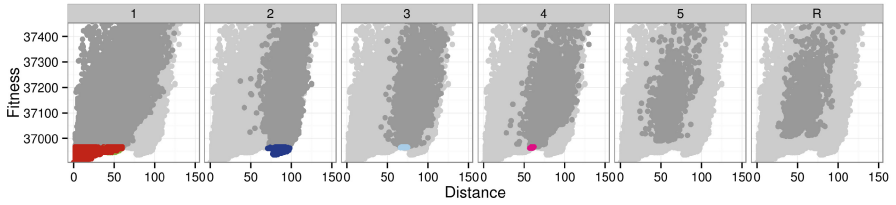


(d) gr666

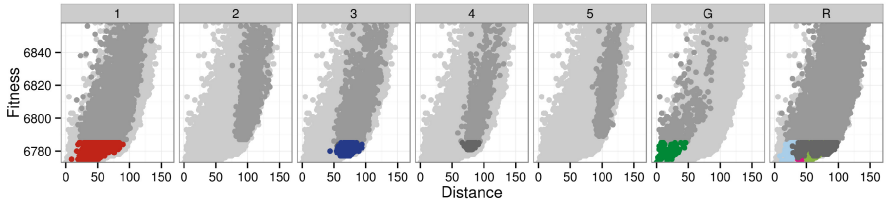
Fig. 5. Fitness-distance plots for the *best* 10 % sampling. All facets show the full set of solutions of the sampling in the background. Facets 1 to 5 display the overlay of the largest five connected components. Facet G shows the component containing the global optima (when it is not among the first five, as in the case of rat575). Facet R displays the remaining components if there are any.



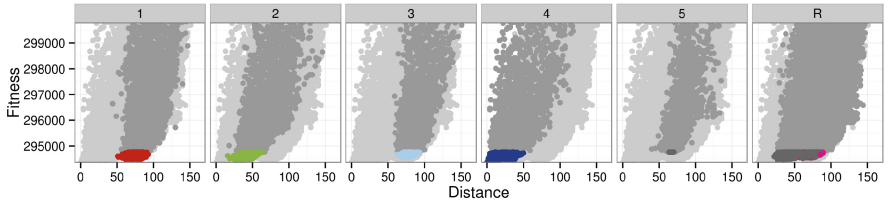
(a) att532



(b) u574



(c) rat575



(d) gr666

Fig. 6. Fitness distance plots for the *all* sampling. In the background (light grey), all facets show the set of sampled solutions below the 95^{th} fitness percentile. Points within specific components are coloured in darker grey in each facet. Points in common with Figure 5 use the same colours. Facets 1 to 5 display the overlay of the largest five connected components. Facet G shows the component containing the global optima (when it is not among the first 5, such as in rat575). Facet R displays the remaining components.

6 Conclusions

Our study suggest that there is not always a single valley in the fitness landscape of travelling salesman problems under LK-search and double-bridge escape moves. Instead, local optima might decompose into a number of sub-valleys or funnels, as illustrated in Fig. 7 for two funnels, but more than two are generally present. This decomposition occurs not only among solutions near in evaluation to the global optimum, but it may also happen among solutions with higher cost. In our local optima network model, the funnels are clearly identified and visualised as the connected components of the networks.

This has significant consequences in our understanding of iterated local search. Once the search process is trapped in a sub-optimal funnel, it simply cannot escape from it using the underlying escaping mechanism (double-bridge moves in our study). Increasing the number of iterations will not improve the performance, the search will stall, as transitions to other funnels are not possible. We foresee that this observation will inspire new escaping and tunnelling mechanisms that allow the search process to navigate among funnels.

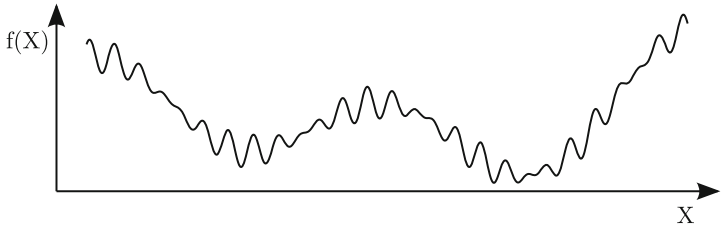


Fig. 7. Depiction of two funnels.

Future work will study the structure of larger, and more diverse TSP instances and other combinatorial problems where the big valley has been observed. More extensive sampling methods will need to be considered to confirm or infirm our results. We will also look at search strategies to escape from sub-optimal funnels. We also aim to produce improved and informative images of fitness landscapes using the local optima network model.

Acknowledgements. Thanks are due to Darrell Whitley for relevant discussions and suggesting the paper's title. This work was supported by the UK's Engineering and Physical Sciences Research Council [grant number EP/J017515/1].

Data Access. All data generated during this research are openly available from the Stirling Online Repository for Research Data (http://hdl.handle.net/11667/71). All data generated during this research are openly available from the Stirling Online Repository for Research Data (http://hdl.handle.net/11667/71).

References

1. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Concorde TSP solver (2003). http://www.math.uwaterloo.ca/tsp/concorde.html
2. Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problems. INFORMS J. Comput. **15**, 82–92 (2003)
3. Boese, K.D., Kahng, A.B., Muddu, S.: A new adaptive multi-start technique for combinatorial global optimizations. Oper. Res. Lett. **16**, 101–113 (1994)
4. Csardi, G., Nepusz, T.: The igraph software package for complex network research. Int. J. Complex Syst. **1695**, 1–9 (2006)
5. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exper. **21**(11), 1129–1164 (1991)
6. Hains, D.R., Whitley, L.D., Howe, A.E.: Revisiting the big valley search space structure in the TSP. J. Oper. Res. Soc. **62**(2), 305–312 (2011)
7. Kauffman, S., Levin, S.: Towards a general theory of adaptive walks on rugged landscapes. J. Theor. Biol. **128**, 11–45 (1987)
8. Kauffman, S.A.: The Origins of Order. Oxford University Press, New York (1993)
9. Klemm, K., Flamm, C., Stadler, P.F.: Funnels in energy landscapes. Eur. Phys. J. B **63**(3), 387–391 (2008)
10. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**, 498–516 (1973)
11. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Handbook of Metaheuristics, pp. 320–353 (2003)
12. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the TSP incorporating local search heuristics. Oper. Res. Lett. **11**, 219–224 (1992)
13. 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, p. 765. Springer, Heidelberg (1998)
14. Miller, M.A., Wales, D.J.: The double-funnel energy landscape of the 38-atom Lennard-Jones cluster. J. Chem. Phys. **110**(14), 6896–6906 (1999)
15. Nešetřil, J., Milková, E., Nešetřilová, H.: Otakar Borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. Discrete Math. **233**(1–3), 3–36 (2001)
16. Newman, M.E.J.: Networks: An Introduction. Oxford University Press, Oxford (2010)
17. Ochoa, G., Chicano, F., Tinos, R., Whitley, D.: Tunnelling crossover networks. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 449–456. ACM (2015)
18. Ochoa, G., Tomassini, M., Verel, S., Darabos, C.: A study of NK landscapes' basins and local optima networks. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). pp. 555–562. ACM (2008)
19. Ochoa, G., Veerapen, N., Whitley, D., Burke, E.: The multi-funnel structure of TSP fitness landscapes: a visual exploration. In: EA 2015. LNCS. Springer (2015, to appear)
20. Ochoa, G., Verel, S., Daolio, F., Tomassini, M.: Local optima networks: a new model of combinatorial fitness landscapes. In: Richter, H., Engelbrecht, A. (eds.) Recent Advances in the Theory and Application of Fitness Landscapes. ECC, vol. 6, pp. 245–276. Springer, Heidelberg (2014)
21. Reeves, C.R.: Landscapes, operators and heuristic search. Ann. Oper. Res. **86**, 473–490 (1999)

22. Reinelt, G.: TSPLIB - A traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991). http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
23. Verel, S., Ochoa, G., Tomassini, M.: Local optima networks of NK landscapes with neutrality. IEEE Trans. Evol. Comput. **15**(6), 783–797 (2011)
24. Whitley, D., Hains, D., Howe, A.: Tunneling between optima: partition crossover for the traveling salesman problem. In: Proceedings Genetic and Evolutionary Computation Conference, GECCO 2009, pp. 915–922. ACM, New York (2009)

Determining the Difficulty of Landscapes by PageRank Centrality in Local Optima Networks

Sebastian Herrmann(✉)

Department of Information Systems and Business Administration,
Johannes Gutenberg-Universität, Jakob-Welder-Weg 9, 55128 Mainz, Germany
s.herrmann@uni-mainz.de
http://wi.bwl.uni-mainz.de

Abstract. The contribution of this study is twofold: First, we show that we can predict the performance of Iterated Local Search (ILS) in different landscapes with the help of Local Optima Networks (LONs) with escape edges. As a predictor, we use the PageRank Centrality of the global optimum. Escape edges can be extracted with lower effort than the edges used in a previous study. Second, we show that the PageRank vector of a LON can be used to predict the solution quality (average fitness) achievable by ILS in different landscapes.

Keywords: Fitness landscape analysis · Search difficulty · PageRank centrality · Local optima networks · NK-landscapes

1 Introduction

Local Optima Networks [1] are a novel approach to study the structure of optimization problems by using complex network analysis. A Local Optima Network (LON) is a compressed representation of a combinatorial fitness landscape. Mathematically, a LON is a graph in which the vertices are the search space's local optima. The edges are modeled to reflect the transitions between the local optima and are weighted by transition probabilities. Three types of LON models have been introduced so far in order to represent different search operators: edges with basin transition probabilities for the trajectory of Hill Climbing algorithms [2], escape edges for Iterated Local Search [3] and LONs for Partition Crossover [4].

An application of fitness landscape analysis is to predict the performance of a search algorithm in a particular problem instance (search difficulty) [5,6]. Network features of LONs can in particular capture the search difficulty of landscapes [7]. Among the different network metrics, it was shown that the shortest path to the global optimum [8] and its PageRank Centrality [9] are good predictors for local search-based methods. The PageRank of the global optimum

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 74–87, 2016.
DOI: 10.1007/978-3-319-30698-8_6

predicts the empirical success rate of Hill Climbing with approx. 90 % accuracy. Success rate is the probability that a search algorithm hits the global optimum. The explanation for this high correlation is that LONs are an approximate Markov Chain representation of fitness landscapes. As the PageRank vector of the nodes represents their stationary distribution in the stochastic process, the PageRank of the global optimum approximates the probability that a search algorithm finds it.

A major limitation of previous studies is that success rate is just one alternative of measuring search performance. Most heuristics are—in the first place—not designed to solve a problem exactly. Instead, they make use of an existing trade-off between computation time and solution quality, with the goal to generate a yet sub-optimal, but acceptable solution. Thus, predicting the expected solution quality is a relevant issue. Another limitation of the PageRank study [9] is that predicting success rates has only been tested for the LON model with basin transition probabilities. However, the extraction procedure for this model is computationally expensive.

In this paper, we take up previous efforts on determining search difficulty of fitness landscapes with Local Optima Networks. We present a method to predict both success rate and solution quality (average fitness) with LONs with escape edges (LON_{ee}, [3]) using PageRank Centrality. The escape edges can be extracted with much lower computational effort than the basin transition probabilities. To predict the average fitness, we make use of the fact that the calculation of the PageRank results in a vector which covers the stationary distribution of the whole search space. We combine these probabilities with the distribution of fitness in the search space to calculate an expected value for the fitness. Using this value, we can predict the fitness that is achieved on average by Iterated Local Search in different NK landscapes.

Our paper is structured as follows[1]: In Sect. 2, we describe the search heuristic of which we aimed to predict its performance in our experiments, i.e. Iterated Local Search. In Sect. 3, we give a short introduction to fitness landscapes and provide a formal definition. In Sect. 4, we define LONs with escape edges. In Sect. 5, we shortly describe the concept of PageRank Centrality. In Sect. 6, we describe our experimental design and the search space used (NK family of landscapes). We present our results in Sect. 7 and draw our conclusions in Sect. 8.

2 Iterated Local Search

Iterated Local Search (ILS, [10]) has so far been used in a variety of studies on local optima networks [3,8,11]. The concept of ILS is used in many practically relevant search methods, e.g. the Chained Lin Kernighan heuristic [12,13]. This section gives a brief review on the algorithm.

[1] For the reader's convenience, we wanted this paper to be self-contained. In the introductory sections, we included descriptions and formal definitions for Fitness Landscapes and PageRank following the explanations in [9].

ILS combines the concept of intensification by local search with a number of perturbation steps to obtain some diversification. During intensification, heuristics focus their search on promising areas of the search space, whereas during diversification, new areas are explored [14]. Algorithm 1 describes the search method in pseudo code. Given a search space of valid solutions S for an optimization problem, we assign a fitness value to each $s \in S$ by the function $f : S \rightarrow \mathbb{R}_{\geq 0}$. ILS starts with a randomly selected solution $s_0 \in S$. Then, the algorithm performs a hill climbing procedure with best improvement as selection rule (Algorithm 2): from the neighborhood $N(s)$, the best solution with higher fitness is selected. This requires a scan of the whole neighborhood of s. The neighborhood $N(s)$ is the set of solutions that can be reached by performing an incremental change to s. This hill climbing procedure is then repeated until it reaches a local optimum s^*, i.e. no further improvement is possible. Then, ILS performs a diversification step by applying a limited perturbation to the local optimum, resulting in s'. As a next step, hill climbing is applied from s', until the next local optimum $s^{*'}$ is reached. If the new local optimum $s^{*'}$ is different from the previous s^* and has higher fitness, the algorithm has "escaped" to a new local optimum, and the change is accepted. Otherwise, another perturbation is applied to s^*. This procedure is repeated until a termination condition is met, e.g. a fixed number of escapes without any further improvement.

Algorithm 1. Iterated Local Search (ILS)

Require: Solution space S, Fitness function $f(S)$, Neighborhood function $N(S)$, Stopping Threshold t

1: Choose initial random solution $s_0 \in S$
2: $s^* \leftarrow$ hillClimbBI(s_0)
3: $i \leftarrow 0$
4: **repeat**
5: $s' \leftarrow$ perturbation(s^*)
6: $s*' \leftarrow$ hillClimbBI(s')
7: **if** $(s^{*'} \neq s^*) \wedge (f(s^{*'}) > f(s^*))$ **then**
8: $s^* \leftarrow s^{*'}$
9: $i \leftarrow 0$
10: **end if**
11: $i \leftarrow i + 1$
12: **until** i $\geq$ t
13: **return** s^*

3 Fitness Landscape Analysis

3.1 Concept

The notion of fitness landscapes originated from evolutionary biology [15]. The idea is that there is a fitness for each genome of the different species, and by the

Algorithm 2. Best Improvement Hill Climbing (hillClimbBI)

Require: Solution space S, Fitness function $f(S)$, Neighborhood function $N(S)$, Initial solution s_0

```
i ← 0
repeat
    choose s_{i+1} s.t. f(s_{i+1}) = max_{x∈N(s_i)}(f(x))
    if f(s_i) < f(s_{i+1}) then
        s_i ← s_{i+1}
    end if
    i ← i + 1
until s_i is local optimum: {s ∈ N(s_i) | f(s) < f(s_i)} = {}
return s_i
```

distances between the genomes a landscape is shaped in which the fitness is the height. In combinatorial optimization, a motivation to analyze fitness landscapes is to gain a better understanding of algorithm performance on a related set of problem instances. Landscape characteristics reflect the difficulty for a variety of heuristics [5,6], thus problem specific knowledge can help construct better search methods [16]. In this chapter, we provide a short explanation of important fundamentals of fitness landscape analysis.

3.2 Neighborhood Structure

In combinatorial optimization, a fitness landscape [15] is a triplet of the search space S, the fitness function f, and the neighborhood structure $N(S)$. S contains all valid solution candidates. The fitness function $f : S \to \mathbb{R}_{\geq 0}$ assigns a fitness value to each $s \in S$. The neighborhood function $N : S \to \mathcal{P}(S)$ assigns a set of neighbors $N(s)$ to every $s \in S$. The neighborhood structure determines the position of each s in the landscape [17]. To determine the neighbors, we define a distance function between all pairs of solutions s_0 and s_1 as

$$d : (s_0, s_1) \to \mathbb{N}_0, s_0 \wedge s_1 \in S. \tag{1}$$

The distance function depends on the search operator used. Starting from a solution s_0, local search uses a small distance $d_{max} = d_{s_0,s_1}$ to choose a new solution s_1. We define the neighborhood function as

$$N : s_0 \to \{s_1 \in S \,|\, s_1 \neq s_0 \wedge 0 < d(s_0, s_1) \leq d_{\max}\}. \tag{2}$$

Iterated Local Search varies $d_{\max}$ during run-time to obtain higher diversification and to escape from a local optimum by using perturbation steps. This results in changes in the landscape during the run of the algorithm, and makes static analyses more difficult. Despite that, it is generally accepted to study a landscape defined by a fitness function and one or several induced distances [16].

3.3 Definition of Local Optima

A fitness landscape can have one or more local optima. A local optimum is a solution that has no superior neighbors. For a maximization problem, we define a function

$$N_{\text{sup}}(s) = \{n \in S \,|\, n \in N(s) \wedge f(n) > (f(s)\} \tag{3}$$

which returns the neighbors of a solution $s \in S$ that have a superior fitness. As local optima have no superior neighbors, the set

$$LO = \{lo \in S \,|\, N_{\text{sup}}(lo) = \{\}\,\}. \tag{4}$$

contains all the local optima, which also includes the global optimum.

3.4 Basins of Attraction

The basin of attraction is defined as the set of solution candidates from which local search converges to a particular local or global optimum. Extracting the basins of a fitness landscape is required in order to calculate the edges of LONs. The extraction process depends on the selection rule of the hill climbing algorithm [2]. Our implementation of ILS used best improvement hill climbing, (Algorithm 2) which accepts only the best of all superior neighbor solutions. Consequently, each solution in the search space belongs to the basin of exactly one local optimum and the basins form a partition set of the search space. These basins are referred to as unconditional basins [16]. The function

$$B : lo \rightarrow \mathcal{P}(S \backslash LO) \tag{5}$$

assigns a subset from the power set of solutions over the search space to each local optimum $lo \in LO$, which is the basin of lo. In the following, we use B as a function which returns the unconditional basins.

3.5 Landscape Features

Structural features of fitness landscapes are often used to predict the performance of algorithms. A well-elaborated collection of such features is given by [18]. Two of the frequently used features are ruggedness [19] and deceptiveness [20]. The idea of ruggedness is that the smoother the landscape is, the easier it is to search the landscape in order to find the global optimum. Ruggedness is a consequence of modality, i.e. the presence of local optima. The higher the number of local optima, the more rugged is the landscape. Ruggedness is usually measured as the correlation of fitness values between pairs of neighboring solutions ρ_{nn} (nearest-neighbor-correlation). The usual way to calculate ρ_{nn} is to perform a random walk across the search space and draw samples of the fitness of solution pairs that are neighbors.

A landscape is deceptive if the structure of the search space leads away from the global optimum. A measurement for deceptiveness is the correlation

between the fitness of the solutions and their distance to the global optimum [20]. For the calculation of the fitness-distance correlation ρ_{fd}, the global optimum must be known in advance. A random sample of solutions is drawn and ρ_{fd} is determined between their fitness and their distance to the global optimum. A totally misleading landscape with a fitness-distance correlation $\rho_{fd} \approx -1$ is often referred to as a trap.

4 Local Optima Networks

4.1 Concept

LONs have been inspired by the study of energy landscapes in chemical physics [1,21]. A LON is a graph representation of a fitness landscape. A graph G consists of two sets each for vertices and edges $G = (V, E)$. The vertex set V contains all the local optima of the fitness landscape. E contains the edges that model transitions between the local optima. In the case of LONs, the edges are directed and weighted. The existence and weight of edges depend on the trajectory of the search algorithm. To model the dynamics of Iterated Local Search, [3] introduced the concept of escape edges.

4.2 Escape Edges

Escape edges are defined according to the distance function of the fitness landscape d (minimal number of moves between two solutions). There is an integer $D > 0$ that is depicted as the distance that the ILS search applies to perform a perturbation step. There is a directed edge $E_{xy} > 0$ from local optimum lo_x to lo_y if there exists a solution s such that

$$d(s, lo_x) \leq D \wedge s \in B(lo_y). \tag{6}$$

The weight of this edge reflects the probability that an algorithm escapes from lo_x to lo_y. It is the number of solutions within reach of the escape step and which belong to the basin of lo_y. Since our implementation used best improvement hill climbing, the function B here returns the unconditional basin of attraction. The number of solutions with an opportunity to escape is normalized by the total number of solutions within reach:

$$E_{xy} = \frac{|\{s \in S \,|\, d(s, lo_x) \leq D \wedge s \in B(lo_y)\}|}{|\{s \in S | d(s, lo_x) \leq D\}|}. \tag{7}$$

5 PageRank Centrality

The centrality of nodes is a concept of network analysis to identify important or influential nodes [22]. Google were the first to assess the relevance and importance of web sites by their centrality in the linkage structure of the web. To this purpose, they have been using PageRank Centrality [23], which is a variant of the

Eigenvector centrality. It is based on the model of a user who surfs the web by randomly clicking links. The PageRank value of a website reflects the probability that the surfer currently is on this website. There are three factors determining the PageRank of a web page: the number of links a page receives, the number of outgoing links of the linking pages, and the PageRank of the linking pages. Thus, PageRank is a recursively defined concept. For detailed information on the notion and application of PageRank, we refer to [24].

To calculate the PageRank of websites, we need a transition matrix Π, which is a stochastic matrix of the linkage structure matrix E with all rows and columns normalized to sum up to 1. In LONs, the transition matrix E as defined by the edge weights in lemma (7) is a normalized, stochastic matrix. Thus, we can set $\Pi = E$.

A parameter of PageRank is the damping factor α, which reflects the fact that a random surfer may—instead of following links—visit a totally random page at some point. This probability is represented by $1 - \alpha$. A typical value is $\alpha = 0.85$, which says that a surfer chooses a random page after about five link clicks. Hill climbing algorithms do not make any jumps in the search space, thus $\alpha = 1.0$ was set for analyzing the LONs with Basin Transition Probabilities in previous work [9]. In the case of ILS, there is a perturbation operator. However, the escape edges in the corresponding LON model already reflect this behavior. Consequently, we set $\alpha = 1.0$ for our analysis.

Then, the PageRank centrality of all nodes (local optima) is given by the vector P, which is the Eigenvector of Π:

$$P = \Pi \times P. \tag{8}$$

If Π is a strongly connected graph, there exists a unique solution for P [25, 26]. These conditions are fulfilled in our case, since negative probabilities are impossible by our definition of the LONs transition matrix. In addition, we did not observe any disconnected components in our LONs. The vector P contains the PageRank centralities of all the local optima in the search space's LON. Consequently, we define P_{opt} as the PageRank value of the global optimum.

6 Experiment

6.1 Search Space: NK Model

For our experiment, we used the well-known Kauffman NK model [27], which is a family of combinatorial optimization problems from the class of pseudo-boolean functions. Each instance of the model can be generated by the two parameters N and K. Each solution $s \in S$ consists of N binary decision variables, forming a search space of $|S| = 2^N$ possible states. The fitness function

$$f_{NK} : [0, 1]^N \rightarrow [0, 1] \tag{9}$$

assigns a score to every combination of bits. It consists of N sub-functions, which assign a fitness for each bit i, depending on the state of bit i and the states of

K other bits

$$f_i : [0,1]^{K+1} \rightarrow [0,1]. \tag{10}$$

The total fitness $f_{NK}(s)$ is the average of the values of the N sub-functions. All function values are normalized between 0 and 1, with 1.0 as the fitness of the global optimum. The parameter K determines the number of co-variables per decision variable and thus the complexity of an instance (epistasis). A value of $K = 0$ results in a problem solvable in linear time. $K = N-1$ leads to a problem where each decision variable can only be set to the optimal value if all other $N-1$ co-variables are considered. Even though it is commonly accepted that a higher level of epistasis lead to higher search difficulty of landscapes, it is only a rough measure for difficulty. Landscapes with an identical level of epistasis can have a significant variety of search difficulty. Our results on the performance of ILS in Sect. 7.1 underpin this assumption.

The distance between two binary solutions $x, y \in S$ is calculated by the Hamming-distance $d(x,y) = \sum_{i=0}^{n} |x_i - y_i|$, i.e., the number of bits that are set to different values when comparing two solutions. For the hill climbing procedure in ILS, we assumed that two solutions x, y are neighbors if their Hamming distance equals one ($d_{\max} = 1$). Thus, a local search step flips exactly one bit of the current solution. As perturbation operator in ILS, we flipped two bits in one step.

6.2 Implementation

The objective of our experiment is to predict the success rate p_s and the average fitness $avg(f)$ achieved by ILS in a variety of different search spaces. We generated 300 instances in total of the NK model with $N = 15$ bits. To test different levels of epistasis, we used 100 instances each for $K \in \{2, 7, 12\}$. A search space contained $2^{15} = 32,768$ solutions, which is small, but manageable for our analysis. For each instance, we extracted the fitness landscape and the LON with escape edges and calculated the following features:

1. P: the PageRank Vector, and P_{opt}: the PageRank of the global optimum,
2. F: the vector containing the fitness values of all the local optima,
3. ρ_{nn}: the ruggedness of the fitness landscape, measured by the Pearson correlation between the fitness of nearest neighbors [19] and
4. ρ_{fd}: the deceptiveness of the landscape, measured by the Pearson correlation between fitness and distance to the global optimum [20].

For each problem instance, we performed 1,000 independent runs of ILS. The initial solutions were randomly selected. As a stopping threshold for ILS, we limited the running time by a reasonable number of function evaluations, which was 1/5th of the search space size [8]. We examined two relationships: first, we studied how the PageRank of the global optimum P_{opt} predicts the success rate of ILS in the different problem instances. The purpose of this approach is to confirm that the findings from the previous study on LONs with basin transition

probabilities [9] also hold for the LON_{ee} model. We have also compared the PageRank to the average number of function evalutions $avg(t)$ that were performed in those runs in which the global optimum was found. Second, we aimed to predict the solution quality of ILS (the average fitness) by the PageRank vectors of the LONs. To achieve this, we used the PageRank vector P and the fitness vector F of the local optima for each search space. We expect that P provides a stationary distribution over the whole search space and is a probability vector, s.t. $\sum P = 1$. We use these probabilities to calculate an average of the fitness of the local optima as given by F, weighted by their stationary probability:

$$E[f] = P \times F. \tag{11}$$

The result is a scalar value which we call the expected fitness $E[f]$ achieved by ILS in a distinct search space. We calculated $E[f]$ for all the problem instances. We assessed the predictive power of the different predictor metrics by the determination coefficient R^2 from a univariate, linear regression model. As a benchmark, we have also calculated the R^2 between the performance measures and the classical metrics from fitness landscapes analysis (ruggedness and deceptiveness).

We implemented our generator for NK landscapes, the extraction procedure for LONs and the ILS algorithm in the Java programming language. For computation, we utilized 20 nodes from an HPC cluster called *Mogon* with each 64 cores and 256 GB of RAM. To calculate the PageRank values of the nodes, we used the *NetworkX Library*. Our statistical analysis was conducted using the *R framework*.

7 Results

7.1 Empirical Performance of ILS

As a pre-test of our experiments, we examined the performance of ILS by success rate p_s, average fitness $avg(f)$ and the number of fitness function evaluations to find the global optimum $avg(t)$. The results can be obtained from Fig. 1. In the landscapes with low epistasis, we can see that ILS could easily find the global optimum in the majority of the search spaces. With increasing value of the exogenous parameter for epistasis K, the average success rate decreases, and so does the average score achieved by ILS. The number of fitness evaluations necessary to find the global optimum increases with the epistasis: more epistasis lead to a higher modality, i.e. the number of local optima. The more local optima are in a search space, the more perturbations are necessary to find the global optimum. All of these observations are as expected: higher epistasis leads to a higher search difficulty, and thus lead to a lower success rate, a lower average fitness and longer running times. We can also see that there is as high variance of all the performance measures within the different classes of K, indicating that epistasis has only limited explanatory power for search difficulty.

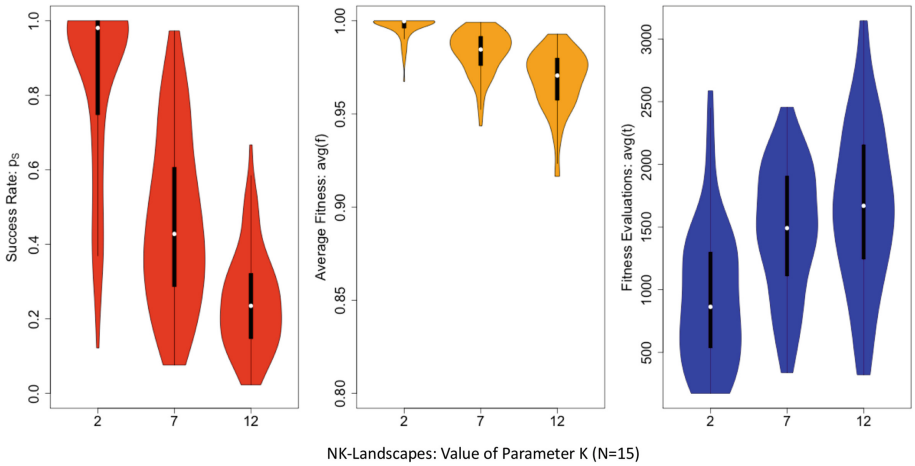


Fig. 1. The three performance measures of ILS over the epistasis K of the NK-landscapes: Success Rate (left), Average Fitness (middle) and Running Time by the Number of Fitness Evaluations (right).

7.2 Prediction of Success Rate and Average Fitness

To assess the predictive quality, we calculated all coefficients of determination (R^2) for each combination of predictor metric and performance measure. We have also made separate calculations for different levels of epistasis. The results for all combinations can be obtained from Table 1. In Fig. 2, we have plotted the performance of ILS (success rate, average fitness and running time) over the three predictor metrics. Each dot in the plot represents one search space.

As a first step, we take a look at the standard metrics that are frequently used in literature. Over all K, ruggedness and deceptiveness each can explain around 55 % of the variance of success rate and 44 %/35 % of average fitness. This is an intermediate statistical correlation. This correlation becomes weaker the higher the level of epistasis is. In the cases where $K \in \{7, 12\}$, the traditional metrics fail to explain any variance in the performance of ILS over all metrics. An explanation for this could be that the landscapes with high epistasis have a low variance in their ruggedness ρ_{nn} and deceptiveness ρ_{fd}. A low variance in the regressor variables then results in a low R^2.

We will now take a look at the prediction by PageRank Centrality. Our expectations were a high correlation between the PageRank of the global Optimum and success rate, as well as between the average score of the local optima (weighted by their PageRank) and the fitness achieved by ILS on average. We observed the following patterns in our results:

– The PageRank of the global optimum in the LON with escape edges P_{opt} explains almost 97 % of the success rate of ILS p_s. Obviously, the PageRank as obtained from the LON_{ee} model is a good indicator of the search difficulty for ILS.

Table 1. R^2 for the different performance measures and predictor metrics.

Performance	Predictor	$\forall K$	K = 2	K = 7	K = 12
Success rate: p_s	NN correl.: ρ_{nn}	0.5567	0.5567	0.0019	0.0134
	FD correl.: ρ_{fd}	0.5400	0.5400	0.0751	0.0897
	PageRank: P_{opt}	0.9675	0.9675	0.9683	0.8870
Average fitness: $avg(f)$	NN correl.: ρ_{nn}	0.4497	0.4497	0.0235	0.0012
	FD correl.: ρ_{fd}	0.3523	0.3524	0.0075	0.0295
	PageRank weighted fitn.: E[f]	0.9714	0.9714	0.9668	0.9554
Running time: $avg(t)$	NN correl.: ρ_{nn}	0.2090	0.0027	0.0019	0.0007
	FD correl.: ρ_{fd}	0.1455	0.0006	0.0089	0.0069
	PageRank: P_{opt}	0.0006	0.1444	0.3081	0.3950

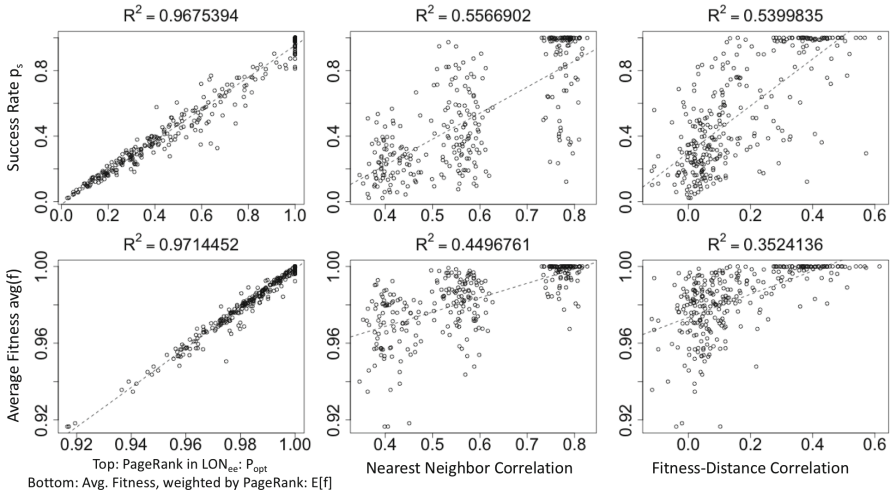


Fig. 2. The performance of ILS (Top: Success Rate, Bottom: Average Fitness) over the Predictor Metrics (PageRank/PageRank Weighted Avg. Fitness, Ruggedness and Deceptiveness, from left to right). Each of the Dots represents a single Problem Instance.

- The expected average fitness $E[f]$ explains almost 97 % of the average score achieved by ILS $avg(f)$. Thus, the PageRank vector of a LON P seems to reflect the dynamics of ILS in terms of the probability to achieve a certain state.
- These results are robust for different levels of epistasis K. For high values of K, the R^2 is slightly reduced for both predictors, but it is still a very strong correlation and significantly better than traditional landscape metrics. Thus, the LON with escape edges nearly approximates the dynamics of ILS.
- The number of fitness evaluations needed to locate the global optimum $avg(t)$ is only weakly correlated to all of the predictor metrics. Surprisingly, in the case of medium and high epistasis, the PageRank seems to predict 30–40% of the variance of running time. An explanation for this could be that in

cases of high epistasis, the basins are very small. In landscapes with small basins, the running time of ILS is dominated by perturbation steps, and there is nearly no hill climbing. Since the escape edges in the LON_{ee} map these perturbations, the LON_{ee} perfectly matches the stochastic process of ILS in such cases. Then, the LON is more likely to reflect the running time than in cases where ILS needs to spend many function evaluations for the hill climbing procedure.

In summary, we have shown that the PageRank of the global optimum in LONs with escape edges perfectly predicts the search difficulty of landscapes for ILS, i.e. the empirical success rate. Moreover, we found that the stationary distribution of the PageRank vector over all local optima is useful to make predictions about the solution quality when running ILS in a certain search space[2]. Both predictions work for all levels of epistasis, which is a clear advantage to the concepts of ruggedness and deceptiveness.

8 Conclusions

In this study, we have contributed to recent research on predicting search difficulty of landscapes with the help of local optima networks and the metrics from the network analysis framework. We have shown that the PageRank Centrality of local optima can be used to predict the average fitness and success rate achieved by search heuristics. This works because LONs are an approximation of the fitness landscape's Markov Chain and the PageRank reflects the stationary distribution of the states in this chain. Other than classical metrics of landscape analysis, this method is robust against different levels of epistasis, i.e. the number of interdependencies between the decision variables. The PageRank of the global optimum also predicts the running time with limited accuracy in landscapes with high epistasis. Thus, LONs can be used as a tool to draw conclusions on the structure of problems. We have shown that predictions made with PageRank in a previous study are applicable with a LON model that can be extracted in reasonable time. A practical application of these findings could be in the selection process of problem instances for benchmark purposes. In benchmarks, test reliability is an important criterion, and the PageRank could be easily used to select instances that guarantee a uniform search difficulty or expected fitness outcome.

A limitation of our study is the size of the problem instances used. Even though we are convinced that our results extrapolate to larger instances, it would be interesting to perform further examinations on this, e.g. by sampling the local optima instead of evolving the whole search space. Another limitation is of fundamental nature: even though we have not made further tests in this assumption, we think that it is not possible to make general statements on the

[2] We have also replicated this result to predict the average fitness achieved by local search with LONs with basin transition probabilities. Results are available from the authors upon request.

performance of an algorithm with an arbitrary LON model. Instead, the LON model must match the dynamics of the search method. For instance, in the case of ILS, the escape edges must consider the distance of the perturbation step. However, this study provides evidence that the prediction by PageRank works across different LON models in combination with a distinct search heuristic. For future work, we suggest to conduct further analysis on the analysis of problem structure by LONs. For instance, it would be worthwhile to study whether it is possible to make assumptions over the search difficulty of landscapes for a variety of search heuristics with the help of LONs. Furthermore, it would be interesting to study the structure of larger problem instances with networks (e.g. by sampling instead of fully evolving the search space) and see if there are patterns in the distribution of local optima.

References

1. Ochoa, G., Tomassini, M., Vérel, S., Darabos, C.: A study of NK landscapes' basins and local optima networks. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation - GECCO 2008, p. 555. ACM Press, New York (2008)
2. Ochoa, G., Verel, S., Tomassini, M.: First-improvement vs. best-improvement local optima networks of NK landscapes. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 104–113. Springer, Heidelberg (2010)
3. Vérel, S., Daolio, F., Ochoa, G., Tomassini, M.: Local optima networks with escape edges. In: Hao, J.-K., Legrand, P., Collet, P., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) EA 2011. LNCS, vol. 7401, pp. 49–60. Springer, Heidelberg (2012)
4. Ochoa, G., Chicano, F., Tinós, R., Whitley, D.: Tunnelling crossover networks. In: Proceedings of the 2015 Genetic and Evolutionary Computation Conference - GECCO 2015, pp. 449–456. ACM Press, Madrid, Spain (2015)
5. Lu, G., Li, J., Yao, X.: Fitness landscapes and problem difficulty in evolutionary algorithms: from theory to applications. In: Richter, H., Engelbrecht, A. (eds.) Recent Advances in the Theory and Application of Fitness Landscapes. ECC, vol. 6, pp. 141–162. Springer, Heidelberg (2014)
6. Malan, K.M., Engelbrecht, A.P.: Fitness landscape analysis for metaheuristic performance prediction. In: Richter, H., Engelbrecht, A. (eds.) Recent Advances in the Theory and Application of Fitness Landscapes. ECC, vol. 6, pp. 109–140. Springer, Heidelberg (2014)
7. Ochoa, G., Verel, S., Daolio, F., Tomassini, M.: Local optima networks: a new model of combinatorial fitness landscapes. In: Richter, H., Engelbrecht, A. (eds.) Recent Advances in the Theory and Application of Fitness Landscapes. ECC, vol. 6, pp. 245–276. Springer, Heidelberg (2014)
8. Daolio, F., Verel, S., Ochoa, G., Tomassini, M.: Local optima networks and the performance of iterated local search. In: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference - GECCO 2012, p. 369 (2012)
9. Herrmann, S., Rothlauf, F.: Predicting heuristic search performance with pageRank centrality in local optima networks. In: Proceedings of the 2015 Genetic and Evolutionary Computation Conference - GECCO 2015, pp. 401–408. ACM Press, Madrid, Spain (2015)

10. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. Handbook of Metaheuristics, pp. 320–353. Kluwer Academic Publishers, Boston (2003)
11. Ochoa, G., Veerapen, N., Whitley, D., Burke, E.K.: The multi-funnel structure of TSP fitness landscapes : a visual exploration. In: Artificial Evolution (EA 2015). Lyon (2015)
12. Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problems. INFORMS J. Comput. **15**, 82–92 (2003)
13. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**, 498–516 (1973)
14. Rothlauf, F.: Design of Modern Heuristics: Principles and Application. Springer, Heidelberg (2011)
15. Wright, S.: The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In: Proceedings of the 6th International Congress of Genetics, pp. 356–366 (1932)
16. 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)
17. Reidys, C.M., Stadler, P.F.: Combinatorial landscapes. SIAM Rev. **44**, 3–54 (2002)
18. Kallel, L., Naudts, B., Reeves, C.R.: Properties of fitness functions and search landscapes. In: Kallel, L., Naudts, B., Rogers, A. (eds.) Theoretical Aspects of Evolutionary Computing. Natural Computing Series, pp. 175–206. Springer, Berlin and Heidelberg (2001)
19. Weinberger, E.: Correlated and uncorrelated fitness landscapes and how to tell the difference. Biol. Cybern. **336**, 325–336 (1990)
20. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the Sixth International Conference on Genetic Algorithms (1995)
21. Stillinger, F.H.: A topographic view of supercooled liquids and glass formation. Science **267**, 1935–1939 (1995)
22. Borgatti, S.P.: Centrality and network flow. Soc. Netw. **27**, 55–71 (2005)
23. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Netw. ISDN Syst. **30**, 107–117 (1998)
24. Franceschet, M.: PageRank: standing on the shoulders of giants. Commun. ACM **54**, 92 (2011)
25. Frobenius, G.: Über Matrizen aus nicht negativen Elementen. In: Sitzungsberichte Preussische Akademie der Wissenschaft, pp. 456–477, Berlin (1912)
26. Perron, O.: Zur Theorie der Matrices. Mathematische Annalen **1**(64), 248–263 (1907)
27. Kauffman, S., Levin, S.: Towards a general theory of adaptive walks on rugged landscapes. J. Theor. Biol. **128**(1), 11–45 (1987)

Efficient Hill Climber for Multi-Objective Pseudo-Boolean Optimization

Francisco Chicano[1(✉)], Darrell Whitley[2], and Renato Tinós[3]

[1] Dept. de Lenguajes y Ciencias de la Computación, University of Málaga, Málaga, Spain
chicano@lcc.uma.es
[2] Department of Computer Science, Colorado State University, Fort Collins, CO, USA
whitley@cs.colostate.edu
[3] Department of Computing and Mathematics, University of São Paulo, Ribeirão Preto, SP, Brazil
rtinos@ffclrp.usp.br

Abstract. Local search algorithms and iterated local search algorithms are a basic technique. Local search can be a stand-alone search method, but it can also be hybridized with evolutionary algorithms. Recently, it has been shown that it is possible to identify improving moves in Hamming neighborhoods for k-bounded pseudo-Boolean optimization problems in constant time. This means that local search does not need to enumerate neighborhoods to find improving moves. It also means that evolutionary algorithms do not need to use random mutation as a operator, except perhaps as a way to escape local optima. In this paper, we show how improving moves can be identified in constant time for multiobjective problems that are expressed as k-bounded pseudo-Boolean functions. In particular, multiobjective forms of NK Landscapes and Mk Landscapes are considered.

Keywords: Hamming Ball Hill Climber · Delta evaluation · Multiobjective optimization · Local search

1 Introduction

Local search and iterated local search algorithms [8] start at an initial solution and then search for an improving move based on a notion of a neighborhood

F. Chicano—This research was partially funded by the Fulbright program, the Spanish Ministry of Education (CAS12/00274), the University of Málaga, Andalucía Tech, and the Spanish Ministry of Science and Innovation and FEDER (TIN2014-57341-R).
D. Whitley—It was also sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-11-1-0088. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.
R. Tinós—Would like to thank FAPESP (under grant 2015/06462-1) and CNPq for the financial support.

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 88–103, 2016.
DOI: 10.1007/978-3-319-30698-8_7

of solutions that are adjacent to the current solution. This paper will consider k-bounded pseudo-Boolean functions, where the Hamming distance 1 neighborhood is the most commonly used local search neighborhood.

Recently, it has been shown that the location of improving moves can be calculated in constant time for the Hamming distance 1 "bit flip" neighborhood [16]. This has implications for both local search algorithms as well as simple evolutionary algorithms such as the $(1 + 1)$ Evolutionary Algorithm. Since we can calculate the location of improving moves, we do not need to enumerate neighborhoods to discover improving moves.

Chicano et al. [3] generalize this result to present a local search algorithm that explores the solutions contained in a Hamming ball of radius r around a solution in constant time. This means that evolutionary algorithms need not use mutation to find improving moves; either mutation should be used to make larger moves (that flip more than r bits), or mutation should be used to enable a form of restart. It can also makes crossover more important. Goldman et al. [6] combined local search that automatically calculates the location of improving moves in constant time with recombination to achieve globally optimal results on relatively large Adjacent NK Landscape problems (e.g. 10,000 variables).

Whitley [15] has introduced the notion of Mk Landspaces to replace NK Landscapes. Mk Landscapes are k-bounded pseudo-Boolean optimization problems composed of a linear combination of M subfunctions, where each subfunction is a pseudo-Boolean optimization problem defined over k variables. This definition is general enough to include NK landscapes, MAX-kSAT, as well as spin glass problems.

In this paper, we extend these related concepts to multi-objective optimization. We define a class of multi-objective Mk Landscapes and show how these generalize over previous definitions of multi-objective NK Landscapes. We also show how exact methods can be used to select improving moves in constant time. In the multi-objective space, the notion of an "improving move" is complex because improvement can be improvement in all objectives, or improvement in only part of the objectives. When there are improvement in all objectives, then clearly the improvement should be accepted. However, when there are improvement in only a subset of objectives, it is less clear what moves should be accepted because it is possible for search algorithms to cycle and to visit previously discovered solutions. Methods are proposed that allow the identification of improving moves in constant time for multi-objective optimization. Methods are also proposed to prevent local search algorithms from cycling and thus repeatedly revisiting previously discovered solutions. The results of this work could also be introduced in existing local search algorithms for multi-objective optimization, like Anytime Pareto Local Search [5].

The rest of the paper is organized as follows. In the next section we introduce multi-objective pseudo-Boolean optimization problems. Section 3 defines the "Scores" of a solution. The Score vector tracks changes in the evaluation function and makes it possible to track the locations of improving moves. An algorithm is introduced to track multiple Scores and to efficiently update them for multi-objective optimization. Section 4 considers how to address the problems

of selecting improving moves in a multi-objective search space when the move only improves some, but not all, of the objectives. Section 5 empirically evaluates the proposed algorithms. Section 6 summarizes the conclusions and outline the potential for future work.

2 Multi-Objective Pseudo-Boolean Optimization

In this paper we consider pseudo-Boolean vector functions with k-bounded epistasis, where the component functions are *embedded landscapes* [7] or *Mk Landscapes* [15]. We will extend the concept of Mk Landscapes to the multi-objective domain and, thus, we will base our nomenclature in that of Whitley [15].

Definition 1 (Vector Mk Landscape). *Given two constants k and d, a* vector Mk Landscape $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ *is a d-dimensional vector pseudo-Boolean function defined over $\mathbb{B}^n$ whose components are Mk Landscapes. That is, each component f_i can be written as a sum of m_i subfunctions, each one depending at most on k input variables*[1]*:*

$$f_i(x) = \sum_{l=1}^{m_i} f_i^{(l)}(x) \qquad \text{for } 1 \leq i \leq d, \tag{1}$$

where the subfunctions $f_i^{(l)}$ depend only on k components of $x \in \mathbb{B}^n$.

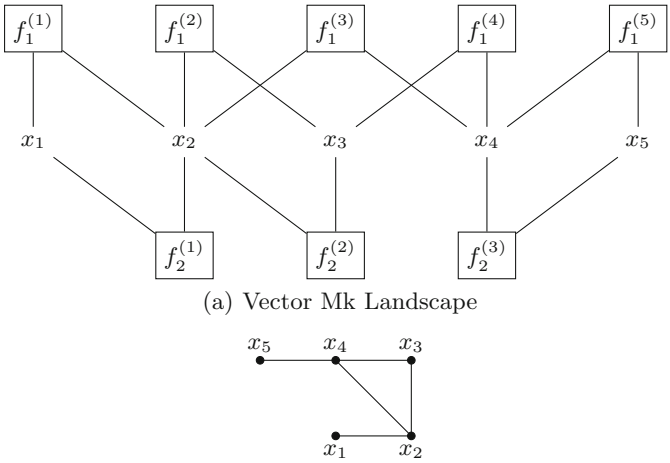
This definition generalizes that of Aguirre and Tanaka [1] for MNK Landscapes. In Fig. 1(a) we show a vector Mk Landscape with $d = 2$ dimensions. The first objective function, f_1, can be written as the sum of 5 subfunctions, $f_1^{(1)}$ to $f_1^{(5)}$. The second objective function, f_2, can be written as the sum of 3 subfunctions, $f_2^{(1)}$ to $f_2^{(3)}$. All the subfunctions depend at most on $k = 2$ variables.

It could seem that the previous class of functions is restrictive because each subfunction depends on a bounded number of variables. However, every compressible pseudo-Boolean function can be transformed in polynomial time into a quadratic pseudo-Boolean function (with $k = 2$) [12].

A useful tool for the forthcoming analysis is the *co-ocurrence graph* [4] $G = (V, E)$, where V is the set of Boolean variables and E contains all the pairs of variables (x_{j_1}, x_{j_2}) that *co-occur* in a subfunction $f_i^{(l)}$ for any $1 \leq i \leq d$ and $1 \leq l \leq m_i$ (both variables are arguments of the subfunction). In Fig. 1(b) we show the variable co-occurrence graph of the vector Mk Landscape of Fig. 1(a).

We will consider, without loss of generality, that all the objectives (components of the vector function) are to be maximized. Next, we include the definition of some standard multi-objective concepts to make the paper self-contained.

[1] In general, we will use **boldface** to denote vectors in $\mathbb{R}^d$, as $\mathbf{f}$, but we will use normal weight for vectors in $\mathbb{B}^n$, like x.



(a) Vector Mk Landscape

(b) Co-occurrence graph

Fig. 1. A vector Mk Landscape with $k = 2$, $n = 5$ variables and $d = 2$ dimensions (top) and its corresponding co-occurrence graph (bottom).

Definition 2 (Dominance). *Given a vector function* $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$*, we say that solution* $x \in \mathbb{B}^n$ dominates *solution* $y \in \mathbb{B}^n$*, denoted with* $x \succ_{\mathbf{f}} y$*, if and only if* $f_i(x) \geq f_i(y)$ *for all* $1 \leq i \leq d$ *and there exists* $j \in \{1, 2, \ldots, d\}$ *such that* $f_j(x) > f_j(y)$*. When the vector function is clear from the context, we will use* $\succ$ *instead of* $\succ_{\mathbf{f}}$*.*

Definition 3 (Pareto Optimal Set and Pareto Front). *Given a vector function* $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$*, the* Pareto Optimal Set *is the set of solutions* P *that are not dominated by any other solution in* $\mathbb{B}^n$*. That is:*

$$P = \{x \in \mathbb{B}^n | \nexists y \in \mathbb{B}^n, y \succ x\}\,. \tag{2}$$

The Pareto Front *is the image by* $\mathbf{f}$ *of the Pareto Optimal Set:* $PF = \mathbf{f}(P)$*.*

Definition 4 (Set of Non-dominated Solutions). *Given a vector function* $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$*, we say that a set* $X \subseteq \mathbb{B}^n$ *is a* set of non-dominated solutions *when there is no pair of solutions* $x, y \in X$ *where* $y \succ x$*, that is,* $\forall x \in X, \nexists y \in X, y \succ x$*.*

Definition 5 (Local Optimum [11]). *Given a vector function* $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$*, and a neighborhood function* $N : \mathbb{B}^n \to 2^{\mathbb{B}^n}$*, we say that solution* x *is a* local optimum *if it is not dominated by any other solution in its neighborhood:* $\nexists y \in N(x), y \succ x$*.*

3 Moves in a Hamming Ball

We can characterize a *move* in $\mathbb{B}^n$ by a binary string $v \in \mathbb{B}^n$ having 1 in all the bits that change in the solution. Following [3] we will extend the concept of *Score*[2] to vector functions.

[2] What we call *Score* here is also named Δ-evaluation by other authors [13].

Definition 6 (Score). *For $v, x \in \mathbb{B}^n$, and a vector function $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$, we denote the* Score *of x with respect to* move *v as $\mathbf{S}_v(x)$, defined as follows:*

$$\mathbf{S}_v(x) = \mathbf{f}(x \oplus v) - \mathbf{f}(x), \tag{3}$$

where $\oplus$ denotes the exclusive OR bitwise operation (sum in $\mathbb{Z}_2$).

The Score $\mathbf{S}_v(x)$ is the change in the vector function when we move from solution x to solution $x \oplus v$, that is obtained by flipping in x all the bits that are 1 in v. Our goal is to efficiently decide where to move from the current solution. If possible, we want to apply *improving* moves to our current solution. While the concept of "improving" move is clear in the single-objective case (an improving move is one that increases the value of the objective function), in multi-objective optimization any of the d component functions could be improving, disimproving or neutral. Thus, we need to be more clear in this context, and define what we mean by "improving" move. It is useful to define two kinds of improving moves: the *weak* improving moves and the *strong* improving moves. The reason for this distinction will be clear in Sect. 4.

Definition 7 (Strong and Weak Improving Moves). *Given a solution $x \in \mathbb{B}^n$, a move $v \in \mathbb{B}^n$ and a vector function $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$, we say that move v is a* weak improving move *if there exists $i \in \{1, 2, \ldots, d\}$ such that $f_i(x \oplus v) > f_i(x)$. We say that move v is a* strong improving move *if it is a weak improving move and for all $j \in \{1, 2, \ldots, d\}$ $f_j(x \oplus v) \geq f_j(x)$.*

Using our definition of Score, we can say that a move v is a weak improving move if there exists a $j \in \{1, 2, \ldots, d\}$ for which $S_{j,v}(x) > 0$. It is a strong improving move if $S_{i,v}(x) \geq 0$ for all $i \in \{1, 2, \ldots, d\}$ and there exists a $j \in \{1, 2, \ldots, d\}$ for which $S_{j,v}(x) > 0$.

From Definition 7 it can be noticed that if v is a strong improving move in x then $x \oplus v \succ x$, that is, the concept of strong improving move coincides with that of dominance. It can also be noticed that in the single-objective case, $d = 1$, both concepts are the same. Strong improving moves are clearly desirable, since they cannot be disimproving for any objective and they will improve at least one. Weak improving moves, on the other hand, improve at least one objective but could disimprove other ones.

In particular, if v is a weak, but not strong, improving move in solution x, then it will improve at least one objective, say i-th, and disimprove at least another one, say j-th. If this move is taken, in the new solution, $x \oplus v$, the same move v will be again a weak, but not strong, improving move. However, now v will improve (at least) the j-th objective and will disimprove (at least) i-th. Taking v again in $x \oplus v$ will lead to x, and the algorithm cycles. Any hill climber taking weak improving moves should include a mechanism to avoid cycling.

Scores are introduced in order to efficiently identify where the (weak or strong) improving moves are. For this purpose, we can have a data structure where all the improving moves can be accessed in constant time. As the search progresses the Score values change and they also move in the data structure

to keep improving moves separated from the rest. A naïve approach to track all improving moves in a Hamming Ball of radius r around a solution would require to store all possible Scores for moves v with $|v| \leq r$, where $|v|$ denotes the number of 1 bits in v.

If we naively use Eq. (3) to explicitly update the scores, we will have to evaluate all $\sum_{i=1}^{r} \binom{n}{i} = O(n^r)$ neighbors in the Hamming ball. Instead, if the objective function is a vector Mk Landscape where each Boolean variable appears in at most a constant number of subfunctions, we can design an efficient next improvement hill climber for the radius r neighborhood that only stores a linear number of Scores and requires a constant time to update them.

3.1 Scores Update

Using the fact that each component f_i of the objective vector function is an Mk Landscape, we can write:

$$S_{i,v}(x) = \sum_{l=1}^{m_i} \left(f_i^{(l)}(x \oplus v) - f_i^{(l)}(x) \right) = \sum_{l=1}^{m_i} S_{i,v}^{(l)}(x), \tag{4}$$

where we use $S_{i,v}^{(l)}$ to represent the score of the subfunction $f_i^{(l)}$ for move v. Let us define $w_{i,l} \in \mathbb{B}^n$ as the binary string such that the j-th element of $w_{i,l}$ is 1 if and only if $f_i^{(l)}$ depends on variable x_j. The vector $w_{i,l}$ can be considered as a mask that characterizes the variables that affect $f_i^{(l)}$. Since $f_i^{(l)}$ has bounded epistasis k, the number of ones in $w_{i,l}$, denoted with $|w_{i,l}|$, is at most k. By the definition of $w_{i,l}$, the next equalities immediately follow.

$$f_i^{(l)}(x \oplus v) = f_i^{(l)}(x) \quad \text{for all } v \in \mathbb{B}^n \text{ with } v \wedge w_{i,l} = 0, \tag{5}$$

$$S_{i,v}^{(l)}(x) = \begin{cases} 0 & \text{if } w_{i,l} \wedge v = 0, \\ S_{i,v \wedge w_{i,l}}^{(l)}(x) & \text{otherwise.} \end{cases} \tag{6}$$

Equation (6) claims that if none of the variables that change in the move characterized by v is an argument of $f_i^{(l)}$ the Score of this subfunction is zero, since the value of this subfunction will not change from $f_i^{(l)}(x)$ to $f_i^{(l)}(x \oplus v)$. On the other hand, if $f_i^{(l)}$ depends on variables that change, we only need to consider for the evaluation of $S_{i,v}^{(l)}(x)$ the changed variables that affect $f_i^{(l)}$. These variables are characterized by the mask vector $v \wedge w_{i,l}$. With the help of (6) we can re-write (4):

$$S_{i,v}(x) = \sum_{\substack{l=1 \\ w_{i,l} \wedge v \neq 0}}^{m_i} S_{i,v \wedge w_{i,l}}^{(l)}(x). \tag{7}$$

Equation (7) simply says that we don't have to consider all the subfunctions to compute a Score. This can reduce the runtime to compute the scores from scratch.

During the search, instead of computing the Scores using (7) after every move, it is more efficient in time to store the Scores $\mathbf{S}_v(x)$ of the current solution x in memory and update only those that are affected by the move.

In the following, and abusing of notation, given a move $v \in \mathbb{B}^n$ we will also use v to represent the set of variables that will be flipped in the move (in addition to the binary string).

For each of the Scores to update, the change related to subfunction $f_i^{(l)}$ after move $t \in \mathbb{B}^n$ can be computed with the help of $S_{i,v}^{(l)}(x \oplus t) = f_i^{(l)}(x \oplus t \oplus v) - f_i^{(l)}(x \oplus t)$ and $S_{i,v}^{(l)}(x) = f_i^{(l)}(x \oplus v) - f_i^{(l)}(x)$. The component $S_{i,v}$ will be updated by subtracting $S_{i,v}^{(l)}(x)$ and adding $S_{i,v}^{(l)}(x \oplus t)$. This procedure is shown in Algorithm 1, where the term $S_{i,v}$ represents the i-th component of the Score of move v stored in memory and M^r is the set of moves whose scores are stored. In the worst (and naïve) case M^r is the set of all strings v with at most r ones, $M^r = \{v | 1 \leq |v| \leq r\}$, and $|M^r| = O(n^r)$. However, we will prove in Sect. 3.2 that, for some vector Mk Landscapes, we only need to store $O(n)$ Scores to identify improving moves in a ball of radius r.

Algorithm 1. Efficient algorithm for Scores update

Input: scores vector $\mathbf{S}$, current solution x, move t

1: **for** (i, l) such that $w_{i,l} \wedge t \neq 0$ **do**
2: **for** $v \in M^r$ such that $w_{i,l} \wedge v \neq 0$ **do**
3: $S_{i,v} \leftarrow S_{i,v} + f_i^{(l)}(x \oplus t \oplus v) - f_i^{(l)}(x \oplus t) - f_i^{(l)}(x \oplus v) + f_i^{(l)}(x)$
4: **end for**
5: **end for**

3.2 Scores Decomposition

Some scores can be written as a sum of other scores. The benefit of such a decomposition is that we do not really need to store all the scores in memory to have complete information of the influence that the moves in a Hamming ball of radius r have on the objective function $\mathbf{f}$. The co-occurrence graph has a main role in identifying the moves whose Scores are fundamental to recover all the improving moves in the Hamming ball.

Let us denote with $G[v]$ the subgraph of G induced by v, that is, the subgraph containing only the vertices in v and the edges of E between vertices in v.

Proposition 1 (Score decomposition). *Let $v_1, v_2 \in \mathbb{B}^n$ be two moves such that $v_1 \cap v_2 = \emptyset$ and variables in v_1 do not co-occur with variables in v_2. In terms of the co-occurrence graph this implies that there is no edge between a variable in v_1 and a variable in v_2 and, thus, $G[v_1 \cup v_2] = G[v_1] \cup G[v_2]$. Then, the score function $\mathbf{S}_{v_1 \cup v_2}(x)$ can be written as:*

$$\mathbf{S}_{v_1 \cup v_2}(x) = \mathbf{S}_{v_1}(x) + \mathbf{S}_{v_2}(x). \tag{8}$$

Proof. Using (7) we can write:

$$S_{i,v_1\cup v_2}(x) = \sum_{\substack{l=1 \\ w_{i,l}\wedge(v_1\vee v_2)\neq 0}}^{m_i} S^{(l)}_{i,(v_1\vee v_2)\wedge w_{i,l}}(x)$$
$$= \sum_{\substack{l=1 \\ (w_{i,l}\wedge v_1)\vee(w_{i,l}\wedge v_2)\neq 0}}^{m_i} S^{(l)}_{i,(v_1\wedge w_{i,l})\vee(v_2\wedge w_{i,l})}(x).$$

Since variables in v_1 do not co-occur with variables in v_2, there is no $w_{i,l}$ such that $v_1 \wedge w_{i,l} \neq 0$ and $v_2 \wedge w_{i,l} \neq 0$ at the same time. Then we can write:

$$S_{i,v_1\cup v_2}(x) = \sum_{\substack{l=1 \\ w_{i,l}\wedge v_1\neq 0}}^{m_i} S^{(l)}_{i,v_1\wedge w_{i,l}}(x) + \sum_{\substack{l=1 \\ w_{i,l}\wedge v_2\neq 0}}^{m_i} S^{(l)}_{i,v_2\wedge w_{i,l}}(x) = S_{i,v_1}(x) + S_{i,v_2}(x),$$

and the result follows. □

For example, in the vector Mk Landscape of Fig. 1 the scoring function $\mathbf{S}_{\underline{1,3,4}}$ can be written as the sum of the scoring functions $\mathbf{S}_{\underline{1}}$ and $\mathbf{S}_{\underline{3,4}}$, where we used $\underline{i_1, i_2, ...}$ to denote the binary string having 1 in positions $i_1, i_2, \ldots$, and the rest set to 0.

A consequence of Proposition 1 is that we only need to store scores for moves v where $G[v]$ is a connected subgraph. If $G[v]$ is not a connected subgraph, then there are sets of variables v_1 and v_2 such that $v = v_1 \cup v_2$ and $v_1 \cap v_2 = \emptyset$ and, applying Proposition 1 we have $\mathbf{S}_v(x) = \mathbf{S}_{v_1}(x) + \mathbf{S}_{v_2}(x)$. Thus, we can recover all the scores in the Hamming ball of radius r from the ones for moves v where $1 \leq |v| \leq r$ and $G[v]$ is connected. In the following we will assume that the set M^r of Algorithm 1 is:

$$M^r = \{v \in \mathbb{B}^n | 1 \leq |v| \leq r \text{ and } G[v] \text{ is connected}\}. \tag{9}$$

3.3 Memory and Time Complexity of Scores Update

We will now address the question of how many of these Scores exist and what is the cost in time of updating them after a move.

Lemma 1. *Let $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ be a vector Mk Landscape where each Boolean variable appears in at most c subfunctions $f_i^{(l)}$. Then, the number of connected subgraphs with size no greater than r of the co-occurrence graph G containing a given variable x_j is $O((3ck)^r)$.*

Proof. For each connected subgraph of G containing x_j we can find a spanning tree with x_j at the root. The degree of any node in G is bounded by ck, since each variable appears at most in c subfunctions and each subfunction depends at most on k variables. Given a tree of l nodes with x_j at the root, we have to assign variables to the rest of the nodes in such a way that two connected

nodes have variables that are adjacent in G. The ways in which we can do this is bounded by $(ck)^{l-1}$. We have to repeat the same operation for all the possible rooted trees of size no greater than r. If T_l is the number of rooted trees with l vertices, then the number of connected subgraphs of G containing x_j and with size no greater than r nodes is bounded by

$$\sum_{l=1}^{r} T_l (ck)^{l-1} \leq \sum_{l=1}^{r} 3^l (ck)^{l-1} \leq 3(3ck)^r, \tag{10}$$

where we used the result in [10] for the asymptotic behaviour of T_l:

$$\lim_{l\to\infty} \frac{T_l}{T_{l-1}} \approx 2.955765. \tag{11}$$

□

Lemma 1 provides a bound for the number of moves in M^r that contains an arbitrary variable x_j. In effect, the connected subgraphs in G containing x_j corresponds to the moves in M^r that flip variable x_j. An important consequence is given by the following theorem.

Theorem 1. *Let* $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ *be a vector Mk Landscape where each Boolean variable appears in at most* c *subfunctions. Then, the number of connected subgraphs of* G *of size no greater than* r *is* $O(n(3ck)^r)$*, which is linear in* n *if* c *is independent of* n*. This is the cardinality of* M^r *given in* (9)*.*

Proof. The set of connected subgraphs of G with size no greater than r is the union of connected subgraphs of G of size no greater than r that contains each of the n variables. According to Lemma 1 the cardinality of this set must be $O(n(3ck)^r)$. □

The next Theorem bounds the time required to update the scores.

Theorem 2. *Let* $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ *be a vector Mk Landscape where each Boolean variable appears in at most* c *subfunctions* $f_i^{(l)}$*. The time required to update the Scores using Algorithm 1 is* $O(b(k)|t|(3ck)^{r+1})$ *where* $b(k)$ *is a bound on the time required to evaluate any subfunction* $f_i^{(l)}$*.*

Proof. Since each variable appears in at most c subfunctions, the number of subfunctions containing at least one of the bits in t is at most $c|t|$, and this is the number of times that the body of the outer loop starting in Line 1 of Algorithm 1 is executed. Once the outer loop has fixed a pair (i, l), the number of moves $v \in M^r$ with $w_{i,l} \wedge v \neq 0$ is the number of moves $v \in M^r$ that contains a variable in $w_{i,l}$. Since $|w_{i,l}| \leq k$ and using Lemma 1, this number of moves is $O(k(3ck)^r)$. Line 3 of the algorithm is, thus, executed $O(|t|ck(3ck)^r)$ times, and considering the bound on the time to evaluate the subfunctions, $b(k)$ the result follows. □

Since $|t| \leq r$, the time required to update the Scores is $\Theta(1)$ if c does not depend on n. Observe that if c is $O(1)$, then the number of subfunctions of the vector Mk Landscape is $m = \sum_{i=1}^{d} m_i = O(n)$. On the other hand, if every variable appears in at least one subfunction (otherwise the variable could be removed), $m = \Omega(n)$. Thus, a consequence of $c = O(1)$ is that $m = \Theta(n)$.

4 Multi-Objective Hamming-Ball Hill Climber

We have seen that, under the hypothesis of Theorem 1, a linear number of Scores can provide information of all the Scores in a Hamming ball of radius r around a solution. However, we need to sum some of the scores to get complete information of where all the improving moves are, and this is not more efficient than exploring the Hamming ball. In order to efficiently identify improving moves we have to discard some of them. In particular, we will discard all the improving moves whose scores are not stored in memory. In [3] the authors proved for the single-objective case that if none of the $O(n)$ stored scores is improving, then it cannot exist an improving move in the Hamming ball of radius r around the current solution. Although not all the improving moves can be identified, it is possible to identify local optima in constant time when the hill climber reaches them. This is a desirable property for any hill climber. We will prove in the following that this result can be adapted to the multi-objective case.

If one of the scores stored indicates a strong improving move, then it is clear that the hill climber is not in a local optima, and it can take the move to improve the current solution. However, if only weak improving moves can be found in the Scores store, it is not possible to certify that the hill climber reached a local optima. The reason is that two weak improving moves taken together could give a strong improving move in the Hamming ball. For example, let us say that we are exploring a Hamming ball of radius $r = 2$, variables x_1 and x_2 do not co-occur in a two-dimensional vector function, and $\mathbf{S}_{\underline{1}} = (-1, 3)$ and $\mathbf{S}_{\underline{2}} = (3, -1)$. Moves $\underline{1}$ and $\underline{2}$ are weak improving moves, but the move $\mathbf{S}_{\underline{1,2}} = \mathbf{S}_{\underline{1}} + \mathbf{S}_{\underline{2}} = (2, 2)$ is a strong improving move. We should not miss that strong improving move during our exploration.

To discover all strong improving moves in the Hamming ball we have to consider weak improving moves. But we saw in Sect. 3 that taking weak improving moves is dangerous because they could make the algorithm to cycle. One very simple and effective mechanism to avoid cycling is to classify weak improving moves according to a weighted sum of their score components.

Definition 8 (w-improving move and w-score). *Let $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ be a vector Mk Landscape, and $\mathbf{w} \in \mathbb{R}^d$ a d-dimensional weight vector. We say that a move $v \in \mathbb{B}^n$ is $\mathbf{w}$-improving for solution x if $\mathbf{w} \cdot \mathbf{S}_v(x) > 0$, where $\cdot$ denotes the dot product of vectors. We call $\mathbf{w} \cdot \mathbf{S}_v(x)$ the $\mathbf{w}$-score of move v for solution x.*

Proposition 2. *Let $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ be a vector Mk Landscape, and $\mathbf{w} \in \mathbb{R}^d$ a d-dimensional weight vector with $w_i > 0$ for $1 \leq i \leq d$. If there exists a strong improving move in a ball of radius r around solution x, then there exists $v \in M^r$ such that $\mathbf{w} \cdot \mathbf{S}_v(x) > 0$.*

Proof. Let us say that v is a strong improving move in the Hamming ball of radius r. Then there exist moves $v_1, v_2, \ldots v_j \in M^r$ such that $\mathbf{S}_v(x) = \sum_{l=1}^{j} \mathbf{S}_{v_l}(x)$. Since v is strong improving and all $w_i > 0$, we have $\mathbf{w} \cdot \mathbf{S}_v(x) = \sum_{l=1}^{j} \mathbf{w} \cdot \mathbf{S}_{v_l}(x) > 0$. There must be a v_l with $1 \leq l \leq j$ such that $\mathbf{w} \cdot \mathbf{S}_{v_l}(x) > 0$. □

Proposition 2 ensures that we will not miss any strong improving move in the Hamming ball if we take the weak improving moves with an improving **w**-score. Thus, our proposed Hill Climber, shown in Algorithm 2, will select strong improving moves in first place (Line 4) and **w**-improving moves when no strong improving moves are available (Line 6). In this last case, we should report the value of solution x, since it could be a non-dominated solution (Line 7). The algorithm will stop when no **w**-improving move is available. In this case, a local optima has been reached, and we should report this final (locally optimal) solution (Line 12). The algorithm cannot cycle, since only **w**-improving moves are selected, and this means that an improvement is required in the direction of **w**. A cycle would require to take a **w**-disimproving move at some step of the climb.

Algorithm 2. Multi-objective Hamming-Ball Hill Climber.

Input: scores vector **S**, weight vector **w**, initial solution x
Output: local optimum in x (and potentially non-dominated intermediate solutions)

```
 1: S ← computeScores(x);
 2: while w · S_v > 0 for some v ∈ M^r do
 3:    if there is a strong improving move v ∈ M^r then
 4:       t ← selectStrongImprovingMove(S);
 5:    else
 6:       t ← selectWImprovingMove(S);
 7:       report(x);
 8:    end if
 9:    updateScores(S,x,t);
10:    x ← x ⊕ t;
11: end while
12: report(x);
```

The procedure `report` in Algorithm 2 should add the reported solution to an external set of non-dominated solutions. This set should be managed by the high-level algorithm invoking the Hamming Ball Hill Climber.

For an efficient implementation of Algorithm 2, the scores stored in memory can be classified in three categories, each one stored in a different bucket: strong improving moves, **w**-improving moves that are not strong improving moves, and the rest. The scores can be moved from one of the buckets to the other as they are updated. The move from one bucket to another requires constant time, and thus, the expected time per move in Algorithm 2 is $\Theta(1)$, excluding the time required by `report`. This implementation corresponds to a next improvement

hill climber. An approximate form of best improvement hill climber could also be implemented following the guidelines in [14].

The weight vector $\mathbf{w}$ in the hill climber determines a direction to explore in the objective space. The use of $\mathbf{w}$ to select the weak improving moves is equivalent to consider improving moves of the single-objective function $\mathbf{w} \cdot \mathbf{f}$. However, there are two main reasons why it is more convenient to update and deal with the vector scores $\mathbf{S}$ rather than using scalar scores S of $\mathbf{w} \cdot \mathbf{f}$. First, using vector scores we can identify strong improving moves stored in memory, while using scalar scores of $\mathbf{w} \cdot \mathbf{f}$ it is not possible to distinguish between weak and strong improving moves. And second, it is possible to change $\mathbf{w}$ during the search without re-computing all the scores. The only operation to do after a change of $\mathbf{w}$ is a re-classification of the moves that are not strong improving[3].

Regarding the selection of improving moves in `selectStrongImprovingMove` and `selectWImprovingMove`, our implementation selects always a random one with the lowest Hamming distance to the current solution, that is, the move t with the lowest value of $|t|$. As stated by Theorem 2, such moves are faster, in principle, than other more distant moves, since the time required for updating the Scores is proportional to $|t|$.

5 Experimental Results

We implemented a simple Multi-Start Hill Climber algorithm to measure the runtime speedup of the proposed Multi-Objective Hamming Ball Hill Climber of Algorithm 2. The algorithm iterates a loop where a solution and a weight vector are randomly generated and Algorithm 2 is executed starting on them. The algorithm keeps a set of non-dominated solutions, that is potentially updated whenever Algorithm 2 reports a solution. The loop stops when a given time limit is reached. In our experiments shown here this time limit was 1 min. The machine used in all the experiments has an Intel Core 2 Quad CPU (Q9400) at 2.7 GHz, 3 GB of memory and Ubuntu 14.04 LTS. Only one core of the Processor is used. The algorithm was implemented in Java 1.6 and the source code is publicly available in GitHub[4].

To test the algorithm we have focused on MNK Landscapes [1]. An MNK Landscape is a vector Mk Landscape where all $m_i = N$ for all $1 \leq i \leq d$ and each subfunction $f_i^{(l)}$ depends on x_i and other K more variables (thus, $k = K+1$). The subfunctions $f_i^{(l)}$ are randomly generated using real values between 0 and 1. In order to avoid inaccuracy problems with floating point arithmetic, instead of real numbers we use integer numbers between 0 and $q-1$ and the sum of subfunctions are not divided by N. That is, each component f_i is an NKq Landscape [2]. We also focused on the *adjacent model* of NKq Landscape. In this model the

[3] Distinguishing the weak, but not strong, improving moves from the strong disimproving moves in the implementation would reduce the runtime here, since only weak improving moves need to be re-classified.

[4] https://github.com/jfrchicanog/EfficientHillClimbers.

variables each $f_i^{(l)}$ depends on are consecutive, that is, $x_i, x_{i+1}, \ldots, x_{i+K}$. This ensures that the number of subfunctions a given variable appears in is bounded by a constant, in particular, $K+1$, and Theorems 1 and 2 apply. Although these functions are not very common in pseudo-Boolean multi-objective optimization they are appropriate to empirically illustrate the theoretical results.

5.1 Runtime

There are two procedures in the hill climber that requires $\Omega(n)$ time. The first one is a *problem-dependent* initialization procedure, where the scores to be stored in memory are determined. This procedure is run only once in one run of the multi-start algorithm. In our experiments this time varies from 284 to 5,377 milliseconds.

The second procedure is a *solution-dependent* initialization of the hill climber starting from random solution and weight vector. This procedure is run once in each iteration of the multi-start hill climber loop, and can have an important impact on algorithm runtime, especially when there are no many moves during the execution of Algorithm 2. On the other hand, as the search progresses and the non-dominated set of solutions grows, the procedure to update it could also require a non-negligible runtime that depends on the number of solutions in the non-dominated set, which could be proportional to the number of moves done during the search.

In Fig. 2 we show the average time per move in microseconds (μs) for the Multi-Start Hill Climber solving MNK Landscapes with a time limit of 1 min, where N varies from $10,000$ to $100,000$, $q = 100$, $K = 3$, the dimensions are $d = 2$ and $d = 3$, and the exploration radius r varies from 1 to 3. We performed 30 independent runs of the algorithm for each configuration, and the results are the average of these 30 runs. To compute the average, we excluded the time required by the problem-dependent initialization procedure.

We can observe that moves are done very fast (tens to hundreds of microseconds). This is especially surprising if we consider the number of solutions "explored" in a neighborhood. For $N = 100,000$ and $r = 3$ the neighborhood contains around 166 trillion solutions that are explored in around 1 millisecond. For all values of r and d the increase in the average time per move is very slow (if any) when N grows. This slight growth in the average runtime is due to the solution-dependent initialization and the non-dominated set update, and contrasts with the theoretical $\Omega(n^r)$ time required by a black box algorithm.

As we could expect, the value of r has a great influence in the average time per move. In fact, the time is exponential in r. Regarding the memory required to store the Scores, we have already seen that it is $\Theta(n)$. In the particular case of the MNK Landscapes with an adjacent interaction model and $r \leq N/K$, it is not hard to conclude that the exact number of scores is $N(K^r - 1)/(K - 1)$, which is linear in N.

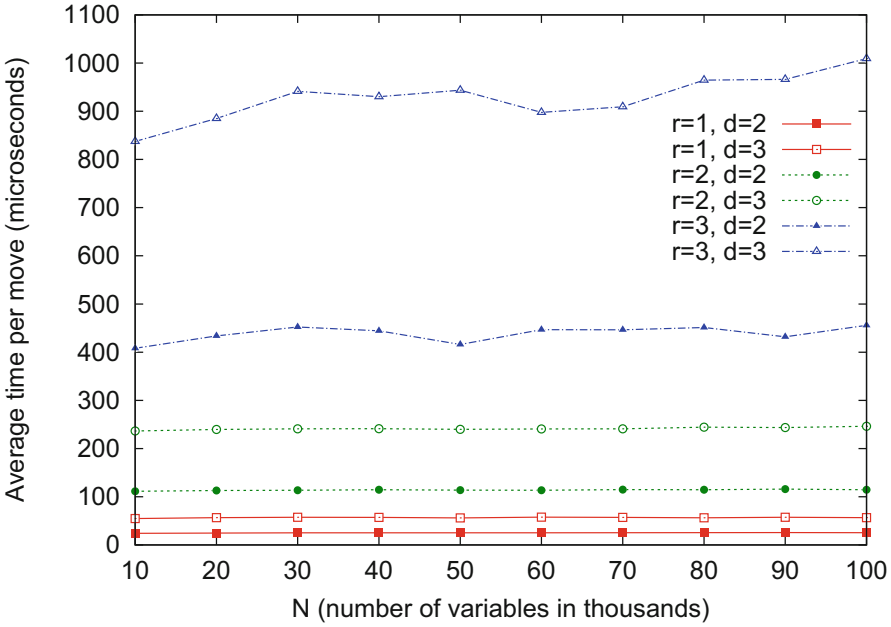


Fig. 2. Average time per move in μs for the Multi-Start Hill Climber based on Algorithm 2 for a MNK Landscape with $d = 2, 3$, $K = 3$, $q = 100$, $N = 10,000$ to $100,000$ and $r = 1$ to 3.

5.2 Quality of the Solutions

In a second experiment we want to check if a large value of r leads to better solutions. This highly depends on the algorithm that includes the hill climber. In our case, since the algorithm is a multi-start hill climber, we would expect an improvement in solution quality as we increase r. But at the same time, the average time per move is increased. Thus, there must be a value of r at which the time is so large that lower values for the radius can lead to the same solution quality. In Fig. 3 we show the 50 %-empirical attainment surfaces of the fronts obtained in the 30 independent runs of the multi-start hill climber for $N = 10,000$, $d = 2$, $q = 100$ and r varying from 1 to 3. The 50 %-empirical attainment surface (50 %-EAS) limits the region in the objective space that is dominated by half the runs of the algorithm. It generalizes the concept of *median* to the multi-objective case (see [9] for more details).

We can see in Fig. 3 that the 50 %-EAS obtained for $r = 2$ completely dominates the one obtained for $r = 1$, and the 50 %-EAS for $r = 3$ dominates that of $r = 2$. That is, increasing r we obtained better approximated Pareto fronts, in spite of the fact that the time per move is increased. This means that less moves are done in the given time limit (1 min) but they are more effective.

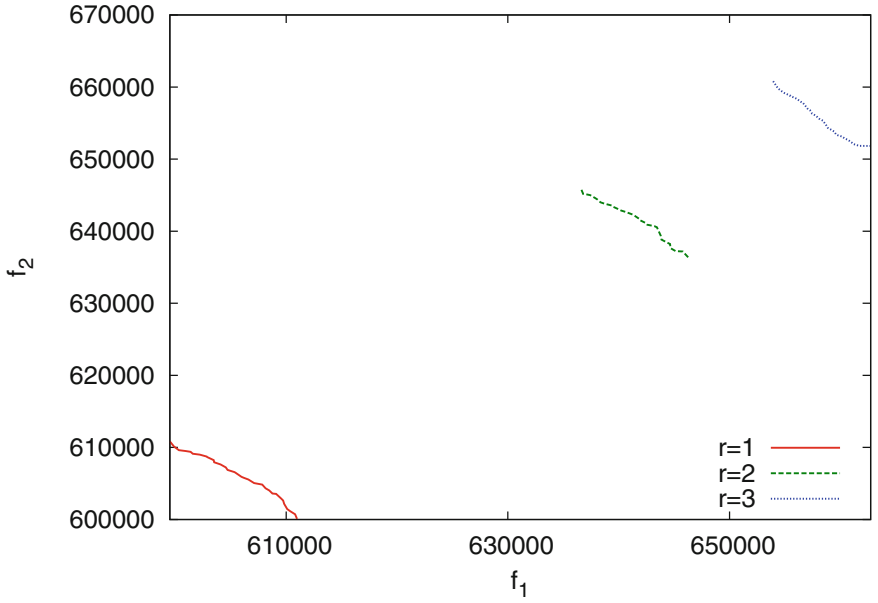


Fig. 3. 50 %-empirical attainment surfaces of the 30 independent runs of the Multi-Start Hill Climber based on Algorithm 2 for a MNK Landscape with $d = 2$, $K = 3$, $q = 100$, $N = 10,000$ and $r = 1$ to 3.

6 Conclusions and Future Work

We proposed in this paper a hill climber based on an efficient mechanism to identify improving moves in a Hamming ball of radius r around a solution of a k-bounded pseudo-Boolean multi-objective optimization problem. With this paper we contribute to an active line of research, sometimes called Gray-Box optimization [6], that suggests the use of as much information of the problems as possible to provide better search methods, in contrast to the Black-Box optimization.

Our proposed hill climber performs each move in bounded constant time if the variables of the problem appears in at most a constant number of subfunctions. In practice, the experiments on adjacent MNK Landscapes show that when $K = 3$ the average time per move varies from tenths to hundreds of microseconds if the exploration radius r varies from 1 to 3. This number is independent of n despite the fact that the hill climber is considering a Hamming Ball of radius r with $O(n^r)$ solutions.

Further work is needed to integrate this hill climber in a higher-level algorithm including mechanisms to escape from plateaus and local optima. On the other hand, one important limitation of our hill climber is that is does not take into account constraints in the search space. Constraint management and the combination with other components to build an efficient search algorithm seem two promising and challenging directions to work in the near future.

References

1. Aguirre, H.E., Tanaka, K.: Insights on properties of multiobjective MNK-landscapes. In: Proceedings of the Congress on Evolutionary Computation, vol. 1, pp. 196–203, June 2004
2. Chen, W., Whitley, D., Hains, D., Howe, A.: Second order partial derivatives for NK-landscapes. In: Proceeding of GECCO, pp. 503–510. ACM, New York (2013)
3. Chicano, F., Whitley, D., Sutton, A.M.: Efficient identification of improving moves in a ball for pseudo-boolean problems. In: Proceedings of Genetic and Evolutionary Computation Conference, pp. 437–444. ACM, New York (2014)
4. Crama, Y., Hansen, P., Jaumard, B.: The basic algorithm for pseudo-boolean programming revisited. Discrete Appl. Math. **29**(2–3), 171–185 (1990)
5. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Anytime pareto local search. Eur. J. Oper. Res. **243**(2), 369–385 (2015)
6. Goldman, B.W., Punch, W.F.: Gray-box optimization using the parameter-less population pyramid. In: Proceedings of Genetic and Evolutionary Computation Conference, pp. 855–862. ACM, New York (2015)
7. Heckendorn, R., Rana, S., Whitley, D.: Test function generators as embedded landscapes. In: Foundations of Genetic Algorithms, pp. 183–198. Morgan Kaufmann (1999)
8. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufman, San Francisco (2004)
9. Knowles, J.: A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. In: Proceedings of Intelligent Systems Design and Applications, pp. 552–557, September 2005
10. Otter, R.: The number of trees. Ann. Math. **49**(3), 583–599 (1948)
11. Paquete, L., Schiavinotto, T., Stützle, T.: On local optima in multiobjective combinatorial optimization problems. Ann. Oper. Res. **156**(1), 83–97 (2007)
12. Rosenberg, I.G.: Reduction of bivalent maximization to the quadratic case. Cahiers Centre Etudes Rech. Oper. **17**, 71–74 (1975)
13. Taillard, E.: Robust taboo search for the quadratic assignment problem. Parallel Comput. **17**(4–5), 443–455 (1991)
14. Whitley, D., Howe, A., Hains, D.: Greedy or not? best improving versus first improving stochastic local search for MAXSAT. In: Proceedings of the AAAI-2013 (2013)
15. Whitley, D.: Mk landscapes, NK landscapes, MAX-kSAT: a proof that the only challenging problems are deceptive. In: Proceedings of Genetic and Evolutionary Computation Conference, pp. 927–934. ACM, New York (2015)
16. Whitley, D., Chen, W.: Constant time steepest descent local search with lookahead for NK-landscapes and MAX-kSAT. In: Soule, T., Moore, J.H. (eds.) GECCO, pp. 1357–1364. ACM (2012)

Evaluating Hyperheuristics and Local Search Operators for Periodic Routing Problems

Yujie Chen[1(✉)], Philip Mourdjis[1], Fiona Polack[1], Peter Cowling[1], and Stephen Remde[2]

[1] YCCSA, Computer Science Department, University of York, York, UK
{yc1005,pjm515,fiona.polack,peter.cowling}@york.ac.uk
[2] Gaist Solutions Limited, Lancaster, UK
stephen.remde@gaist.co.uk

Abstract. Meta-heuristics and hybrid heuristic approaches have been successfully applied to Periodic Vehicle Routing Problems (PVRPs). However, to be competitive, these methods require careful design of specific search strategies for each problem. By contrast, hyperheuristics use the performance of low level heuristics to automatically select and tailor search strategies. Hyperheuristics have been successfully applied to problem domains such as timetabling and production scheduling. In this study, we present a comprehensive analysis of hyperheuristic approaches to solving PVRPs. The performance of hyperheuristics is compared to published performance of state-of-the-art meta-heuristics.

Keywords: Hyperheuristic · Computational analysis · PVRP

1 Introduction

Most (meta-)heuristic approaches applied to new search problem domains need expert input in design. To automatize the process, hyperheuristics provide a problem-independent approach that automatically applies an appropriate search strategy, by calling low level heuristics (LLHs) at each decision point [1]. Whilst various hyperheuristics have been tested for a range of optimization problems (e.g. [1–4]), none has yet addressed PVRPs.

Hyperheuristic approaches operate at a management level, consisting of selection and acceptance stages. The LLH(s) to test at each search stage may be deterministically or probabilistically selected. In more advanced selection strategies, hyperheuristics can learn from the performance of previous selections; performance is usually evaluated using problem-independent measures such as change in the solution quality or elapsed CPU time. Acceptance determines whether to replace the current solution with the one yielded by the selected LLH(s). A large variety of acceptance strategies, such as only improved (OI) (e.g. [1]) and simulated annealing (e.g. [5]), have been tested in literature. Theoretically, a hyperheuristic should adapt to any hard computational search problem, and provides a mechanism for studying the strengths and weaknesses of LLHs for a specific problem.

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 104–120, 2016.
DOI: 10.1007/978-3-319-30698-8_8

In this paper, we provide a comprehensive analysis on three types of hyperheuristics and apply them to benchmark and real-world PVRP instances with different characteristics. We also use the hyperheuristics to explore the strengths and weaknesses of LLHs designed for PVRPs. Section 2, briefly introduces PVRP and (meta-)heuristic solvers from literature. We then review LLHs designed for PVRP in Sect. 3. Section 4 presents the hyperheuristics. Sections 5 and 6 present the experimental design and analysis of the experiments respectively.

2 Periodic Routing Problem

PVRPs [6–9] provide a well-researched mathematical model for real-world problems such as inventory servicing, periodic maintenance, and on-site service planning. A PVRP comprises K vehicles which can be used to service the demands of N customers over M days. Each PVRP has constraints that must be met by legal solutions: vehicles start and end their journey at a depot; no more than K routes are built each day; vehicle capacity restrictions are respected; each customer request is serviced in one time slot by one vehicle; only one service pattern is chosen for each customer. A feasible visit pattern, $\lambda_i \in \Lambda_i$, for a customer i, is a pattern that meets all constraints and provides the level of servicing required for customer i. For example, a customer might require two service visits per week, on either Monday and Thursday or Tuesday and Friday, giving two feasible patterns. The PVRP objective is to design a set of daily routes, comprising feasible patterns for each customer, that minimizes the total travelling cost and satisfies the PVRP constraints.

2.1 Existing (Meta-)heuristic Solvers for PVRP

Heuristic approaches to PVRP developed since the 1970s [6,10–13] generate solutions by determining customer-day patterns that group geographically close customers. In 1995, Chao et al. [7] introduced a record-to-record meta-heuristic that outperformed the earlier heuristics approaches. Subsequent meta-heuristics approaches, including tabu search [8], scatter search [14] and variable neighbourhood search (VNS) [5], have all produced new best solutions for benchmark problems. Hybrid heuristics now present very competitive results: Gulczynski et al. [9] use integer programming-based improvement heuristics combined with routing-based local searches; Vidal et al. [15] propose a hybrid genetic algorithm that combines local search and sophisticated population management strategies to guide the search – an approach shown to perform better than all the above algorithms. Cordeau and Maischberger [16] combine tabu search and iterated local search to give a competitive, broad exploration of the search space.

3 Low Level Heuristics for the PVRP

A hyperheuristic has a repository of LLHs that operate directly on the solution space, and should provide good coverage of the solution space.

3.1 Constructive Heuristics

To construct a valid PVRP initial solution, most PVRP solvers first assign customers to days, then build a solution of a vehicle routing problem (VRP) for each day. For assignment, Cordeau et al. [8,16], Hemmelmayr et al. [5] and Vidal et al. [15] randomly select a feasible customer-day pattern for each customer; Chao et al. [7] and Gulczynski et al. [9] minimize the maximum demand serviced each day; Christofides and Beasley [6] minimize the daily total distance from each customer to the depot. To construct daily routes, the Clarke and Wright algorithm (CW) [17] and GENI insertion heuristic [18] are generally applied. Our hyperheuristics use the same approach as [5]: random assignment followed by a CW routes construction process for each day.

3.2 Perturbation Operators

From an initial solution, a hyperheuristic manages the application of perturbation operators to either daily routes or customer patterns. Application may be *first improvement* (FI) – seeking to improve the current solution, or *mutation* (shaking) to derive a similar, but new solution from the current solution.

Route Related Operators. There are a number of common operators used to modify single and multiple routes in PVRP (see VRP local search library [19]).

1. 2Opt: replace two edges from a route with two new edges (e.g. [7,9,15])
2. 3Opt: replace three edges from a route with three new edges (e.g. [5]).
3. Or-opt: remove a string of two to four nodes and insert it into a new position, either in the same route (e.g. [14]) or in a different route (e.g. [15]).
4. One point move (1PM): relocate a point to a new position, either in the same route or in a different route (e.g. [7,9]).
5. Two points swap (2PS): swap two points, either in the same route or between different routes (e.g. [9]).
6. Relocate: relocate a string of points from one route to another (e.g. [5,15]).
7. Cross: swap two chains of points between two routes. ([5,14]).

Route-based perturbation is typically applied as FI, embedded in an iterative local search (ILS) [7,9,14,15]; however, they can also be used as mutation operators: Hemmelmayr [5] uses "Relocate" and "Cross", for this purpose.

Pattern Related Operators. All these operators assign different valid patterns for selected customers. A customer with a new pattern is removed from their current routes and re-inserted to their new lowest-cost position on each day in the new pattern, meaning that we always get a complete PVRP solution.

1. Random pattern reassign (Pa_RR): randomly assign a new feasible visit pattern to n customers drawn at random. A tabu mechanism prevents a customer from being subject to reassignment again in the short term.

2. Score based reassign (Pa_SR): chooses n random customers, for each $i \in n$, assign the pattern λ_i with the highest score, $Q(\lambda_i)$. $Q(\lambda_i)$ is updated after any pattern related operator use; if no improvement is found, $Q(\lambda_i) = \sqrt{Q(\lambda_i)}$.
3. Pattern reassign FI (Pa_FIR): for each customer i, successively test each feasible pattern, the first improvement found is executed.
4. Two points pattern swap (Pa_2SW): swaps the visit patterns of two customers i and j which have the same available patterns, $\Lambda_i = \Lambda_j$ and $\lambda_i \neq \lambda_j$.

Mixed Operators. To improve flexibility, mixed operators support moves between days, and potentially modify both routes structure and customer patterns. Chao et al. [7] propose an operator that removes the current routing of a customer's current pattern, and inserts into a different set of routes, with or without changing the customer's current pattern. For our LLH repository, we design two mixed operators that operate on chains; all customers moved must have the same available patterns, and only customers with a single visit per pattern are used.

1. Relocate with Pattern (MRPa): relocate a chain of points from one route to another route in the same day or a different day.
2. Cross with Pattern (MCPa): swap two chains of points between two routes within the same day or between days.

3.3 Reinitialization

If the current solution has not been improved for a certain number of iterations, we assume the search is stuck in a local optimum that cannot be escaped by a small mutation. A reinitialization mechanism, Algorithm 1, is applied. The new solution is made feasible by repeatedly removing the customer with the greatest load requirement from any route in the candidate solution that violates duration or load constraints, and re-inserting in a route where the constraints are met.

Algorithm 1. Reinitialisation

Define:
  x_{best} is the best found solution so far
  p_{random} is the probability of generating a random initial solution
Reinitialisation(x_{best}, p_{random})
if random(seed) $< p_{random}$ **then**
  random assignment and CW daily routes construction (Sect. 3.1).
else
  Destroy w% of the longest routes in x_{best}.
  For each customer in destroyed routes, randomly reassign feasible visit pattern.
  Insert each customer greedily to cheapest position in each day of assigned pattern.
end if
Return the new (re)constructed solution x.

4 Hyperheuristics

We consider three types of hyperheuristics: simple hyperheuristics, learning based hyperheuristics and VNS based methods.

4.1 Simple Hyperheuristics

Simple hyperheuristics [1] have basic LLH selection mechanisms such as simple random (SR), random descent (RD), random permutation, random permutation descent, and greedy. Acceptance strategies, such as all moves or only improving (OI) were originally tested on a sales summit problem [1]. Here, we implement SR and RD combined with OI acceptance, designated SROI and RDOI, respectively.

SR randomly selects a LLH, based on a uniform distribution. RD randomly selects a LLH and applies it repeatedly until there is no further improvement in the solution. OI accepts a new solution only if it is better than the current solution, evaluated by fitness.

4.2 Learning Based Hyperheuristics

Learning based hyperheuristics adapt the LLH trial set according to the historical performance of each LLH. In each iteration, a favourable LLH is applied, based on predefined rules: in our implementation, the LLH from the trial set that produces the most improved solution is applied. Three well known learning based selection mechanisms are tested: binary exponential back off (BEBO) [3], reinforcement learning (RL) [20] and a ranked choice function (CF) [1].

BEBO. [3] uses a tabu based learning mechanism. The tabu tenure, $tabu_i$, changes dynamically, such that a LLH that performs poorly is disabled for a number of iterations (which increases exponentially if the LLH subsequently perform poorly). Each iteration only LLHs i with $tabu_i = 0$ are selected to form the trial set, T.

RL. [20] uses positive reinforcement to reward good LLH choices and negative reinforcement to penalise bad LLH choices. The utility value of each LLH is dynamically updated based on its performance, and the $w\%$ of LLHs with the highest utility form the trial set, T, for the next iteration. We apply hyperheuristic RL methods identified by Nareyek [20]. For each $LLH_i \in T$, $utility_i = \sqrt{utility_i}$. After testing, the best performing $LLH_{best} \in T$ that improves the solution is rewarded by setting $utility_i = utility_i^2 + 1$.

CF. [1] provides a different utility adoption scheme. In each iteration, the utility of each $LLH_i \in T$ is updated based on a linear function that considers the LLH's performance (evaluated by fitness change and execution time), the ability of the LLH in collaboration (evaluated by successively applied pairs of LLHs), and the elapsed time since the LLH was last called.

4.3 General Variable Neighbourhood Search (GVNS) with Learning

GVNS [21,22] differs from the hyperheuristics above in its more intensive use of local search (LS): the selected LLH is applied repeatedly rather than for one iteration only. Shaking is another critical component for GVNS, meaning that when no further improvement is found using LS, mutation operators are applied to facilitate the search to jump out of a local optimum.

Algorithm 2. General VNS

Define: $k_{max} = |LLH_{MU}|$, the number of mutation operators
GVNS$(x, LLH_{MU}, LLH_{FI}, t_{max})$

while $t < t_{max}$ **do**
  $k = 1$
  while $k < k_{max}$ **do**
    $x' = shaking(x, LLH^k_{MU})$
    $x'' =$ **VND**(x', LLH_{FI})
    If x'' is better than x **then**
    $x = x''$ and $k = 1$
    Otherwise $k = k + 1$
  end while
end while

Algorithm 3. General VNSr

GVNSr$(x, LLH_{MU}, LLH_{FI}, t_{max})$
while $t < t_{max}$ **do**
  randomly choose LLH^k_{MU} from LLH_{MU}
  $x' = shaking(x, LLH^k_{MU})$
  $x'' =$ **VND**(x', LLH_{FI})
  If x'' is better than x **then**
  $x = x''$
end while

GVNS is a parameter-free approach; LLH selection uses a pre-ordered LLH set [5]. Our experiments need to test a large number of LLHs, which is very CPU-intensive. We propose variations to the GVNS: Algorithms 2 and 3 with, respectively, fixed order and random selection strategies to manage the selection of mutation operators (LLH_{MU}). The first stage of GVNS takes a candidate solution, x, and shakes it by applying one mutation operator (from LLH_{MU}). The second stage of GVNS calls a variable neighbourhood descent algorithm, VND, which applies FI operators (LLH_{FI}) to the shaken solution. VNS is run over a fixed time, $t < t_{max}$.

Algorithm 4 describes the VND procedure called in Algorithms 2 and 3. The VND manages selection of FI operators using either random ordering or one of three RL-based LLH orderings: ascending, descending, and top w%. Although utility is calculated in all cases, it is not used in random ordering selection.

5 Experimental Design

We design experiments that allow us to analyse the performance of hyperheuristics from different angles. We use data (benchmark and real) with different

Algorithm 4. VND(x, LLH_{FI})

Choose subset $LLH'_{FI} \subseteq LLH_{FI}$ based on the LLH selection approach used.
Define: $k_{max} = |LLH'_{FI}|$, the number of local search operators to be tested.
$k = 1$
while $k < k_{max}$ **do**
  $x' = \mathbf{ILS}(x, LLH^k_{FI})$, where **ILS** applies the selected FI operator, LLH^k_{FI}, repeatedly until no further improvement occurs. In each iteration of **ILS**, $utility_k = \sqrt{utility_k}$; if LLH^k_{FI} makes an improvement, then $utility_k = utility_k^2 + 1$.
  If x' is better than x **then** $x = x'$ and $k = 1$
  Otherwise $k = k + 1$
end while

spatial characteristics. We also compare the performance of hyperheuristics with that of meta-heuristics applied to the benchmark problems.

The experiments are designed to replicate benchmark conditions from [15]. In particular, the search is always terminated after the fixed amount of CPU time stated in [15]. To check the suitability of this time limit for scalability experiments (Sect. 6.3), we ran preliminary experiments using twice the CPU time. We found no significant change in the quality of solutions, suggesting that a performance plateau is attained, and the chosen CPU time is appropriate.

All experiments are implemented in C♯ and executed on a cluster composed of 8 Windows computers, each with Intel Xeon E3-1230 CPU.

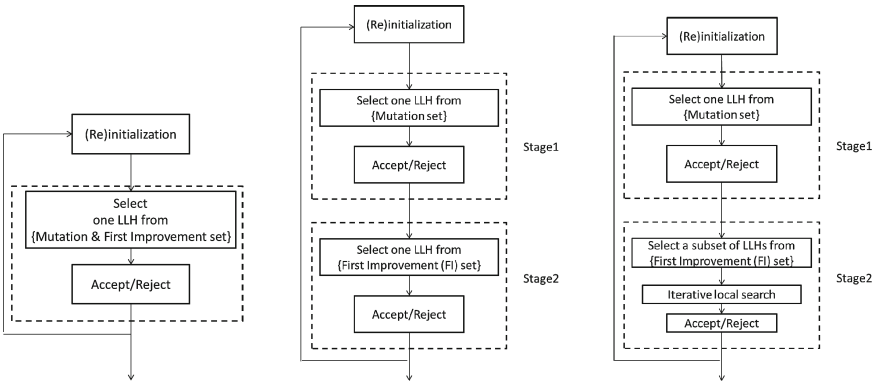
LLH Repository Settings. The operators introduced in Sect. 3 are classified according to whether we use them for mutation and/or FI, Table 1. Route related operators are parametrized by route ID, day, length of chain and number of points changed in one move; this makes it possible for an intelligent hyperheuristic to select a LLH specifically related to each sub-problem (e.g. daily

Table 1. LLH Repository used in our PVRP hyperheuristics

Type	Operators
Route related: mutation	2 points swap (2PS), Relocate, Cross
Route related: FI	2Opt, 3Opt, 2PS, Relocate, Cross
Pattern related: mutation	Random pattern reassign (Pa_RR),
	Score based reassign (Pa_SR),
	Two points pattern swap (Pa_2SW)
Pattern related: FI	Pattern reassign first improvement (Pa_FIR),
	Two points pattern swap (Pa_2SW)
Mixed: FI	Relocate with pattern (MRPa),
	Cross with pattern (MCPa)

VRP or single route optimization). Pattern related operators reassign the patterns of n customers; we consider $n = 1, 2, \ldots, 6$ in our experiments. Because of the structure of LLH parameter design, our LLH repository contains 70 to 110 LLHs, depending on the problem instance.

Algorithm Frameworks. To test LLH management strategies, we use three hyperheuristic frameworks, Fig. 1. The only difference between the first two frameworks is the strategy used to organize different types of LLHs (mutation and FI). Both the simple hyperheuristics and learning based hyperheuristics (Sect. 4) can be applied in frameworks 1 and 2. The third framework supports a VNS-based method (Sect. 4.3). Compared to framework 2, it replaces the second stage (a single selection) with an ILS over a subset of the LLH repository.



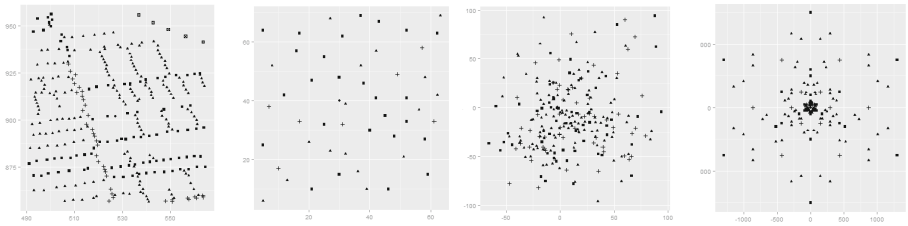
(a) Framework 1 (b) Framework 2 (c) Framework 3

Fig. 1. Hyperheuristic frameworks (the first two are modified from [23])

5.1 Problem Instance

Our data comprises 42 benchmark problems (summarised by [5]) and six instances from a real-world periodic maintenance problem[1]. We classify the problems according to their spatial characteristics (Fig. 2). Table 2 summarises each class. The six real-world instances are all street type. The big random benchmark problems have both a larger number of customers and greater clustering of data points than the small random class.

[1] The real-world data and associated best-performance results (Sect. 6) can be found at https://www-users.cs.york.ac.uk/~yujiec/.



(a) street style (real world example)

(b) Small random (from benchmark p02 [6])

(c) Big random (from benchmark pr05 [8])

(d) Symmetrical (From benchmark p32 [7])

Fig. 2. Examples of four types of spatial distributions in the PVRP instance set.

Table 2. PVRP instances. n is number of customers; m is number of vehicles; t is length of planning period in days. Benchmark labelling is from [5].

Class	n	m	t	Average visit frequency	Number of problem instances
Street style	240–324	3–5	6	1.6-2.1	6
Small random	50–100	1–6	2–10	1–2.1	10 (benchmark p01-p10)
Big random	48–417	2–12	4–7	1.1–3	13 (benchmark p11-p13, pr01-pr10)
Symmetrical	20–184	2–9	4–6	1.8-2	19 (benchmark p14-p32)

6 Experimental Results and Analysis

6.1 Random Vs Learning Based Selection Strategies

A hyperheuristic needs an efficient selection strategy, because it is impractical to apply all LLHs exhaustively. The first experiment compares the SR selection strategy to the learning based strategies, RL, CF and BEBO. The experiment uses framework 1 (Fig. 1a) and OI acceptance. For RL and CF, we test using both the best 30 % and the best 80 % of LLHs in each iteration (See Sect. 4.2).

Each selection strategy is run 20 times on each instance of each of the four classes of problem, to give the percentage differences to the best-found benchmark route length of each instance. We then average the results for each class of problems.

The results in Table 3 show that, whilst acceptable, none of our solutions matches the best-found benchmark solution. Learning based selection strategies consistently out-perform SR LLH selection, with BEBO performing best. For both RL and CF, the limited CPU time makes it difficult for the hyperheuristics to produce competitive results for $w = 80$. In subsequent experiments we only use the best 30 % of LLHs.

Table 3. The average percentage difference to the best found solution over all instances in each group, for simple random (SR) and learning based hyperheuristics, using framework 1 and OI acceptance.

	SR	RL(30 %)	CF(30 %)	RL(80 %)	CF(80 %)	BEBO
Street style	+8.90	+5.36	+6.08	+6.47	+6.64	+4.90
Small random	+4.67	+2.12	+2.28	+2.15	+2.33	+2.06
Big random	+4.32	+4.14	+4.32	+4.28	+4.35	+4.13
Symmetrical	+2.74	+1.46	+1.56	+1.55	+1.53	+1.50

6.2 Impact of Algorithm Framework

Having shown that learning based selection strategies can manage a large number of LLHs in a simple hyperheuristic framework, we now consider the different hyperheuristic frameworks.

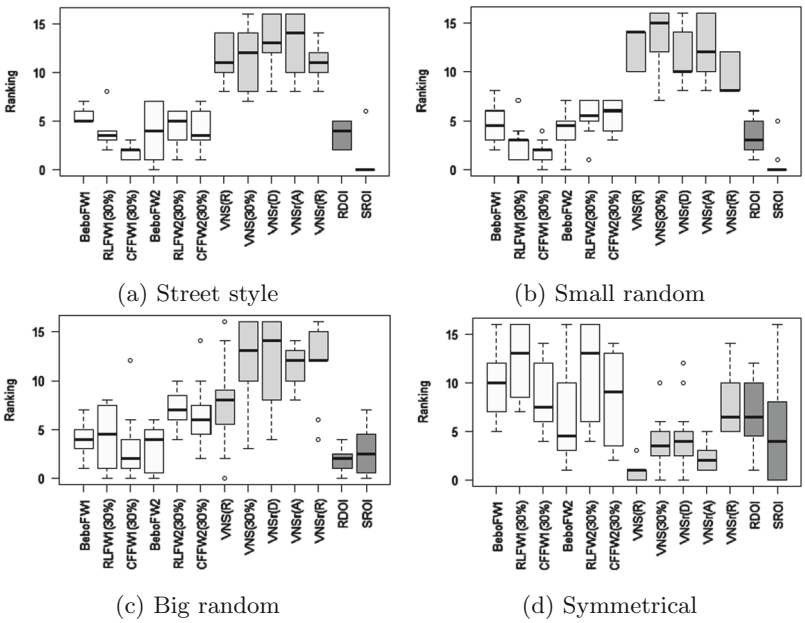
In framework 1, the OI acceptance rule means that mutation LLHs are unlikely to be favoured. In framework 2, a mutation operator is randomly selected, and is applied as long as it generates valid solutions, then FI LLHs are selected using a learning based strategy, as above. Framework 2 is similar to framework 1 when we use the all-move-accept rule, but, whereas framework 1 evaluates the mutation and FI operators together, framework 2 allows separate consideration.

The results in Table 4 show that framework 2 improves the performance of both RL and CF hyperheuristics for all types of problem instances, and, BEBO does not show obvious difference between framework 1 and 2.

Framework 3 uses the VNS-based algorithms; the main difference to framework 2 lies in the use of ILS once a FI operator is selected. Five variants are tested. The first two use GVNS (Algorithm 2), with VND using, respectively, randomly ordered FI LLHs (VNS(R)) and the best 30 % of FI LLHs (VNS(30 %)). Three variants use GVNSr (Algorithm 3), with random, ascending or descending FI LLH ordering determined using utility (respectively, VNSr(R), VNSr(A), VNSr(D)). We compare performance with the above three framework 1 and

Table 4. The average percentage differences to the best found solutions over all instances in each group, for learning based hyperheuristics using frameworks 1 and 2 (FW1, FW2)

Instances	RL(30 %)		CF(30 %)		BEBO	
	FW1	FW2	FW1	FW2	FW1	FW2
Street style	+5.36	**+5.26**	+6.08	**+5.54**	**+4.90**	+5.31
Small random	+2.19	**+1.88**	+2.28	**+1.90**	+2.06	**+2.03**
Big random	+4.14	**+3.93**	+4.32	**+3.88**	+4.13	**+4.12**
Symmetrical	+1.46	**+1.45**	+1.56	**+1.54**	**+1.50**	+1.56



(a) Street style (b) Small random

(c) Big random (d) Symmetrical

Fig. 3. Ranking of hyperheuristics for PVRPs. Higher rank is better.

three framework 2 strategies, plus random descent (RDOI) and simple random (SROI) embedded in framework 1. We then rank the performance of the 12 combinations of framework and LLH selection strategy, awarding 16 points to the best performing hyperheuristic, then $14, 12, 10, 8, 7, \ldots 1, 0$ points successively to worse performing hyperheuristics.

Figure 3 shows a small difference in performance between frameworks 1 and 2. Compared to the framework 3 results, they are both generally low-ranking for all cases except the symmetrical benchmark problems. This shows the positive impact of using ILS. Among framework 3, the five VNS-based algorithms show similar ranking, except for big random, where VNS(R) is not highly ranked; random selection of the shaking operator combined with random ordered FI LLHs (VNSr(R)) is the most robust over all classes of problem.

6.3 Scalability

The PVRP is NP-hard [15]. One of its biggest challenges is the rate of growth in complexity with problem size. In preliminary experiments, we determined that the performance of algorithms on symmetrical and non-symmetrical problems is very different. To test the scalability of our hyperheuristics, we first group the problem instances into symmetrical and non-symmetrical problems and then order them by the number of customers. Each method is runs 20 times.

Figure 4 shows that SROI has the worst scalability in both symmetrical and non-symmetrical problems. For the other algorithms, there is little difference in

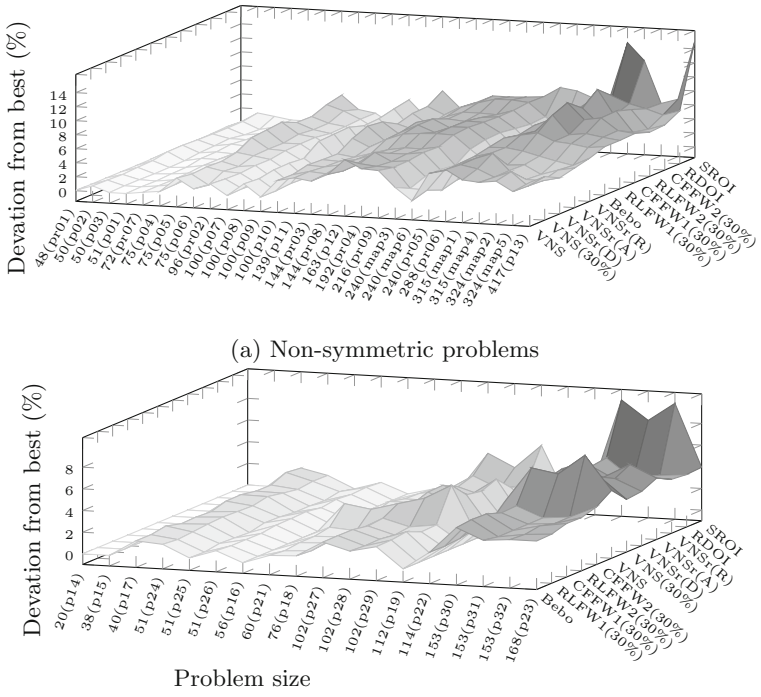


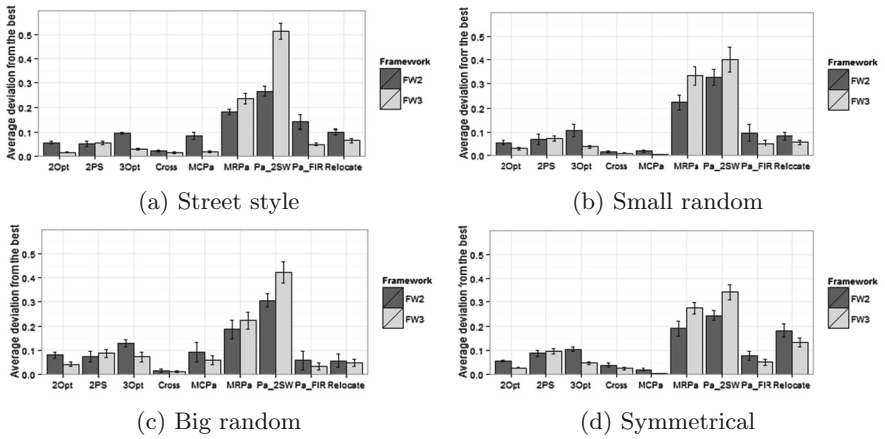
Fig. 4. Performance of hyperheuristics tested on PVRP with various sizes

performance on problems with fewer than about 60 customers. VNS-based algorithms are the most robust across non-symmetrical instances with 150 to 420 customers. However, performance decreases dramatically for VNS-based algorithms applied to bigger problem instances in the symmetrical data set.

6.4 LLH Usage Analysis

Whilst hyperheuristics need little specialised design, the LLH repository does need thought. In this experiment, we explore the usage of LLHs by the different hyperheuristics. We use frameworks 2 and 3, which manage the mutation and FI operators separately. The results focus on the 9 FI LLHs, since there is no learning in mutation operator selection.

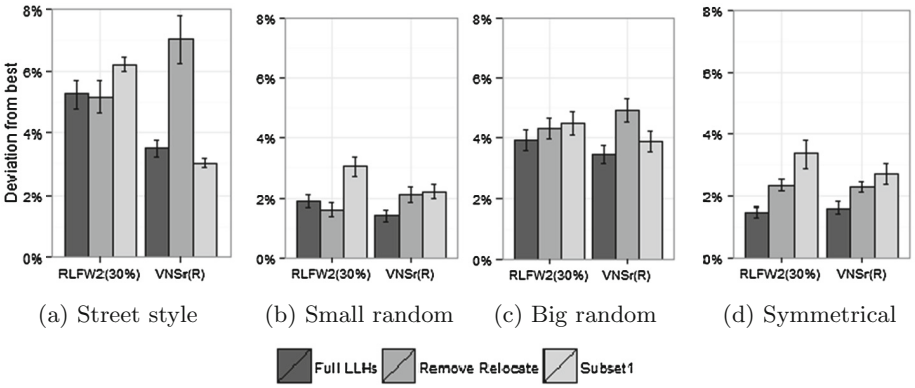
Figure 5 summarises average usage of FI LLHs for all learning based algorithms using framework 2 (BeboFW2, RLFW2(30 %), CFFW2(30 %)) and all VNS-based algorithms using framework 3 (VNS(R), VNS(30 %), VNSr(D), VNSr(A), VNDr(R)). The stronger LLHs are favoured more in framework 3 than in framework 2. "Relocate with pattern" (MRPa) and "two points pattern swap" (Pa_2SW) are the most applied LLHs by all hyperheuristics. Since we



(a) Street style (b) Small random

(c) Big random (d) Symmetrical

Fig. 5. FI LLHs usage for different types of problem. Results show the mean value of percentage of each LLH are applied during the search, where 0.1 stands for 10 %. Error bars show 95 % confidence interval.



(a) Street style (b) Small random (c) Big random (d) Symmetrical

Fig. 6. Performance of RLFW2(30 %) and VNSr(R) using different subset of LLHs. Error bars show 95 % confident interval. The subset1 removes the most used LLH (Pa_2SW) and all mutation operators except Pa_RR

are using an OI strategy, this implies that they consistently produce improved solutions.

The "Relocate" operator is preferred in symmetrical problems, but not in other instances. The importance of this operator is emphasised by the big reduction in performance when the "relocate" operator is removed (Fig. 6d).

To further explore the contribution of specific LLHs in improving PVRP solutions, we test the two best performing hyperheuristics for frameworks 2 and 3 (RLFW2(30 %) and VNSr(R)) with different subsets of the original LLHs. Each method is run over all problem instances; results are the average of 20 runs.

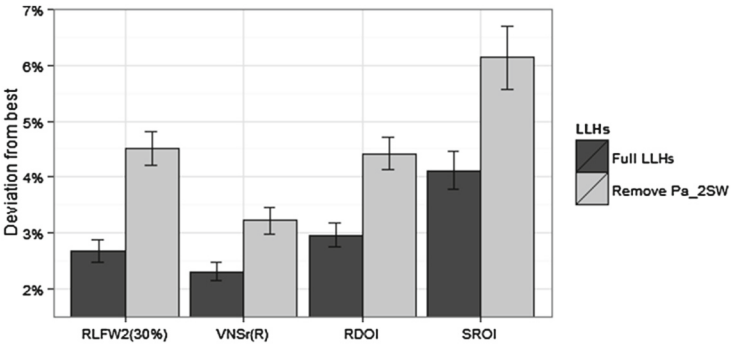


Fig. 7. Impact of removing Pa_2SW on four hyperheuristics over all problem instances. Error bars show 95 % confidence intervals.

Figure 6 shows a change in performance after removal of the most-used FI LLH (Pa_2SW) and all mutation operators except Pa_RR (Subset 1): the performance of RLFW2(30 %) and VNSr(R) decreases dramatically for the small random and symmetrical problems. However, there is little difference for the big random instances, and we even find a small improvement for VNSr(R) on street style problems. One interpretation of this result is that the strongly-performing FI LLHs, which are most effective in small and symmetric problems, tend to become stuck in local optima in the street style and big random problems. Further work is needed to understand why removing the "relocate" operator affects performance on street style problems more than less-structured spatial distributions.

To explore the robustness of different hyperheuristics when we remove the strongest LLH (Pa_2SW), we extend the LLH subset experiments to SROI and RDOI. VNSr(R) shows the best robustness (Fig. 7). Comparing the RLFW2(30 %) with VNSr(R) and SROI with RDOI, the algorithms with ILS mechanisms are more robust than the algorithms without ILS.

6.5 Comparison Between Hyperheuristics and Other Meta-Heuristics

This section compares the two best performing hyperheuristics from framework 2 and 3 (RLFW2(30 %) and VNSr(R)), to published meta-heuristics which have been designed or tailored for PVRP, including (parallel) tabu search [8,16], scatter search [14], VNS [5], record-to-record ILP [9] and hybrid Genetic Algorithm (GA) [15]. No comparative data exists for our street style data set.

Benchmark research uses 32 instances collected from early work on PVRP (the old data set). Cordeau [8] presents 10 additional PVRP instances (the new data set). We present our results for these two groups, because some research has not tested both groups. Table 5 reports the percentage difference in average performance from the best found (summarised in [15]) over these two data sets.

Table 5. Performance on PVRP benchmarks compared with meta-heuristics; tabu search (CGL) [8], scatter search (ALP)[14], VNS (HDH)[5], record-to-record ILP (GGW) [9], hybrid-GA (VCGLR)[15], parallel tabu search (CM) [16]

	RLFW2(30 %)		VNSr(R)		CGL	ALP	HDH	GGW	VCGLR	CM
	Avg. 20 run	Avg. (best)	Avg. 20 run	Avg. (best)	1 run	-	Avg. 10 run	-	Avg. 10 run	Avg. 10 run
Old data (%)	1.86	1.08	1.77	0.93	1.8	1.57	1.6	1.11	0.032	0.044
New data (%)	3.88	2.40	3.44	2.12	2.82	-	1.86	-	0.071	0.091

Our hyperheuristics achieve competitive results compared to the tabu search [8], scatter search [14] and VNS [5] for the "old data" set. For the relatively larger "new data" set, we achieve close to the best found solutions in most cases. The hyperheuristic approaches are about 1 % worse than these problem-specific algorithms, in terms of total route distance.

Compared to the hybrid-GA, which out performs all the other algorithms, our hyperheuristics produce routes that are about 2 % longer on average. However, hyperheuristics do not require any knowledge directly from the solution space and require minimal design effort, whereas the meta-heuristics need to be designed and tailored for each problem.

7 Conclusion

Our analysis of hyperheuristics for PVRP shows that both learning selection strategy and ILS have positive impacts on an algorithm's performance and enhance the scalability. ILS also improves the robustness of hyperheuristics when a poor LLH set is given, because ILS concentrates on a neighbourhood structure until it reaches a local optimum, whilst approaches without ILS have a wider, but shallower, exploration within the search space.

Our hyperheuristics find solutions that are almost as good as those published for meta-heuristics. Since all experiments have limited CPU time, it is possible that this is due to the hyperheuristics' additional overhead in applying search at the LLH selection level. The hyperheuristics are more adaptable to new problems: our results show that hyperheuristics can efficiently manage a large LLH set and automatically select appropriate LLHs.

The tested hyperheuristics show similar performance on real-world street style problem instances and random instances, but the symmetrical benchmarks tend to favour different strategies and LLHs. This suggests that symmetrical instances are not a good indicator of algorithm performance for real-world PVRP.

"Relocate with pattern" and "two points pattern swap" are the most applied LLHs across all PVRP hyperheuristics: these LLHs make most improvements during the search. However, experiments on LLH subsets show that a strong LLH may lead to premature local optima; further work is needed on the effect of structure in real-world problems, and on ways to measure "strong" LLHs.

For PVRP, we show that hyperheuristics perform similarly to problem-specific meta-heuristics, despite their working mechanism potentially increasing the complexity of solving a specific problem within limited time. We are working on improving hyperheuristic efficiency, and investigating whether the positive impact of learning based selection and ILS translates to other problem domains.

Acknowledgement. The authors would like to thank Gaist Solutions Ltd. for providing data. This research is part of the LSCITS project funded by the EPSRC.

References

1. Cowling, P.I., 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)
2. Bilgin, B., Özcan, E., Korkmaz, E.: An experimental study on hyper-heuristics and exam timetabling. In: Burke, E.K., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 394–412. Springer, Heidelberg (2007)
3. Remde, S., Cowling, P.I., Dahal, K., Colledge, N., Selensky, E.: An empirical study of hyperheuristics for managing very large sets of low level heuristics. J. Oper. Res. Soc. **63**(3), 392–405 (2011)
4. Kalender, M., Kheiri, A., Özcan, E., Burke, E.K.: A greedy gradient-simulated annealing selection hyper-heuristic. Soft Comput. **17**(12), 2279–2292 (2013)
5. Hemmelmayr, V.C., Doerner, K.F., Hartl, R.F.: A variable neighborhood search heuristic for periodic routing problems. Eur. J. Oper. Res. **195**(3), 791–802 (2009)
6. Christofides, N., Beasley, J.E.: The period routing problem. Networks **14**(2), 237–256 (1984)
7. Chao, I.M., Golden, B.L., Wasil, E.: An improved heuristic for the period vehicle routing problem. Networks **26**(6), 25–44 (1995)
8. Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks **30**(2), 105–119 (1997)
9. Gulczynski, D., Golden, B., Wasil, E.: The period vehicle routing problem: new heuristics and real-world variants. Transp. Res. Part E: Logistics Transp. Rev. **47**(5), 648–668 (2011)
10. Beltrami, E., Bodin, L.: Networks and vehicle routing for municipal waste collection. Networks **4**(1), 65–94 (1974)
11. Russell, R., Igo, W.: An assignment routing problem. Networks **9**(1), 1–17 (1979)
12. Tan, C.C.R., Beasley, J.E.: A heuristic algorithm for the period vehicle routing problem. J. Omega **12**(5), 497–504 (1984)
13. Russell, R.A., Gribbin, D.: A multiphase approach to the period routing problem. Networks **21**(7), 747–765 (1991)
14. Alegre, J., Laguna, M., Pacheco, J.: Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. Eur. J. Oper. Res. **179**(3), 736–746 (2007)
15. Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W.: A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. **60**(3), 611–624 (2012)
16. Cordeau, J.F., Maischberger, M.: A parallel iterated tabu search heuristic for vehicle routing problems. Comput. Oper. Res. **39**(9), 2033–2050 (2012)
17. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. **12**, 568–582 (1964)

18. Gendreau, M., Hertz, A., Laporte, G.: New insertion and post optimization procedures for the traveling salesman problem. Oper. Res. **40**(6), 1086–1095 (1992)
19. Groër, C., Golden, B., Wasil, E.: A library of local search heuristics for the vehicle routing problem. Math. Program. Comput. **2**(2), 79–101 (2010)
20. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. Metaheuristics: Computer Decision-Making, pp. 523–544. Springer, New York (2004)
21. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001)
22. Hansen, P., Mladenović, N., Moreno Pérez, J.A.: Variable neighbourhood search: methods and applications. Ann. Oper. Res. **175**(1), 367–407 (2010)
23. Özcan, E., Bilgin, B., Korkmaz, E.: A comprehensive analysis of hyper-heuristics. Intell. Data Anal. **12**(1), 3–23 (2008)

Evolutionary Algorithms for Finding Short Addition Chains: Going the Distance

Stjepan Picek[1(✉)], Carlos A. Coello Coello[2], Domagoj Jakobovic[3], and Nele Mentens[1]

[1] ESAT/COSIC and iMinds, KU Leuven, Kasteelpark Arenberg 10, Bus 2452, 3001 Leuven-Heverlee, Belgium
stjepan@computer.org
[2] Department of Computer Science, CINVESTAV-IPN, Av. IPN No. 2508, Col. San Pedro Zacatenco, Mexico D.F. 07360, Mexico
[3] Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

Abstract. The problem of finding the shortest addition chain for a given exponent is of great relevance in cryptography, but is also very difficult to solve since it is an **NP**-hard problem. In this paper, we propose a genetic algorithm with a novel representation of solutions and new crossover and mutation operators to minimize the length of the addition chains corresponding to a given exponent. We also develop a repair strategy that significantly enhances the performance of our approach. The results are compared with respect to those generated by other metaheuristics for instances of moderate size, but we also investigate values up to $2^{127} - 3$. For those instances, we were unable to find any results produced by other metaheuristics for comparison, and three additional strategies were adopted in this case to serve as benchmarks. Our results indicate that the proposed approach is a very promising alternative to deal with this problem.

Keywords: Addition chains · Cryptography · Genetic algorithms · Exponentiation

1 Introduction

Field or modular exponentiation has several important applications in error-correcting codes and cryptography. Well-known public-key cryptosystems such as Rivest-Shamir-Adleman (RSA) [1] adopt modular exponentiation. In a simplified way, modular exponentiation can be defined as the problem of finding the (unique) integer $B \in [1, \ldots, p-1]$ that satisfies:

$$B = A^c \bmod p, \tag{1}$$

where A is an integer in the range $[1, \ldots, p-1]$, c is an arbitrary positive integer and p is a large prime number. One possible way of reducing the computational

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 121–137, 2016.
DOI: 10.1007/978-3-319-30698-8_9

load of Eq. (1) is to minimize the total number of multiplications required to compute the exponentiation.

Since the exponent in Eq. (1) is additive, the problem of computing powers of the base element A can be formulated as an addition calculation, for which so-called *addition chains* are used. Informally, an addition chain for the exponent c of length l is a sequence V of positive integers $v_0 = 1, \ldots, v_l = c$, such that for each $i > 1$, $v_i = v_j + v_k$ for some j and k with $0 \leq j \leq k < i$. An addition chain provides the correct sequence of multiplications required for performing an exponentiation. Thus, given an addition chain V that computes the exponent c as indicated before, we can find $B = A^c$ by successively computing: $A, A^{v_1}, \ldots, A^{v_{l-1}}, A^c$. For example, if we want to compute A^{60}, the traditional procedure would require 60 multiplications. However, if we use instead the following addition chain: $[1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 30 \rightarrow 60]$, then only seven multiplications are required:

$$A^1; A^2 = A^1 A^1; A^4 = A^2 A^2; A^6 = A^4 A^2; A^{12} = A^6 A^6; \\ A^{24} = A^{12} A^{12}; A^{30} = A^{24} A^6; A^{60} = A^{30} A^{30}. \tag{2}$$

Thus, the length of the addition chain defines the number of multiplications required for computing the exponentiation. The aim is to find the shortest addition chain for a given exponent c (several addition chains can be produced for the same exponent). Naturally, as the exponent value grows, it becomes more difficult to find a chain that forms the exponent in a minimal number of steps.

One simple algorithm that can be used (although, in general it will not give optimal results) works in the following way. First, write the exponent in its binary representation. Then, replace each occurrence of the digit 1 with the letters "DA" and each occurrence of the digit 0 with the letter "D". After all digits are replaced, cross out the first "DA" that appears on the left. What remains represents a rule to calculate the exponent, since the letter "A" stands for addition (multiplication) and the letter "D" for doubling (squaring). If we consider again the example A^{60}, the exponent in binary representation would be "111100". After the replacement and the removal of "DA" at the left we have "DADADADD". Thus, the rule is: square, multiply, square, multiply, square, multiply, square, square $(1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 14 \rightarrow 15 \rightarrow 30 \rightarrow 60)$. This is a simple example describing *the binary method*. We can immediately observe that the binary method does not always give the shortest chain (cf. with the chain given in Eq. (2)). In fact, already for the value 15, the binary method will not produce the shortest chain [2]. However, the binary method can be generalized to some more powerful methods as presented in Sect. 2. Unfortunately, in general, the problem of finding the shortest addition chain is **NP**-hard [3]. This has motivated the use of metaheuristics to tackle this problem as indicated in Sect. 3.

Here, we propose a genetic algorithm to find short addition chains for a given exponent. Our main contributions are the following: the first one is a new representation of solutions. With that representation, we can obtain a better granularity than when using just the representation based on the values

in the addition chains. Next, we present several mutation and crossover operators designed to improve convergence. The behavior of those operators is modeled on the basis of several relevant test case scenarios as presented in Sect. 2. We then design *repair heuristics* that we believe are an integral part of the algorithm and we use several examples to justify our approach. From a more pragmatic perspective, in Sect. 5, we investigate a number of exponents that we want to obtain, whose values progress gradually from small ones up to the ones that are relevant in real-world applications. Finally, we identify a possible oversight in most of the relevant works that limits the applicability of those algorithms.

2 On Addition Chains

We start this section with basic notions about addition chains and, afterwards, we give several important results we use when designing our algorithm. Next, we briefly discuss algorithms that are commonly used to compute exponentiations. We follow the notation and theoretical results presented in "The Art of Computer Programming, Volume 2: Seminumerical Algorithms" [2]. For more detailed information about addition chains, we refer the readers to Chap. 4.6.3. "Evaluation of Powers" [2].

2.1 Theoretical Background

Definition 1. *An addition chain is a sequence of positive values starting with the value 1 and finishing with the desired exponent value n.*

Definition 2. *An addition chain is called ascending if:*

$$1 = a_0 < a_1 < a_2 < ... < a_r = n. \tag{3}$$

In this work, we focus only on ascending chains. From this point on, when we talk about addition chains, we mean ascending addition chains.

The values in the addition chain have the property that they are the sum of two values appearing previously in the chain. Formally, an addition chain is a sequence $a_0 = 1, a_1, ..., a_r = n$ where:

$$a_i = a_j + a_k, \text{ for some } k \leq j < i. \tag{4}$$

The shortest length of any valid addition chain is denoted as $l(n)$. In the length of a chain, one does not count the initial step that has a value of one.

Next, it is possible to define types of steps in the addition chain based on Eq. (4):

1. *Doubling step*; when $j = k = i - 1$. This step always gives the maximal possible value at the position i.
2. *Star step*; when j but not necessarily k equals $i - 1$.
3. *Small step*; when $log_2(a_i) = log_2(a_{i-1})$.
4. *Standard step*; when $a_i = a_j + a_k$ where $i > j > k$.

On the basis of the aforementioned steps, it is easy to infer the following conclusions: [2]:

- The first step is always a doubling step.
- A doubling step is always a star step and never a small step.
- A doubling step must be followed by a star step.

Now, we focus on the shortest addition chains. Trivially, the shortest chain for any number n must have at least $log_2(n)$ steps. To be more precise, any chain length is equal to $log_2(n)$ plus the number of small steps [2].

Let $\nu(n)$ be the number of ones in the binary representation of the exponent n. When $\nu(n) \geq 9$ then there are at least four small steps in any chain for exponent length n [4]. That statement can be also generalized with the following theorem [4]:

Theorem 1. *If* $\nu(n) \geq 2^{4 \cdot m-1} + 1$, *then* $l(n) \geq log_2(n) + m + 3$ *where* m *is a nonnegative value.*

Definition 3. *A star chain is a chain that involves only star operations.*

The minimal length of a star chain is denoted as $l^*(n)$ and it holds:

$$l(n) \leq l^*(n). \tag{5}$$

Although it seems intuitive that the shortest addition chain is also a star chain, in 1958, Walter Hansen proved that for certain large exponents n, the value of $l(n)$ is smaller than $l^*(n)$ [2]. The smallest such exponent n equals 12 509.

Albeit counterintuitive, there exist values of n for which $l(n) = l(2n)$ with the smallest example being $n = 191$. Here, both n and $2n$ have length l equal to 11. Furthermore, there exist values of n where $l(n) > l(2n)$ [5]. The smallest such n is 375 494 703 [6].

Finally, the length seems to be the most difficult to compute for one specific class of numbers: let $c(r)$ be the smallest value of n such that $l(n) = r$ [2]. Therefore, $c(r)$ is the first integer value requiring r steps in a shortest addition chain [5]. To obtain such shortest addition chains is regarded more difficult than to obtain a shortest addition chain for some other value (of course, with regards to the size).

Up to now, we discussed only ascending addition chains, but there exists a number of other types of chains, e.g. addition-subtraction chains [2], differential addition chains [7] or differential addition-subtraction chains [7].

2.2 Techniques for Exponentiation

A number of techniques that are useful for cryptography, and that apply to both exponentiation in a multiplicative group and elliptic curve point multiplication, are explained in [3,8] and can be divided into three categories:

1. techniques for general exponentiation,
2. techniques for fixed-base exponentiation and
3. techniques for fixed-exponent exponentiation.

In the following paragraphs, we use the term exponentiation, while all principles hold for both exponentiation and elliptic curve point multiplication. In the first category, the most straightforward ways to perform an exponentiation or a point multiplication, are the left-to-right and right-to-left binary methods. An option for speeding up these algorithms consists of evaluating more than one bit of the exponent at a time after precomputing a number of multiples of the base. An example is the window or k-ary method that evaluates k bits of the exponent at a time. The precomputation of base multiples maximizes the speed by minimizing the number of multiplications. However, the optimizations require a larger memory usage for the storage of the precomputed values. When the base is fixed, the precomputed multiples of the base can be prestored, which shortens the time needed for the online exponentiation.

Another way of minimizing the number of multiplications without storing precomputed multiples of the base is exponent recoding, which uses a representation of the exponent that is different from the binary representation. The recoding of the exponent requires additional resources on a chip (logic gates) or a microprocessor (program memory).

For elliptic curve cryptography, further speed optimizations are possible by considering elliptic curves with special properties, like the Gallant-Lambert-Vanstone (GLV) curve [9], the Galbraith-Lin-Scott (GLS) curve [10] or the FourQ curve [11]. In [12], side-channel security is taken into account in the derivation of efficient algorithms for scalar multiplication on GLS-GLV curves.

In this paper, we focus on addition chains for fixed-exponent exponentiations or fixed-scalar point multiplication without taking into account optimizations using specific fields or curves. We do not consider side-channel analysis, but we believe this does not undermine our results, since a number of side-channel countermeasures can be applied on top of the proposed addition chains. Examples are point blinding or randomized projective coordinates [13].

3 Related Work

In 1990, Bos and Coster present the Makesequence algorithm that produces an addition sequence of a set of numbers [14]. The proposed method is able to find chains of large dimensions, and the authors conclude that their method is relatively more effective than the binary method. The heuristics in the algorithm choose, on the basis of a weight function, which method will be used to produce the sequence (the authors experiment with four methods). However, the authors report that their current weight function does not give satisfactory results and they experiment with simulated annealing, but without success.

Nedjah and de Macedo Mourelle experiment with a genetic algorithm (GA) in order to find minimal addition chains [15]. They use binary encoding where

value 1 means that the entry number is in the chain, and 0 means the opposite. This representation is not suitable for large numbers and the authors experiment with values of only up to 250. We note that the chromosome is of length 250 for that value, and for any value of practical interest the chromosome would amount to more than the memory of all computers in the world. The same authors focus on optimizing addition-subtraction chains with GAs [16]. They use the same representation and exponent values as in [15], which makes their work also far from applicable. They also experiment with addition-subtraction chains with a maximal value of 343 in [17].

Nedjah and de Macedo Mourelle use Ant Colony Optimization to find minimal addition chains working with exponent sizes of up to 128 bits [18]. However, since they do not provide the numbers themselves, but only their sizes, it is impossible to assess the quality of this approach besides the fact that they report it is better than the binary, quaternary, and octal method. The same authors extend their work for exponent sizes up to 1 024 bits resulting in better results for the Ant Colony Optimization algorithm than in cases when binary, quaternary, octal, and GA methods are used [19].

Cruz-Cortés et al. propose a genetic algorithm approach for which the encoding is the chain itself [20]. Besides that, the authors also propose dedicated mutation and crossover operators. Using this approach, they report to successfully find minimal addition chains for numbers up to 14 143 037. Cortés, Rodríguez-Henríquez, and Coello present an Artificial Immune System for generating short addition chains of sizes up to 14 143 037 [21]. With that approach, the authors were successful in finding almost all optimal addition chains for exponents $e < 4\,096$.

Osorio-Hernáandez et al. [22] propose a genetic algorithm coupled with a local search algorithm and repair mechanism in order to find minimal short addition chains. This work is of high relevance since it clearly discusses the need for a repair mechanism when using heuristics for the addition chains problem.

León-Javier et al. [23] experiment with the Particle Swarm Optimization algorithm in order to find optimal short addition chains. Nedjah and de Macedo Mourelle [24] implement the Ant Colony Optimization algorithm on a SoC in order to speed up the modular exponentiation in cryptographic applications. Sarkar and Mandal [25] use Particle Swarm Optimization to obtain faster modular multiplication in cryptographic applications for wireless communication.

Rodriguez-Cristerna and Torres-Jimenez [26] use a GA to find minimal Brauer chains where a Brauer chain is an addition chain in which each member uses the previous member as a summand. Finally, Domínguez-Isidro et al. [27,28] investigate the usage of evolutionary programming for minimizing the length of addition chains.

4 The Design of the Proposed Algorithm

Before discussing the choice of the algorithm, we briefly enumerate some basic rules our chains need to fulfill:

1. Every chain (solution) needs to be an ascending chain.
2. Every chain needs to be non-redundant, i.e., there should not be two identical numbers in a chain.
3. Every chain needs to be valid, i.e., every number in a chain needs to be a sum of two previously appearing numbers.
4. Every chain needs to start with the value 1 and finish with the desired exponent value.

When choosing the appropriate algorithm for the evolution of chains, we start with the considerations about the representation. If we disregard the approach where one encodes individuals in a binary way (i.e., for each possible value, we use either zero if it is not a part of the chain, or one when it is a part of the chain), up to now there is not much of a choice. Indeed, encoding solutions as integer values where each value represents the number that occurs in the chain seems rather natural. Accordingly, we also use that representation, which we denote as encoding with *chain values*.

However, internally, our algorithm works with one more representation where we represent each value n as a pair of positions i_1 and i_2 that hold the previous values n_1 and n_2 forming the value n, which is denoted as encoding with *summand positions*.

Although such position based encoding gives longer chromosomes, for large exponents the encoded values are much smaller and the memory requirements for storing an individual are consequently smaller. Furthermore, it is possible to use operators that work on the positions and to give an algorithm more options to combine solutions (since we have two positions for every number, the length of a chain encoded with positions is always twice as long as the one encoded with values).

For both representations, a GA seems a natural choice, but there is one important difference in both approaches. When using the representation based on chain values for large numbers, the chromosome encoding needs to support large numbers, while in the representation based on summand positions we only need to support large numbers for calculating the chain elements, but not for storing them.

However, one cannot aim to fulfill the aforementioned rules and use a standard GA. Therefore, we need to design a custom initialization procedure, mutation, and crossover operators. In fact, only the selection algorithm can be used as in the standard GA. In all our experiments, we work with k-tournament selection where $k = 3$. In each tournament, the worst of k randomly selected individuals is replaced by the offspring of the best two from the same tournament.

Since initialization and variation operators are expected to produce many invalid solutions (in fact, for larger chains our experiments showed that it is highly unlikely that genetic operators will produce valid solutions) we also need to design a repair strategy. The repair strategy can be incorporated in each of the previous parts or to be considered as a special kind of operator, which is the approach we opted to follow. Next, we present the operators we use in our GA.

4.1 Initialization Algorithm

We design the initialization algorithm in a way to offer as much diversity as possible. We accomplish this by analyzing a number of known optimal chains (both star and standard chains) and checking the necessary steps to obtain them. Here, we note that if the initialization can produce only star chains and the mutation can generate only star steps, the whole algorithm will be able to produce only star chains. Naturally, one could circumvent this by adding additional steps in the repair mechanism. In that case, the model would not follow the intuition, since one expects that the repair mechanism only repairs the chains and it should not possess additional mechanisms for the generation of new values.

The initial population is generated via a set of hardcoded values that are positioned at the beginning of the chain and randomly generated chain sequences as presented below. The probability values are selected on the basis of a set of tuning experiments.

1. Set the zeroth element to 1 and the first element to 2.
2. Uniformly at random select between all minimal subchains consisting of three elements (i.e., the second, third, and fourth position in the chain) and a random choice of the second element (according to the rules, either the value 3 or 4).
3. With a probability equal to 3/5, double the elements until they reach half of the exponent size.
4. Check whether the current element and any previous element sum up to the exponent value.
5. Uniformly at random, choose among the following mechanisms to obtain the next value in the chain, under the constraint that it needs to be smaller than the exponent value:
   (a) Sum two preceding elements of the chain.
   (b) Sum the previous element and a random element.
   (c) Sum two random elements. One random element is chosen between the zeroth position and the element in the middle of the chain and the second one is chosen between the middle element and the final (exponent) value.
   (d) Loop from the element on the position $i-1$ until the largest element that can be summed up with the last element is found.

4.2 Variation Operators

Next, we present the mutation and crossover operators we use. They are very similar to the operators provided, for instance, in [20,21]. For such a specific problem as the one we study here, the task of devising new operators is difficult. Furthermore, many operators reduce to the ones described here. For instance, we present here something that is analogous to a single-point mutation, but since the change in a single position will invalidate the chain, after the repair mechanism, the mutation can also be regarded as a mixed mutation. Therefore,

the number of mutation points is irrelevant since a single point change brings changes in every position until the end of the chain.

Since we have several branches in the mutation operator, one can say that those branches could be separated into different mutation operators. We note that there are more possibilities on how to combine two values to form a new value in a sequence and there could be possibilities for additional mutation operators.

On the other hand, we implemented two crossover operators and we consider advantageous to use both of them, since this promotes diversity. However, identifying which of them is better than the other is hard, since it depends on the exponent value that we aim to reach.

Crossover. We implemented two versions on the crossover operator: one-point crossover and two-point crossover. We provide the pseudocode for one-point crossover in Algorithm 1 and note that the two-point version is analogous. Here, the function $FindLowestPair(P, i, pair_1, pair_2)$ determines the pair of elements with lowest indexes $(pair_1, pair_2)$ which give the target element i in a chain P. The dominant difference between the mutation operator and the crossover operator lies in the fact that in the crossover, we have defined the rules on how to build elements while in the mutation we do not have such strict rules. However, since both require the usage of the repair mechanism, that difference can become rather fuzzy.

Algorithm 1. Crossover operator.

Require: Exponent $exp > 0$, *Parent addition chains* P_1, P_2
  $rand = random(3, exp - 1)$
  for all i such that $0 \leq i \leq rand$ **do**
    $e_i = P_{1_i}$
  end for
  for all i such that $rand \leq i + 1 \leq n$ **do**
    $FindLowestPair(P_2, i, pair_1, pair_2)$
    $e_i = e_{pair_1} + e_{pair_2}$
  end for
  $RepairChain(e, exp)$
  return $e = e_0, e_1, ..., e_n$

Mutation. The mutation operator is again similar to those presented in the related literature, but we allow more diversity in the generation process as presented in Algorithm 2. As already stated, since the mutation invalidates the chain, it is impossible to expect small changes (except when the mutation point is at the end of the chain) and therefore, this is actually a macromutation operator.

Algorithm 2. Mutation operator.

Require: Exponent $exp > 0, e = e_0, e_1, ..., e_n$
  $rand = random(2, exp - 1)$
  $rand_2 = random(0, 1)$
  if $rand_2$ **then**
    $e_{rand} = e_{rand-1} + e_{rand-2}$
  else
    $rand_3 = random(2, rand - 1)$
    $e_{rand} = e_{rand-1} + e_{rand_3}$
  end if
  $RepairChain(e, exp)$
  return $e = e_0, e_1, ..., e_n$

4.3 The Repair Algorithm

Function $RepairChain(e, exp)$ takes the chain e and repairs it in the following way:

1. Delete duplicate elements in the chain.
2. Delete elements greater than the exp value.
3. Check that all elements are in ascending order, if not, sort them.
4. Ensure that the chain finishes with the exp value by repeating operations in the following order:
   (a) Try to find two elements in the chain that result in exp.
   (b) Uniformly at random apply:
      i. Double the last element of the chain while it is smaller than exp.
      ii. Add the last element and a random element.
      iii. Add two random elements.

This function is in many ways similar to the Initialization procedure, but here with the primary goal of removing redundant chain elements, rather than maximizing diversity as is the case in the Initialization.

There are several places in our algorithm where we choose what branch to enter based on random values. We decided to use uniform random values where each branch has the same probability to be chosen. We believe this mechanism can be further improved. One trivial modification would be with regards to whether one wants to obtain a star chain or not. In the case when only star chains are wanted, then the branches that cannot result in a star step can be set either to a zero or some small value, analogous for the case when we want to have a larger number of standard steps.

4.4 The Fitness Function

We use a simple fitness function where the goal is *minimization*. The number of elements in the chain is minimized as given by the equation:

$$fitness = l(n). \tag{6}$$

5 Results and Discussion

5.1 Experimental Setup

The number of independent runs for each experiment is 50. For the stopping criterion we use stagnation which we set to 100 generations without improvement. We set the total number of generations to 1 500. The population size is set to 300 in all experiments. We note that larger population sizes perform even better thanks to increased diversity from the initialization mechanism, but for large exponent values the evolution then takes a long time. With the current setting, even for larger exponent size, one evolutionary run finishes in less than one hour.

5.2 Results

When discussing the efficiency of our algorithm, we need to establish a number of test cases that will:

1. serve as a comparison with previous work,
2. serve as special test cases and
3. serve as real-world benchmark tests.

Tests Based on a Comparison with Previous Work
For the first category, we used a set of exponent values that are also used in previous work. Namely, those are the exponents belonging to the class that is the most difficult to calculate according to [2]. Recall, those values are the minimal integers that form an addition chain of a certain length i. Up to now, experiments were done for values of i up to 30 [20,21]. However, we wanted to evaluate the performance of our algorithm with even higher values and, therefore, we experimented with values up to $i = 40$. Furthermore, for each of those values we give statistical indicators in order to understand better the performance of our algorithm as well as to serve as a reference for future work.

Any comparison with previous work is difficult since it only reports the value (and the chain) that presents the best obtained solution. From the reproducibility and the efficiency side, we find that approach somewhat incomplete since it makes a difference if the algorithm found the best possible value in one instance out of 100 runs or in 90 instances out of 100 runs.

We note that for exponent values $n < 2^{27}$ one can find optimal chains online [6], while values of up to $n = 2^{31}$ can be obtained from the same web page. Therefore, in a sense, we conclude it is easy to compare with all values up to 2^{31} and we do not investigate such cases any further. However, as n increases, the situation changes since it becomes difficult to find any results for a direct comparison. Therefore, besides our algorithm, we implement the binary algorithm as well as two variants of the window method. In the first m-window method, we set the value of k to four in the expression $m = 2^k$. It has been shown [5] that with this method the length of the chain is:

$$l(n) \leq log_2(n) + 2^{k-1} - (k-1) + [log_2(n)/k], \forall k. \tag{7}$$

The second version of the window method tries to optimize Eq. (7) by choosing the value k that minimizes $2^{k-1} - (k-1) + [log_2(n)/k]$. We emphasize that none of the aforementioned methods should be regarded as the state-of-the-art, but as methods that give good results and should serve as the baseline cases. For smaller values of the exponent, the first window method gives far worse results than even the binary method and therefore we do not present such solutions. We omit the results for the first five values of r where the exponent value $c(r)$ is smaller than 10 since it is trivial to find optimal values in this case (recall Sect. 1 where we stated that the value 15 is the first exponent where the binary method is not the optimal choice). Additionally, the initialization part of our algorithm has all optimal combinations for the first five exponents hardcoded and therefore the comparison is not fair. The results are given in Table 1 where it is easy to observe that the GA performs better than the binary and optimized window methods.

Special Test Cases

Tests constituting special cases deal with the theoretical results we enumerated in Sect. 2. Here, we test the smallest value where $l(n) = l(2n)$ which is 191. Next, we test the smallest number n where the optimal chain is not a star chain, which is 12 509. We present the values that form the chain since it is interesting to observe several things. The smallest chain is $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 13 \rightarrow 24 \rightarrow 48 \rightarrow 96 \rightarrow 192 \rightarrow 384 \rightarrow 768 \rightarrow 781 \rightarrow 1\,562 \rightarrow 3\,124 \rightarrow 6\,248 \rightarrow 12\,496 \rightarrow 12\,509$. Note that this is not the only combination giving this chain of shortest length, but the following observations hold for others. Here, we are interested in values $12 \rightarrow 13 \rightarrow 24$, which is the part that does not follow the rules of a star chain.

If we compare this sequence with those obtainable from related work (cf. [20,21]), we notice that in those approaches there exist no steps that can produce such a sequence. Therefore, although related work presented heuristic algorithms that are good on selected test cases, we show that they would not work for this case and therefore are not general enough for every addition chain, but only for star chains. The final special test case is the number $n = 375\,494\,703$ since $l(n) = 35$ while $l(2n) = 34$. Results for all special cases are given in Table 2. As in the first set of experiments, the GA approach again easily outperforms the binary and optimized window methods.

Real-World Benchmark Tests

As a real-world benchmark, we investigate values up to $2^{127} - 3$. We select that upper limit since it has applications in certain high speed Diffie-Hellman implementations [29]. To provide additional experiments for a comparison, we start with a value $2^{37} - 3$ and we progress by increasing the exponent in steps of ten, i.e., the following value is $2^{47} - 3$. We finish the experiments with the exponent $2^{127} - 3$ $(170\,141\,183\,460\,469\,231\,731\,687\,303\,715\,884\,105\,725)$. We also present the results for the window method with a fixed value of k ($k = 4$) since it produces better results than the binary method. The results are given in Table 3. Similarly as in the previous cases, the GA approach is again superior while the differences between the results are even more striking than before.

Table 1. c(r) family of the exponent values.

r	c(r)	Binary	Optimized window	GA		
				Min	Avg	Stdev
5	11	5	5	5	5	0
6	19	6	11	6	6	0
7	29	7	11	7	7	0
8	47	9	12	8	8	0
9	71	9	13	9	9	0
10	127	12	13	10	10	0
11	191	13	14	11	11	0
12	379	14	16	12	12	0
13	607	15	17	13	13	0
14	1 087	16	18	14	14	0
15	1 903	18	18	15	15	0
16	3 583	21	19	16	16	0
17	6 271	20	20	17	17	0
18	11 231	23	22	18	18	0
19	18 287	23	23	19	19	0
20	34 303	25	24	20	20	0
21	65 131	26	24	21	21	0
22	110 591	30	25	22	22.08	0.27
23	196 591	32	27	23	23.04	0.19
24	357 887	32	28	24	24.28	1.26
25	685 951	33	29	25	25.1	0.58
26	1 176 431	33	31	26	26.18	1.27
27	2 211 837	36	32	27	27.18	1.68
28	4 169 527	37	32	28	28.18	0.38
29	7 624 319	36	33	29	30.16	0.71
30	14 143 037	38	34	30	30.92	0.6
31	25 450 463	38	35	31	32.62	0.66
32	46 444 543	42	36	32	33.5	0.54
33	89 209 343	42	38	33	34.46	0.81
34	155 691 199	42	39	34	35.44	1.03
35	298 695 487	46	41	35	35.67	0.74
36	550 040 063	45	41	36	37.96	0.83
37	994 660 991	46	42	37	38.76	1.47
38	1 886 023 151	48	42	38	40.28	1.21
39	3 502 562 143	48	43	39	41.36	1.19
40	6 490 123 999	52	45	41	41.77	0.63

Table 2. Special test cases.

n	l(n)	Binary	Optimized window	GA		
				Min	Avg	Stdev
191	11	13	14	11	11	0
382	11	14	16	11	11.1	0.3
12 509	17	20	21	17	17.96	0.19
375 494 703	35	41	40	35	36.36	0.87
750 989 406	34	42	40	34	36.56	0.81

Table 3. Exponents up to $2^{127} - 3$.

Exponent	$log_2(n)$	$\nu(n)$	Binary	Window	Optimized win.	GA		
						Min	Avg	Stdev
$2^{37} - 3$	36	35	71	57	51	43	45.32	0.99
$2^{47} - 3$	46	45	91	69	63	54	56.25	1.11
$2^{57} - 3$	56	55	111	82	76	64	64.9	0.87
$2^{67} - 3$	66	65	131	94	88	73	73.2	0.43
$2^{77} - 3$	76	75	151	107	101	85	85.4	0.51
$2^{87} - 3$	86	85	171	119	113	97	104.3	3.56
$2^{97} - 3$	96	95	191	132	126	106	107.2	0.91
$2^{107} - 3$	106	105	211	144	138	115	115.71	0.75
$2^{117} - 3$	116	115	231	157	151	126	126.6	0.89
$2^{127} - 3$	126	125	251	169	163	136	136.8	0.83

We note that the shortest known chain for the exponent value $2^{127}-3$ has 136 elements, which is the same value our algorithm reached. The question is whether this should be regarded as a success or a failure. In a sense, it depends on the perspective; if one knows that the value 136 was obtained (somewhat surprising) by a pen and paper approach in a matter of a few hours by an expert, then our result does not seem impressive. However, recall Definition 1 where it is easy to calculate that $l(2^{127} - 3)$ has a chain of a length at least equal to 130 since this exponent has 125 ones in its binary representation. On the other hand, the GA found the chain of the same length without any problems and in less than 30 min on average. Furthermore, maybe there are no shorter chains for that exponent, so the GA actually reaches the optimal value. Unfortunately, the answer to this question seems out of our reach without some new analytical breakthrough or until the processing power increases sufficiently to run an exhaustive search. Since both of those perspectives are unlikely at this moment, we consider our algorithm useful since it gives us an option to effortlessly find many short chains for a wide range of exponent values.

6 Conclusions and Future Work

In this work, we showed that GAs can be used to find shortest addition chains for a wide set of exponent sizes. However, we note this problem is not as easy as could be perceived from a number of related works. Indeed, the first step is the design of a custom GA and then one needs to carefully tune the parameters. Here, we managed to find chains that are either optimal (where it was possible to confirm based on related work) or as short as possible for a number of values. From that perspective, we see this work also as a reference work against which new heuristics should be tested, since it is undoubtedly possible to compare the results. As far as we know, we are the first to investigate this kind of heuristics for an exponent value that has a real world usage.

As part of our future work we plan to investigate even larger values that are useful in practice. We also note that our position based representation actually corresponds to the Cartesian Genetic Programming (CGP) encoding. There, we always use one function (plus) and for each node the indexes from the two previous nodes are recorded, which can be encoded as a graph of size $1 \times N$, which motivates us to experiment with CGP in the future.

Acknowledgments. This work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882. The second author gratefully acknowledges support from CONACyT project no. 221551. In addition, this work was supported in part by the Research Council KU Leuven (C16/15/058) and IOF project EDA-DSE (HB/13/020).

References

1. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
2. Knuth, D.E.: The Art of Computer Programming: Seminumerical Algorithms, vol. 2, 3rd edn. Addison-Wesley Longman Publishing, Boston (1997)
3. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
4. Thurber, E.G.: The scholz-brauer problem on addition chains. Pac. J. Math. **49**(1), 229–242 (1973)
5. Thurber, E.G.: On addition chains $1(mn) \leq 1(n) - b$ and lower bounds for $c(r)$. Duke Math. J. **40**(4), 907–913 (1973)
6. Flammenkamp, A.: Shortest addition chains (November 2015). http://wwwhomes.uni-bielefeld.de/achim/addition_chain.html
7. Bernstein, D.J.: Differential addition chains (2006). https://cr.yp.to/ecdh/diffchain-20060219.pdf
8. Gordon, D.M.: A survey of fast exponentiation methods. J. Algorithms **27**, 129–146 (1998)
9. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001)

10. Galbraith, S., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. J. Cryptol. **24**(3), 446–469 (2011)
11. Costello, C., Longa, P.: FourQ: four-dimensional decompositions on a Q-curve over the Mersenne prime. Cryptology ePrint Archive, Report 2015/565 (2015). http://eprint.iacr.org/
12. Faz-Hernández, A., Longa, P., Sánchez, A.H.: Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 1–27. Springer, Heidelberg (2014)
13. Coron, J.-S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Ko, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
14. Bos, J.N.E., Coster, M.J.: Addition chain heuristics. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 400–407. Springer, Heidelberg (1990)
15. Nedjah, N., de Macedo Mourelle, L.: Minimal addition chain for efficient modular exponentiation using genetic algorithms. In: Hendtlass, T., Ali, M. (eds.) IEA/AIE 2002. LNCS (LNAI), vol. 2358, p. 88. Springer, Heidelberg (2002)
16. Nedjah, N., de Macedo Mourelle, L.: Minimal addition-subtraction chains using genetic algorithms. In: Yakhno, T. (ed.) ADVIS 2002. LNCS, vol. 2457, pp. 303–313. Springer, Heidelberg (2002)
17. Nedjah, N., de Macedo Mourelle, L.: Minimal addition-subtraction sequences for efficient pre-processing in large window-based modular exponentiation using genetic algorithms. In: Liu, J., Cheung, Y.M., Yin, H. (eds.) IDEAL 2003. LNCS, vol. 2690, pp. 329–336. Springer, Heidelberg (2003)
18. Nedjah, N., de Macedo Mourelle, L.: Finding minimal addition chains using ant colony. In: Yang, Z.R., Yin, H., Everson, R.M. (eds.) IDEAL 2004. LNCS, vol. 3177, pp. 642–647. Springer, Heidelberg (2004)
19. Nedjah, N., de Macedo Mourelle, L.: Towards minimal addition chains using ant colony optimisation. J. Math. Model. Algorithms **5**(4), 525–543 (2006)
20. Cruz-Cortés, N., Rodríguez-Henríquez, F., Juárez-Morales, R., Coello Coello, C.A.: Finding optimal addition chains using a genetic algorithm approach. In: Hao, Y., Liu, J., Wang, Y.-P., Cheung, Y., Yin, H., Jiao, L., Ma, J., Jiao, Y.-C. (eds.) CIS 2005. LNCS (LNAI), vol. 3801, pp. 208–215. Springer, Heidelberg (2005)
21. Cruz-Corteés, N., Rodriguez-Henriquez, F., Coello Coello, C.: An artificial immune system heuristic for generating short addition chains. IEEE Trans. Evol. Comput. **12**(1), 1–24 (2008)
22. Osorio-Hernández, L.G., Mezura-Montes, E., Cortés, N.C., Rodríguez-Henríquez, F.: A genetic algorithm with repair and local search mechanisms able to find minimal length addition chains for small exponents. In: Proceedings of IEEE Congress on Evolutionary Computation, Trondheim, Norway, pp. 1422–1429, 18–21 May 2009
23. León-Javier, A., Cruz-Cortés, N., Moreno-Armendáriz, M.A., Orantes-Jiménez, S.: Finding minimal addition chains with a particle swarm optimization algorithm. In: Aguirre, A.H., Borja, R.M., Garciá, C.A.R. (eds.) MICAI 2009. LNCS, vol. 5845, pp. 680–691. Springer, Heidelberg (2009)
24. Nedjah, N., de Macedo Mourelle, L.: High-performance SoC-based implementation of modular exponentiation using evolutionary addition chains for efficient cryptography. Appl. Soft Comput. **11**(7), 4302–4311 (2011)
25. Sarkar, A., Mandal, J.: Swarm intelligence based faster public-key cryptography in wireless communication (SIFPKC). Int. J. Comput. Sci. Eng. Technol. (IJCSET) **7**, 267–273 (2012)

26. Rodriguez-Cristerna, A., Torres-Jimenez, J.: A genetic algorithm for the problem of minimal brauer chains. In: Castillo, O., Melin, P., Kacprzyk, J. (eds.) RAHIS 2013. SCI, vol. 451, pp. 481–500. Springer, Heidelberg (2013)
27. Domínguez-Isidro, S., Mezura-Montes, E., Osorio-Hernández, L.G.: Addition chain length minimization with evolutionary programming. In: 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Companion Material Proceedings, Dublin, Ireland, pp. 59–60, 12–16 July 2011
28. Domínguez-Isidro, S., Mezura-Montes, E., Osorio-Hernández, L.G.: Evolutionary programming for the length minimization of addition chains. Eng. Appl. AI **37**, 125–134 (2015)
29. Bernstein, D.J., Chuengsatiansup, C., Lange, T., Schwabe, P.: Kummer strikes back: new DH speed records. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 317–337. Springer, Heidelberg (2014)

Experimental Evaluation of Two Approaches to Optimal Recombination for Permutation Problems

Anton V. Eremeev[1] and Julia V. Kovalenko[2(✉)]

[1] Omsk Branch of Sobolev Institute of Mathematics,
13 Pevtsov Str., 644043 Omsk, Russia
eremeev@ofim.oscsbras.ru
[2] Sobolev Institute of Mathematics, 4, Akad. Koptyug Avenue,
630090 Novosibirsk, Russia
julia.kovalenko.ya@yandex.ru

Abstract. We consider two approaches to formulation and solving of optimal recombination problems arising as supplementary problems in genetic algorithms for the Asymmetric Travelling Salesman Problem and the Makespan Minimization Problem on a Single Machine. All four optimal recombination problems under consideration are NP-hard but relatively fast exponential-time algorithms are known for solving them. The experimental evaluation carried out in this paper shows that the two approaches to optimal recombination are competitive with each other.

Keywords: Genetic algorithm · Optimal recombination problem · Permutation

1 Introduction

Performance of genetic algorithms (GA) depends significantly upon the choice of the crossover operator, where the components of parent solutions are combined to build the offspring. *Optimal recombination problem* (ORP) consists in finding the best possible offspring as a result of a crossover operator, given two feasible parent solutions. The ORP is a supplementary problem (usually) of smaller dimension than the original problem, formulated in view of the basic principles of crossover [21]. The experimental results of Yagiura and Ibaraki [26], Cotta et al. [6], Cook and Seymour [4], Tinós et al. [24] indicate that optimal recombination may be used successfully in the genetic algorithms for problems on permutations.

This paper is devoted to analysis and comparison of two approaches to construct the optimal recombination operators for Asymmetric Travelling Salesman Problem (ATSP) and Makespan Minimization Problem on a Single Machine $(1|s_{vu}|C_{\max})$.

The first approach is based on the preservation of elements in positions of the parent permutations. Following this approach for ATSP and $1|s_{vu}|C_{\max}$ we show that on one hand the ORP with position-based representation is strongly

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 138–153, 2016.
DOI: 10.1007/978-3-319-30698-8_10

NP-hard, on the other hand, almost all of the ORP instances are efficiently solvable. We develop an exact algorithm for solving the ORP, using enumeration of all possible combinations of the maximal matchings in cycles of a special bipartite graph. A crossover operator, where this ORP is solved by means of the proposed algorithm, is called here *Optimized Cycle Crossover* (OCX) and may be considered as a derandomization of Uniform Cycle Crossover [7].

The second approach is based on the preservation of the adjacencies found in the parents. Strong NP-hardness of the ORP with adjacency-based representation is proven and a solution method is proposed. A crossover operator, which solves this ORP, is called here *Optimized Directed Edge Crossover* (ODEC) and may be considered as a "direct descendant" of Directed Edge Crossover [25].

The theoretical worst-case and average-case upper bounds on time complexity of optimized crossovers obtained in this paper and in [11] are not sufficient to estimate efficiency of GAs based on such operators. Even if the time complexity of one optimized crossover is greater than the other, this does not necessarily mean that a GA using the slower crossover will require more time to find an optimal solution. Note that given the same pair of parent solutions, the slower crossover operator may be choosing the best offspring from a larger set of possible offspring solutions, thus giving more advantage to the GA. Therefore, a deeper analysis is required in order to decide which of the two approaches is the most appropriate for a given problem. We perform computational experiments to compare the behavior of GAs that use the optimized crossovers based on the two approaches mentioned above.

A wide spectrum of metaheuristics and heuristics has been proposed to the ATSP problem (see e.g. [1,3,20]). Many of these algorithms solve the ATSP instances from [22] very quickly. Note, however, that the present paper is aimed, first of all, at comparison of the optimized crossovers, rather than constructing fast algorithms for ATSP problem. In particular, in the computational experiments we test the optimized crossovers in a very basic GA with elitist recombination without any problem-specific local search procedures or fine tuning of parameters.

The paper is organized as follows. In Sect. 2, we provide a formal description of the Optimal Recombination Problem. Section 3 is devoted to the theoretical analysis of computational complexity of the two ORPs for Makespan Minimization Problem on a Single Machine. A similar analysis for Asymmetric Travelling Salesman Problem is provided in Sect. 4. The computational experiments are described in Sect. 5 and the concluding remarks are given in Sect. 6.

2 Optimal Recombination Problem

The genetic algorithm is a random search method that models a process of evolution of a population of *individuals* [18]. Each individual is a sample solution to the optimization problem being solved. For the sake of generality in this section we can consider combinatorial optimization problems, where the solutions are represented by strings of length n, composed of symbols from some

finite alphabet. The components of these stings are called *genes*. Individuals of a new population are built by means of variation operators (crossover and/or mutation).

Performance of the GA depends significantly upon the choice of the crossover operator, where genes of parent individuals are combined to build the offspring. Optimal Recombination Problem consists in finding the best possible offspring as a result of a crossover operator, given two parent individuals. The following definition of the optimal recombination problem is motivated by the principles of (strictly) gene transmitting recombination formulated by Radcliffe [21].

Given: an instance I of combinatorial optimization problem with the set of feasible solutions Sol, objective function $f : \mathrm{Sol} \to \mathrm{R}$, and two parent solutions $\mathbf{x}^1 = (x_1^1, \ldots, x_n^1), \mathbf{x}^2 = (x_1^2, \ldots, x_n^2)$ from Sol.

Find: a feasible solution (offspring) $\mathbf{x}' = (x'_1, \ldots, x'_n)$ such that

(i) $x'_j = x_j^1$ or $x'_j = x_j^2$ for all $j = 1, \ldots, n$;
(ii) for each $\mathbf{x} \in \mathrm{Sol}$ such that $x_j = x_j^1$ or $x_j = x_j^2$, $j = 1, \ldots, n$, the inequality

$$f(\mathbf{x}') \le f(\mathbf{x}) \text{ holds in the case of minimization problem,}$$

or

$$f(\mathbf{x}') \ge f(\mathbf{x}) \text{ holds in the case of maximization problem.}$$

Note that in the case of permutation problems, the set of feasible solutions Sol consists of permutations, so the offspring is required to be a permutation too.

3 Makespan Minimization Problem on a Single Machine

Consider the Makespan Minimization Problem on a Single Machine $(1|s_{vu}|C_{\max})$, which is equivalent to the problem of finding the shortest Hamiltonian path in a digraph.

The input consists of a set of jobs $V = \{v_1, \ldots, v_k\}$ with positive processing times d_v, $v \in V$. All jobs are available for processing at time zero, and preemption is not allowed. A sequence dependent setup time is required to switch a machine from one job to another. Let s_{vu} be the a non-negative setup time from job v to job u for all $v, u \in V$, where $v \neq u$. The goal is to schedule the jobs on a single machine so as to minimize the maximum job completion time, the so-called *makespan* $C_{\max}$.

Problem $1|s_{vu}|C_{\max}$ is strongly NP-hard [15], and cannot be approximated with any constant or polynomial factor of the optimum in polynomial time, unless P=NP. Therefore metaheuristics, in particular, genetic algorithms, are appropriate for this problem.

The feasible solutions of $1|s_{vu}|C_{\max}$ can be represented in two natural ways in a GA: (I) genes encode jobs and (II) genes encode adjacencies. Let us consider these representations and the ORPs that correspond to them.

3.1 Optimal Recombination with Position-Based Representation

Let $\pi = (\pi_1, \ldots, \pi_k)$ denote a permutation of the jobs, i.e. π_i is the i-th job on the machine, $i = 1, \ldots, k$. Put $s(\pi) = \sum_{i=1}^{k-1} s_{\pi_i, \pi_{i+1}}$. Then the problem $1|s_{vu}|C_{\max}$ is equivalent to finding a permutation π^* that minimizes the total setup time $s(\pi^*)$.

The ORP for problem $1|s_{vu}|C_{\max}$ with position-based representation, given two parent solutions π^1 and π^2, asks for a permutation π' such that:

(i) $\pi'_i = \pi^1_i$ or $\pi'_i = \pi^2_i$ for all $i = 1, \ldots, k$;
(ii) π' has the minimum value of objective function $s(\pi')$ among all permutations that satisfy condition (i).

The following theorem is obtained in [11].

Theorem 1. *The ORP for problem* $1|s_{vu}|C_{\max}$ *with position-based representation is strongly NP-hard.*

We build an algorithm for solving the formulated ORP, using the approach of Serdyukov [23] which was developed for solving the travelling salesman problem with vertex requisitions.

Let us consider a bipartite graph $G = (V_n, V, U)$ where the two subsets of vertices of bipartition $V_n = \{1, \ldots, n\}$ and V have equal sizes and the set of edges is $U = \{\{i, v\} : i \in V_n,\ v = \pi^1_i \text{ or } v = \pi^2_i\}$. Now there is a one-to-one correspondence between the set of perfect matchings in graph G and the set of feasible solutions to an ORP instance with parents π^1, π^2: Given a perfect matching of the form $\{\{1, v^1\}, \{2, v^2\}, \ldots, \{n, v^n\}\}$, this mapping produces the permutation of jobs $(v^1, v^2, \ldots, v^n)$.

An edge $\{i, v\} \in U$ is called *special*, if $\{i, v\}$ belongs to all perfect matchings in graph G. Note that a maximal (by inclusion) connected subgraph of graph G with at least two edges is a *cycle*. Let $q(G)$ denote the number of cycles in graph G. The edges $\{i, v\} \in U$, such that $\pi^1_i = \pi^2_i$, are special and belong to none of the cycles, while the edges $\{i, v\} \in U$, such that $\pi^1_i \neq \pi^2_i$, belong to some cycles. Besides that, each cycle j, $j = 1, \ldots, q(G)$, of graph G contains exactly two maximal (edge disjoint) matchings, so it does not contain the special edges. Hence an edge $\{i, v\} \in U$ is special iff $\pi^1_i = \pi^2_i$, and every perfect matching in G is defined by a combination of maximal matchings chosen in each of the cycles and the set of all special edges.

The cycles of graph G may be computed in $O(k)$ time, e.g. by means of the "depth first" algorithm [5]. The special edges and maximal matchings in cycles may be found easily in $O(k)$ time.

Therefore, the ORP with position-based representation is solvable by the following algorithm: Build the bipartite graph G, identify the set of special edges and cycles and find all maximal matchings in cycles. Enumerate all perfect matchings of graph G by combining the maximal matchings of cycles and joining them with special edges. During enumeration, each of the perfect matchings is assigned the corresponding permutation π of jobs and $s(\pi)$ is computed. As a result, one can find the required permutation π'.

The total number of perfect matchings in graph G is equal to $2^{q(G)}$, so the time complexity of the above algorithm is $O(k2^{q(G)})$, where $q(G) \leq \lfloor \frac{k}{2} \rfloor$ and this bound is tight. Note that the proposed algorithm can be used for different objective functions defined on the set of permutations (see examples in [6,7,17,26]).

In [11], a modification of the described algorithm was proposed to speed up the evaluation of makespan function in the process of perfect matching enumeration. This modification performs a preprocessing stage, where the values of makespan function for *cycle contacts* are computed, and solves the ORP for $1|s_{vu}|C_{\max}$ in $O\left(q(G)2^{q(G)} + q(G)k\right)$ time.

Moreover in [11], it was shown that for almost all pairs of parent solutions $q(G) \leq \frac{\ln(k)}{\ln(2)}$, i.e. the cardinality of the set of feasible solutions is at most k. To describe this result precisely, let us give the following

Definition 1. [23] *A graph $G = (V_k, V, U)$ is called "good" if it satisfies the inequality $q(G) \leq \frac{\ln(k)}{\ln(2)}$.*

Let $\bar{\Re}_k$ denote the set of pairs of parent solutions with k jobs which correspond to "good" bipartite graphs G and let $\Re_k$ be the set of all pairs of parent solutions with k jobs. The results from [11] imply

Theorem 2. *$|\bar{\Re}_k|/|\Re_k| \longrightarrow 1$ as $k \to \infty$.*

According to the frequently used terminology (see e.g. [2]), this theorem means that *almost all* of the ORP instances have at most k feasible solutions and thus solvable in $O(k\ln(k))$ time.

In what follows, the crossover operator, solving the ORP by means of the algorithm described above, will be called Optimized Cycle Crossover. Such crossover may be considered as a derandomization of Uniform Cycle Crossover, which constructs an offspring so that maximal matching is chosen randomly in each of the cycles [7].

3.2 Optimal Recombination with Adjacency-Based Representation

Consider representation of solutions based on adjacencies. Here a solution is encoded as a vector $\mathbf{p} = (p_1, \ldots, p_k)$, where p_i is the job that immediately precedes job v_i, $i = 1, \ldots, k$. We assume that $p_i = v_0$ marks the first element of the sequence, where v_0 is an *artificial* job and $s_{v_0 v} = 0$ for all $v \in V$. Then $s(\mathbf{p}) = \sum_{i=1}^{k} s_{p_i, v_i}$. The problem $1|s_{vu}|C_{\max}$ is equivalent to finding a permutation $\mathbf{p}^*$ that minimizes the total setup time $s(\mathbf{p}^*)$.

The ORP for problem $1|s_{vu}|C_{\max}$ with adjacency-based representation, given two parent solutions $\mathbf{p}^1$ and $\mathbf{p}^2$, asks for a feasible solution $\mathbf{p}'$ such that:

(i) $p'_i = p^1_i$ or $p'_i = p^2_i$ for all $i = 1, \ldots, k$;
(ii) $\mathbf{p}'$ has the minimum value of objective function $s(\mathbf{p}')$ among all solutions that satisfy condition (i).

Theorem 3. *The ORP for problem* $1|s_{vu}|C_{\max}$ *with adjacency-based representation is strongly NP-hard.*

The proof is based on the known result from [11] about NP-hardness of the ORP for ATSP with adjacency-based representation (see Sect. 4). In the proof of NP-hardness of this ORP in Theorem 1.3 in [11], the vertex cover problem is reduced to it in such a way that there are arcs belonging to both parent tours (and thus should belong to the offspring tour). Let us take one of these arcs $(v_\ell, v_{\ell'})$ and delete it from both of the parent tours. The remaining two Hamilton paths will be used now to build a pair of parents for the $1|s_{vu}|C_{\max}$ ORP.

Suppose that an instance of ATSP (an n-vertex graph and arc weights c_{ij}, $i = 1, \ldots, n,\ \ j = 1, \ldots, n$) is given as defined in the proof of Theorem 1.3 from [11]. Let us construct an instance of $1|s_{vu}|C_{\max}$ problem with $k = n + 1$, where $s_{v_i,v_j} = c_{ij}$ for all $i = 1, \ldots, n,\ \ j = 1, \ldots, n$. The job v_{n+1} is introduced to ensure that the offspring solution will end with job v_ℓ. The setup times associated with v_{n+1} are set to zero, i.e. $s_{v_i,v_{n+1}} = s_{v_{n+1},v_i} = 0$. Suppose that two tours $v_{\ell'} = v_{i_1}, v_{i_2}, \ldots, v_{i_n} = v_\ell, v_{\ell'}$ and $v_{\ell'} = v_{j_1}, v_{j_2} \ldots, v_{j_n} = v_\ell, v_{\ell'}$ are the parent solutions of the ORP instance for ATSP constructed in the proof of Theorem 1.3 from [11]. Then the two parent solutions $\mathbf{p}^1$ and $\mathbf{p}^2$ for the $1|s_{vu}|C_{\max}$ ORP problem with adjacency-based representation are defined as follows: $p^1_{i_1} = v_0,\ p^1_{i_2} = v_{i_1}, \ldots, p^1_{i_n} = v_{i_{n-1}},\ p^1_{n+1} = v_{i_n}$ and $p^2_{j_1} = v_0,\ p^2_{j_2} = v_{j_1}, \ldots, p^2_{j_n} = v_{j_{n-1}},\ p^2_{n+1} = v_{j_n}$. The first and the last setups are the same in these schedules so an optimal ORP solution to $1|s_{vu}|C_{\max}$ will define an optimal ORP solution for ATSP, which is NP-hard. The described transformations are efficiently computable, so the ORP for $1|s_{vu}|C_{\max}$ problem is NP-hard as well. Q.E.D.

The optimized crossover operator, which solves the ORP for $1|s_{vu}|C_{\max}$ with adjacency-based representation, will be called Optimized Directed Edge Crossover and may be considered as a deterministic "direct descendant" of Directed Edge Crossover [25]. However, unlike the latter one, Optimized Directed Edge Crossover guarantees a gene transmitting recombination.

The following theorem gives an upper bound on time-complexity of ODEC.

Theorem 4. *The ORP for problem* $1|s_{vu}|C_{\max}$ *with adjacency-based representation is solvable in* $O(k^2 2^{\frac{k}{2}})$ *time.*

The proof is based on a Turing reduction of the ORP to $O(k)$ instances of Travelling Salesman Problem with forced edges on cubic graphs, i.e. the graphs with maximal degree three. Let us consider the ORP for $1|s_{vu}|C_{\max}$ with parent solutions $\mathbf{p}^1$ and $\mathbf{p}^2$ as a Shortest Hamilton Path problem on $(k+1)$-vertex digraph $G' = (V \cup \{v_0\}, A)$ where the arcs correspond to setups presented in $\mathbf{p}^1$ and $\mathbf{p}^2$ and the arc costs c_{ij} are equal to the setup times $s_{v_i v_j},\ i = 0, 1, \ldots, k, j = 1, \ldots, k$. Add a zero-cost arc (v, v_0) where $v \in V$ (enumerate all $O(k)$ options to choose $v \in V$). The resulting digraph is denoted by G'_v. This digraph may be transformed into a cubic graph $\overline{G}_v$ with forced edges the same way as in the

ORP for ATSP [11]. The graph $\overline{G}_v$ is constructed so that the setups presented in both parent solutions $\mathbf{p}^1$ and $\mathbf{p}^2$ correspond to forced edges.

All Hamiltonian cycles in $\overline{G}_v$ w.r.t. the set of forced edges are enumerated in time $O(2^{\frac{k}{2}})$ by the algorithm of Eppstein [10]. Then, for each Hamiltonian cycle C from $\overline{G}_v$ in each of the two directions we can check if it is possible to pass a circuit in G'_v through the arcs corresponding to edges of C, and if possible, compute the cost of the circuit. So, the ATSP problem on graph G'_v is solvable in $O(k2^{\frac{k}{2}})$ time and, therefore, solving the ATSP problems on graphs G'_v for all $v \in V$ requires $O(k^2 2^{\frac{k}{2}})$ time. Q.E.D.

4 Asymmetric Travelling Salesman Problem

In this section, we briefly consider the Travelling Salesman Problem (TSP). Suppose a complete digraph $\bar{G}$ is given. The set of vertices of $\bar{G}$ is $V = \{v_1, \ldots, v_n\}$ and a set of arcs is $A = \{(v_i, v_j) : v_i, v_j \in V,\ i \neq j\}$. A weight (length) $c_{ij} \geq 0$ of each arc $(v_i, v_j) \in A$ is given as well. It is required to find a Hamiltonian circuit of minimum length. If $c_{ij} \neq c_{ji}$ for at least one (v_i, v_j) then the TSP is called Asymmetric Travelling Salesman Problem (ATSP).

Feasible solution to the ATSP may be encoded as a sequence of the vertices in the TSP tour (without loss of generality we assume that the first position contains vertex v_1), or as a vector of adjacencies, where the immediate predecessor is indicated for each vertex.

Position-Based Representation. In the case of position-based encoding of solutions in ATSP, the ORP may be solved by the means of the algorithm described in Subsect. 3.1. A slight modification of the speed-up method from [11] is applicable here as well. Therefore, the ORP for ATSP with position-based representation is solvable in $O(n2^{\frac{n}{2}})$ time and almost all of its instances are solvable in $O(n\ln(n))$ time.

The following theorem is proved analogously to Theorem 2.2 form [11].

Theorem 5. *The ORP for ATSP with position-based representation is strongly NP-hard.*

Adjacency-Based Representation. The ORP for the ATSP with adjacency-based representation is shown to be strongly NP-hard but solvable in $O(n2^{\frac{n}{2}})$ time [11].

5 Computational Experiment on TSPLIB Instances

5.1 Genetic Algorithm

Yagiura and Ibaraki [26] applied a genetic algorithm with *elitist recombination* [16] to a number of combinatorial optimization problems on permutations. Let us consider the scheme of the GA with elitist recombination in a general form as it may be applied to a combinatorial minimization problem from Sect. 2.

Genetic Algorithm with Elitist Recombination

STEP 1. Construct the initial population.
STEP 2. Assign $t := 1$.
STEP 3. Repeat steps 3.1-3.4 until some stopping criterion is satisfied:
3.1. Choose randomly two parent individuals $\mathbf{x}^1, \mathbf{x}^2$ from the population.
3.2. Create an offspring $\mathbf{x}'$, applying a crossover to $\mathbf{x}^1$ and $\mathbf{x}^2$.
3.3. Replace one of the two parents by $\mathbf{x}'$.
3.4. $t := t + 1$.
STEP 4. The result is the best found individual w.r.t. objective function.

In our implementation of the GA with elitist recombination the *arbitrary insertion* method [26] is used for generating individuals of the initial population on Step 1. The population size N remains constant during the execution of the GA.

We apply an optimized crossover (ODEC or OCX) to generate a new individual on Step 3.2. One of the parents $\mathbf{x}^1$, $\mathbf{x}^2$ is replaced by the offspring as follows. We suppose without loss of generality that $f(\mathbf{x}^1) \leq f(\mathbf{x}^2)$. Replace $\mathbf{x}^2$ by $\mathbf{x}'$ with probability $P(\Delta_1/\Delta_2)$, otherwise replace $\mathbf{x}^1$ by $\mathbf{x}'$, where

$$\Delta_i = f(\mathbf{x}^i) - f(\mathbf{x}'), \ i = 1, 2, \tag{1}$$

$$P(\Delta_1/\Delta_2) = \min\left\{\frac{\Delta_1/\Delta_2}{a}, 1\right\}. \tag{2}$$

Note that $\Delta_2 \geq \Delta_1 \geq 0$ by the definition, and hence $\Delta_1/\Delta_2 \in [0, 1]$ (we consider $\Delta_1/\Delta_2 = 1$ if $\Delta_1 = \Delta_2 = 0$). The constant $a \geq 0$ is a tunable parameter. If $a = 0$, then p' always replaces p^2, and if $a = \infty$, then p' always replaces p^1.

The described GA was programmed in Java (NetBeans IDE 7.2.1) and tested on a computer with Intel Core 2 Duo CPU E7200 2,53 GHz processor, 2 Gb RAM.

5.2 Testing Problems and Experimental Outline

In the computational experiment, the described above GA was applied to ATSP and $1|s_{vu}|C_{\max}$ problems for evaluation of the effects of different optimized crossovers. Population size N was set to 50 and the tunable parameter a was set to 0.5.

In the experiments, we used the ATSP instances from TSPLIB [22] library. The ATSP collection includes instances from different applications [12–14]. The rbg instances come from a stacker crane application. The two ft instances arise in a problem of optimal sequencing tasks in the coloring plant of a resin production department. The ftv instances are from vehicle routing. Instances ry48p and kro124p are perturbed random Euclidean instances.

The names of the $1|s_{vu}|C_{\max}$ problems, their dimensions and optimal values $C^*_{\max}$ of makespan function are listed in Tables 1, 2 and 3. The optimal solutions to ATSP instances may be found in [22]. To find the optimal solutions to $1|s_{vu}|C_{\max}$ instances, we employed CPLEX MIP solver with addition of problem specific cuts which were constructed using the well-known approach [8].

Table 1. Instances of $1|s_{vu}|C_{\max}$ problems in series ftv

Instance	ftv33	ftv35	ftv38	ftv44	ftv47	ftv55	ftv64	ftv70	ftv90
k	34	36	39	45	48	56	65	71	91
$C^*_{\max}$	1159	1323	1399	1488	1634	1485	1656	1818	1482
Instance	ftv100	ftv110	ftv120	ftv130	ftv140	ftv150	ftv160	ftv170	
k	101	111	121	131	141	151	161	171	
$C^*_{\max}$	1691	1857	2023	2189	2320	2511	2561	2642	

Table 2. Instances of $1|s_{vu}|C_{\max}$ problems in series rbg

Instance	rbg323	rbg358	rbg403	rbg443
k	323	358	403	443
$C^*_{\max}$	1299	1130	2432	2687

The experiment consisted of two stages. On the first stage, the competing GAs were run for a given number of iterations in order to estimate the influence of different crossover operators upon the CPU cost of one GA iteration. Besides that, the shortest average execution time, denoted $t_{\min}$, was identified for each problem instance. On the second stage of experiments, a number of independent runs of competing GAs were made with each instance, given the time budget $t_{\min}$ seconds for each run. This stage was aimed at evaluation of frequency of finding optimal solutions.

5.3 Makespan Minimization Problem on a Single Machine

First we consider $1|s_{vu}|C_{\max}$ problem. In what follows, GA1 denotes the GA with position-based representation employing the Optimized Cycle Crossover. We use the notation GA2 for the GA with adjacency-based representation where the ORP is solved approximately with only one application of the algorithm of Eppstein. This corresponds to testing at most two options for vertex v among the vertices that correspond to the last jobs of parent schedules. Exact solving of this ORP requires enumeration of $O(k)$ options for vertex v and it was not used in the experiments due to high computational burden.

On the first stage of the experiment, the GA with elitist recombination was run 1000 times for each instance and each run continued for 4000 iterations for all

Table 3. Other $1|s_{vu}|C_{\max}$ instances

Instance	ry48p	ft53	ft70	kro124p
k	48	53	70	100
$C^*_{\max}$	13451	5846	36981	35227

Table 4. Average execution time for $1|s_{vu}|C_{\max}$ instances in series ftv

Instance	ftv33	ftv35	ftv38	ftv44	ftv47	ftv55	ftv64	ftv70	ftv90
$t^{\text{avr}}_{\text{GA1}}$	**0.26**	**0.25**	**0.27**	**0.29**	**0.3**	0.75	**0.72**	**0.95**	**1.27**
$t^{\text{avr}}_{\text{GA2}}$	0.35	0.38	0.44	0.51	0.58	**0.71**	0.87	1.15	1.49
Instance	ftv100	ftv110	ftv120	ftv130	ftv140	ftv150	ftv160	ftv170	
$t^{\text{avr}}_{\text{GA1}}$	**1.59**	**1.93**	2.93	4.34	5.07	**3.76**	**4.41**	**4.52**	
$t^{\text{avr}}_{\text{GA2}}$	1.8	2.5	**2.84**	**3.23**	**4.09**	4.88	5.82	8.51	

Table 5. Average execution time for other $1|s_{vu}|C_{\max}$ instances

Instance	ry48p	ft53	ft70	kro124p
$t^{\text{avr}}_{\text{GA1}}$	**0.43**	**0.32**	**0.35**	8.1
$t^{\text{avr}}_{\text{GA2}}$	0.66	0.67	1.03	**2.03**

problems except rbg series. In the case of rbg series, each run continued for 8000 iterations. These termination conditions were chosen on the basis of preliminary experiments which showed that such numbers of iterations were enough for GA1 to find the optimal solutions with a sufficiently high probability (more than 5 %).

Average execution times of GAs (in seconds) denoted by $t^{\text{avr}}_{\text{GA1}}$ and $t^{\text{avr}}_{\text{GA2}}$ are shown in Tables 4 and 5.

On majority of the problem instances (16 out of 21) presented in Tables 4 and 5, algorithm GA1 terminated faster compared to GA2. However the average execution time of GA2 is at most twice the average execution time of GA1 on all instances, except ft53 and ft70.

On the second stage of experiments, both GAs were run 1000 times for equal amount of time $t_{\min} = \min\{t^{\text{avr}}_{\text{GA1}}, t^{\text{avr}}_{\text{GA2}}\}$. The results of this stage are displayed in Tables 6 and 7. Here $F^{\text{opt}}_{\text{GA1}}$ and $F^{\text{opt}}_{\text{GA2}}$ are the frequencies of finding an optimum for GA1 and GA2 (respectively).

The statistical analysis of experimental data was carried out using the following approach. For each problem, the testing of an algorithm is considered as a sequence of ν Bernoulli trials, where "success" corresponds to finding an optimal solution. In our experiments, we performed $\nu = 1000$ trials with GA1 and GA2 algorithms. The confidence intervals for the success probability p^* are built using the standard method [19] applied to the Bernoulli distribution and presented in Tables 6 and 7 (the confidence level is set to 5 %). These tables show that on most of the instances the considered GAs have similar performance.

On the first stage of experiments with rbg series the execution time of GA2 was much longer than that of GA1. Therefore both algorithms were given $t^{\text{avr}}_{\text{GA1}}$ seconds on the second stage. The execution times $t^{\text{avr}}_{\text{GA1}}$ and the frequencies of finding an optimum in 1000 runs are presented in Table 8. GA1 has a significant advantage on this series, which is due to large computational cost of crossover in GA2.

Table 6. Frequencies of finding the optimum and confidence intervals for $1|s_{vu}|C_{\max}$ instances in series ftv

Instance	ftv33	ftv35	ftv38	ftv44	ftv47	ftv55
$F^{\text{opt}}_{\text{GA1}}$	**0.69**	**0.6**	**0.6**	**0.6**	**0.5**	0.49
$I^{\text{conf}}_{\text{GA1}}$	(0.66;0.72)	(0.57;0.63)	(0.57;0.63)	(0.57;0.63)	(0.47;0.53)	(0.46;0.52)
$F^{\text{opt}}_{\text{GA2}}$	0.65	0.59	0.59	0.44	0.4	**0.5**
$I^{\text{conf}}_{\text{GA2}}$	(0.62;0.68)	(0.56;0.62)	(0.56;0.62)	(0.41;0.47)	(0.37;0.43)	(0.47;0.53)
Instance	ftv64	ftv70	ftv90	ftv100	ftv110	ftv120
$F^{\text{opt}}_{\text{GA1}}$	**0.49**	0.51	**0.4**	**0.37**	0.31	0.24
$I^{\text{conf}}_{\text{GA1}}$	(0.46;0.52)	(0.48;0.54)	(0.37;0.43)	(0.34;0.4)	(0.28;0.34)	(0.21;0.27)
$F^{\text{opt}}_{\text{GA2}}$	**0.49**	**0.53**	0.39	0.33	**0.35**	**0.27**
$I^{\text{conf}}_{\text{GA2}}$	(0.46;0.52)	(0.5;0.56)	(0.36;0.42)	(0.3;0.36)	(0.32;0.38)	(0.24;0.3)
Instance	ftv130	ftv140	ftv150	ftv160	ftv170	
$F^{\text{opt}}_{\text{GA1}}$	0.27	0.31	0.29	**0.4**	**0.36**	
$I^{\text{conf}}_{\text{GA1}}$	(0.24;0.3)	(0.28;0.34)	(0.26;0.32)	(0.37;0.43)	(0.33;0.39)	
$F^{\text{opt}}_{\text{GA2}}$	**0.31**	**0.41**	**0.31**	0.39	0.3	
$I^{\text{conf}}_{\text{GA2}}$	(0.28;0.34)	(0.38;0.44)	(0.28; 0.34)	(0.36;0.42)	(0.27;0.33)	

Table 7. Frequencies of finding the optimum and confidence intervals for other $1|s_{vu}|C_{\max}$ instances

Instance	ry48p	ft53	ft70	kro124p
$F^{\text{opt}}_{\text{GA1}}$	**0.4**	**0.55**	**0.43**	0.22
$I^{\text{conf}}_{\text{GA1}}$	(0.37;0.43)	(0,52;0,58)	(0.4;0.46)	(0.19;0.25)
$F^{\text{opt}}_{\text{GA2}}$	**0.4**	0.35	0.32	**0.53**
$I^{\text{conf}}_{\text{GA2}}$	(0.37;0.43)	(0.32;0.38)	(0.29;0.35)	(0.5;0.56)

Table 8. Average time and frequency of finding the optimum for $1|s_{vu}|C_{\max}$ instances in series rbg

Instance	rbg323	rbg358	rbg403	rbg443
$t^{\text{avr}}_{\text{GA1}}$	7.02	7.47	7.54	7.53
$F^{\text{opt}}_{\text{GA1}}$	**0.195**	**0.111**	**0.107**	**0.091**
$F^{\text{opt}}_{\text{GA2}}$	0	0	0	0

We carried out additional experiment in order to find out whether this drastic difference in computational cost of the two crossovers is due to specific problems structure in rbg series or it is due to high dimension of these problems. To this end, we generated testing instances with the same numbers of jobs as in rbg and chose the setup times uniformly from $[s_{\max}/2, s_{\max}]$, where $s_{\max}$ is the largest

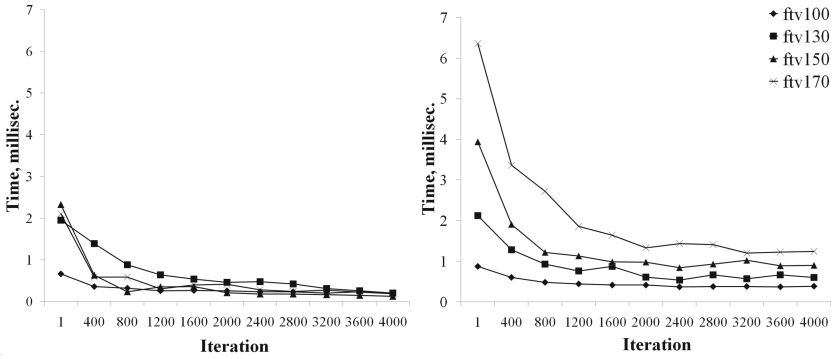


Fig. 1. Average CPU time of crossover on $1|s_{vu}|C_{\max}$ instances in series ftv. The left plot corresponds to GA1, the right plot corresponds to GA2.

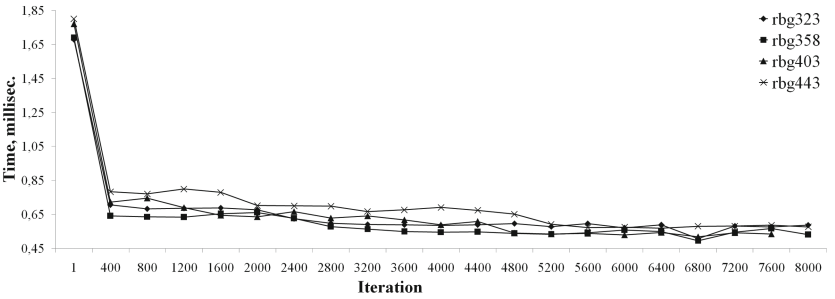


Fig. 2. Average CPU time of crossover in GA1 on series rbg.

setup time on the corresponding instance from series rbg. It turned out that there was no drastic difference between the CPU times of GA1 and GA2 in this additional experiment, so we conclude that the cause of poor performance of GA1 on series rbg might be in the specific structure of these instances.

As seen from the tables, GA1 in general demonstrates more stable results than GA2.

The dynamics of CPU time required for solving ORPs in GA1 and GA2 on instances ftv100, ftv130, ftv150, ftv170 is displayed in Fig. 1. The plots for the remaining problems of ftv series were analogous and they are skipped here. The CPU time required for solving ORPs in GA1 on series rbg is shown in Fig. 2. It can be seen that the time-complexity of crossover operators decreases with iterations count, which is due to decreasing population diversity. The CPU cost of optimized crossover in the case of the position-based representation is somewhat smaller compared to the adjacency-based representation. The ORPs for instance ftv170 are especially hard for GA2. This observation agrees with the greater execution time of GA2 and its lower frequency of obtaining the optimum on ftv170.

It was mentioned in Subsect. 3.1 that with probability approaching to 1, as $k \to \infty$, randomly chosen parent solutions define an ORP instance with "good" graph G and the Optimized Cycle Crossover requires $O(k\ln(k))$ time. The high frequency of such ORP instances was observed in the experiments, e.g. on ftv series the "good" graphs G were observed in more than 60 % of crossover calls, and in more than 80 % of the calls on rbg series. In process of GA execution this frequency increased.

5.4 Asymmetric Travelling Salesman Problem

Experiments with ATSP were carried out following the same outline as with $1|s_{vu}|C_{\max}$ on instances of TSPLIB. In what follows, GA1' denotes the GA for ATSP based on Optimized Cycle Crossover and GA2' denotes the GA for ATSP based on Optimized Directed Edge Crossover, where the ORP with adjacency-based representation is solved exactly (see Sect. 4).

Average execution times of GA1' and GA2' ($t^{\text{avr}}_{\text{GA1'}}$ and $t^{\text{avr}}_{\text{GA2'}}$) are close to those of GA1 and GA2 respectively. A rough comparison on the basis of computers performance table [9] suggests that the CPU resource given to GAs in our experiments is approximately 3 times the resource used by SAX/RAI memetic algorithm in [1] on all instances, except for series rbg. The latter series is excluded in this comparison because in [1], a problem-specific heuristic of Zhang [27] was used in construction of initial populations. This heuristic of Zhang is very efficient on series rbg and most likely the optimal solutions to all rbg instances were found in SAX/RAI memetic algorithm at the initialization stage.

Tables 9, 10 and 11 present the frequencies of finding an optimum in 1000 runs, given $t_{\min} = \min\{t^{\text{avr}}_{\text{GA1'}}, t^{\text{avr}}_{\text{GA2'}}\}$ CPU seconds for each run, and the confidence intervals for the probability of obtaining an optimum (the confidence level is 5 %).

On majority of the problems (19 out of 25) GA2' finds an optimum more frequently than GA1' (in 14 cases among these the confidence intervals for p^* do not intersect), although GA1' is still more successful on rbg series. Better results of GA2' with adjacency-based representation in the case of ATSP, compared to the results of GA2 on $1|s_{vu}|C_{\max}$ problem, are presumably due to exact solving of the ORP in Optimized Directed Edge Crossover.

Comparing GA2' and SAX/RAI memetic algorithm from [1] in terms of frequencies of finding optimal solutions, we estimate the frequency of GA2' as approximately 70 % of the frequency reported for SAX/RAI memetic algorithm on all instances, except for series rbg. This outcome seems to be promising since the general GA outline and tunable parameters were chosen quite straightforwardly in this paper.

Summing up the experimental results for $1|s_{vu}|C_{\max}$ problem and ATSP in terms of frequency of finding optimal solutions we can conclude that the two compared approaches are competitive with each other. In the case of $1|s_{vu}|C_{\max}$ problem, GA1 tends to outperform GA2 on larger instances such as ftv170, rbg323, rbg358, rbg403 and rbg443. In the case of ATSP, GA2' dominates GA1', except for series rbg where instances have special structure.

Table 9. Frequencies of finding the optimum and confidence intervals for ATSP series ftv

Instance	ftv33	ftv35	ftv38	ftv44	ftv47	ftv55
$F^{\text{opt}}_{\text{GA1}}$,	0.51	0.53	0.52	0.51	0.47	0.4
$I^{\text{conf}}_{\text{GA1}}$,	(0.48;0.54)	(0.5;0.56)	(0.49;0.55)	(0.48;0.54)	(0.44;0.5)	(0.37;0.43)
$F^{\text{opt}}_{\text{GA2}}$,	**0.93**	**0.76**	**0.75**	**0.7**	**0.86**	**0.67**
$I^{\text{conf}}_{\text{GA2}}$,	(0.91;0.95)	(0.73;0.79)	(0.72;0.78)	(0.67;0.73)	(0.84;0.88)	(0.64;0.7)
Instance	ftv64	ftv70	ftv90	ftv100	ftv110	ftv120
$F^{\text{opt}}_{\text{GA1}}$,	0.4	0.39	0.35	0.33	0.29	0.22
$I^{\text{conf}}_{\text{GA1}}$,	(0.37;0.43)	(0.36;0.42)	(0.32;0.38)	(0.3;0.36)	(0.26;0.32)	(0.19;0.25)
$F^{\text{opt}}_{\text{GA2}}$,	**0.79**	**0.65**	**0.38**	**0.47**	**0.32**	**0.26**
$I^{\text{conf}}_{\text{GA2}}$,	(0.76;0.82)	(0.62;0.68)	(0.35;0.41)	(0.44;0.5)	(0.29;0.35)	(0.23;0.29)
Instance	ftv130	ftv140	ftv150	ftv160	ftv170	
$F^{\text{opt}}_{\text{GA1}}$,	0.29	0.2	0.22	0.38	**0.31**	
$I^{\text{conf}}_{\text{GA1}}$,	(0.26;0.32)	(0.17;0.23)	(0.19;0.25)	(0.35;0.41)	(0.28;0.34)	
$F^{\text{opt}}_{\text{GA2}}$,	**0.41**	**0.42**	**0.43**	**0.41**	0.3	
$I^{\text{conf}}_{\text{GA2}}$,	(0.38;0.44)	(0.39;0.45)	(0.4;0.46)	(0.38;0.44)	(0.27;0.33)	

Table 10. Frequencies of finding the optimum and confidence intervals for other ATSP instances

Instance	ry48p	ft53	ft70	kro124p
$F^{\text{opt}}_{\text{GA1}}$,	0.37	0.53	**0.42**	0.1
$I^{\text{conf}}_{\text{GA1}}$,	(0.34;0.4)	(0.5;0.56)	(0.39;0.45)	(0.08;0.12)
$F^{\text{opt}}_{\text{GA2}}$,	**0.42**	**0.64**	**0.42**	**0.47**
$I^{\text{conf}}_{\text{GA2}}$,	(0.39;0.45)	(0.61;0.67)	(0.39;0.45)	(0.44;0.5)

Table 11. Frequency of finding the optimum for ATSP series rbg

Instance	rbg323	rbg358	rbg403	rbg443
$F^{\text{opt}}_{\text{GA1}}$,	**0.145**	**0.105**	**0.086**	**0.079**
$F^{\text{opt}}_{\text{GA2}}$,	0	0.001	0	0

We carried out an additional experiment in order to compare the optimized crossovers ODEC and OCX to their randomized prototypes DEC and RCX. It clearly showed an advantage of ODEC and OCX over DEC and RCX. For the large-scale problems such as ftv110, ftv120, ftv150, kro124p, rbg323, and rbg358 the GA with operators DEC and RCX found optimal solution within the same CPU time limit $t_{\min}$ no more than once out of 1000 runs. A similar situation was observed in the case of $1|s_{vu}|C_{\max}$ problem.

6 Conclusions

Optimal recombination problems for Makespan Minimization Problem on a Single Machine and for Asymmetric Travelling Salesman Problem are shown to be NP-hard under two "natural" solutions encodings (position-based representation and adjacency-based representation). In the case of position-based representation, almost all instances of the optimal recombination problem are polynomially solvable both for $1|s_{vu}|C_{\max}$ and ATSP. The worst case time-complexity of optimized crossover operators is $O(k^2 2^{\frac{k}{2}})$ in the case of adjacency-based representation for $1|s_{vu}|C_{\max}$ and it is $O(k2^{\frac{k}{2}})$ (or $O(n2^{\frac{n}{2}})$) in the other cases considered in this paper. The computational experiment indicates that the two approaches to optimal recombination yield competitive results. However GA with position-based representation dominates GA with adjacency-based representation on problems with special structure.

Further research might extend the analysis to other problems on permutations and reduce some of the known upper bounds on the time complexity of optimal recombination. In particular, we hypothesize that the time complexity of the optimized crossover for $1|s_{vu}|C_{\max}$ with adjacency-based representation may be reduced to $O(k2^{\frac{k}{2}})$. We expect that the GA behavior observed in this paper might be helpful for improvement of state-of-the-art metaheuristics for problems on permutations.

Acknowledgements. This research is supported by the Russian Science Foundation grant 15-11-10009, except for Subsect. 3.2 which is supported by RFBI grant 15-01-00785.

References

1. Buriol, L.S., Franca, P.M., Moscato, P.: A new memetic algorithm for the asymmetric traveling salesman problem. J. Heuristics **10**, 483–506 (2004)
2. Chvatal, V.: Probabilistic methods in graph theory. Ann. Oper. Res. **1**, 171–182 (1984)
3. Cirasella, J., Johnson, D.S., McGeoch, L.A., Zhang, W.: The asymmetric traveling salesman problem: algorithms, instance generators, and tests. In: Buchsbaum, A.L., Snoeyink, J. (eds.) ALENEX 2001. LNCS, vol. 2153, pp. 32–59. Springer, Heidelberg (2001)
4. Cook, W., Seymour, P.: Tour merging via branch-decomposition. INFORMS J. Comput. **15**(2), 233–248 (2003)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
6. Cotta, C., Alba, E., Troya, J.M.: Utilizing dynastically optimal forma recombination in hybrid genetic algorithms. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 305–314. Springer, Heidelberg (1998)
7. Cotta, C., Troya, J.M.: Genetic forma recombination in permutation flowshop problems. Evol. Comput. **6**(1), 25–44 (1998)

8. Dantzig, G., Fulkerson, R., Johnson, S.: Solution of a large-scale traveling salesman problem. Oper. Res. **2**, 393–410 (1954)
9. Dongarra, J.J.: Performance of various computers using standard linear equations software. Technical Report No. CS-89-85, University of Manchester, 110 p. (2014)
10. Eppstein, D.: The traveling salesman problem for cubic graphs. J. Graph Algorithms Appl. **11**(1) (2007)
11. Eremeev, A.V., Kovalenko, J.V.: Optimal recombination in genetic algorithms for combinatorial optimization problems: Part II. Yugoslav J. Oper. Res. **24**(2), 165–186 (2014)
12. Fischetti, M., Toth, P.: An additive bounding procedure for the asymmetric travelling salesman problem. Math. Program. A **53**, 173–197 (1992)
13. Fischetti, M., Toth, P.: A polyhedral approach to the asymmetric travelling salesman problem. Manage. Sci. **43**, 1520–1536 (1997)
14. Fischetti, M., Toth, P., Vigo, D.: A branch and bound algorithm for the capacitated vehicle routing problem on directed graphs. Oper. Res. **42**, 846–859 (1994)
15. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-completeness. W.H. Freeman and Company, San Francisco (1979)
16. Goldberg, D., Thierens, D.: Elitist recombination: an integrated selection recombination GA. In: Proceedings of the First IEEE World Congress on Computational Intelligence. vol. 1, pp. 508–512. IEEE Service Center, Piscataway, New Jersey (1994)
17. Hazir, O., Günalay, Y., Erel, E.: Customer order scheduling problem: a comparative metaheuristics study. Int. Journ. Adv. Manuf. Technol. **37**, 589–598 (2008)
18. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
19. Mood, A.M., Graybill, F.A., Boes, D.C.: Introduction to the Theory of Statistics, 3rd edn. McGraw-Hill, New York (1973)
20. Nagata, Y., Soler, D.: A new genetic algorithm for the asymmetric travelling salesman problem. Expert Syst. Appl. **39**(10), 8947–8953 (2012)
21. Radcliffe, N.J.: The algebra of genetic algorithms. Ann. Math. Artif. Intell. **10**(4), 339–384 (1994)
22. Reinelt, G.: TSPLIB - a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
23. Serdyukov, A.I.: On travelling salesman problem with prohibitions. Upravlaemye systemi **1**, 80–86 (1978). (in Russian)
24. Tinós, R., Whitley, D., Ochoa, G.: Generalized asymmetric partition crossover (GAPX) for the asymmetric TSP. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 501–508. ACM, New York (2014)
25. Whitley, D., Starkweather, T., Shaner, D.: The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination. In: Handbook of Genetic Algorithms, pp. 350–372. Van Nostrand Reinhold, New York (1991)
26. Yagiura, M., Ibaraki, T.: The use of dynamic programming in genetic algorithms for permutation problems. Eur. Jour. Oper. Res. **92**, 387–401 (1996)
27. Zhang, W.: Depth-first branch-and-bound versus local search: a case study. In: Proceedings of 17th National Conference on Artificial Intelligence, pp. 930–935. Austin, TX (2000)

Hyperplane Elimination for Quickly Enumerating Local Optima

Brian W. Goldman(✉) and William F. Punch

BEACON Center for the Study of Evolution in Action, Michigan State University, East Lansing, USA
brianwgoldman@acm.org, punch@msu.edu

Abstract. Examining the properties of local optima is a common method for understanding combinatorial-problem landscapes. Unfortunately, exhaustive algorithms for finding local optima are limited to very small problem sizes. We propose a method for exploiting problem structure to skip hyperplanes that cannot contain local optima, allowing runtime to scale with the number of local optima instead of with the landscape size. We prove optimality for linear functions and Concatenated Traps, and we provide empirical evidence of optimality on NKq Landscapes and Ising Spin Glasses. We further refine this method to find solutions that cannot be improved by flipping r or fewer bits, which counterintuitively can reduce total runtime. While previous methods were limited to landscapes with at most 2^{34} binary strings, hyperplane elimination can enumerate the same problems with 2^{77} binary strings, and find all 4-bit local optima of problems with 2^{200} binary strings.

Keywords: Landscape understanding · Gray-Box · Mk Landscapes

1 Introduction

The ruggedness and high dimensionality of most problem landscapes makes them challenging to analyze and understand. However, doing so can be helpful in quantifying the difficulty of a problem. Furthermore, this understanding can be used to design search algorithms that specifically deal with those difficulties. Similarly, knowing problem characteristics that favor a particular algorithm can help researchers choose the algorithm most likely to perform well on their problem.

A common way of analyzing landscapes is to examine both the frequency and distribution of local optima [1], as well as how different search operators transition between these optima [9,13,14]. However, finding all of the local optima of a complex problem is prohibitively time-consuming even for small problems. Many studies have been limited to 18-bit problems [13,14] due to this time constraint. By leveraging recent advancements in Gray-Box optimization that allow for constant time local search [2], this limit was raised to 30-bit problems [9]. While sampling methods can approximate the number and distribution of local optima for much larger landscapes [7], they do not provide enough detail necessary for some metrics [9].

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 154–169, 2016.
DOI: 10.1007/978-3-319-30698-8_11

Here we introduce a method for finding all local optima for the same problems using up to 77 bits. The cornerstone of this method is the identification of hyperplanes that cannot contain any local optima, allowing large portions of the search space to be skipped during enumeration. The techniques developed here can be applied to the generalized problem class of Mk Landscapes, which contains many real world and benchmark combinatorial problems.

2 Mk Landscapes as a Tool for Problem Generalization

An Mk Landscape [15] is any function whose value is equal to the sum of a set of subfunctions. Each subfunction uses a small subset of variables from the original function's input. Combined with limits on the number of subfunctions and the size of each subfunction's subset, Mk Landscapes can be efficiently searched for both local [2,16] and global [4,12] optima.

Formally, an Mk Landscape is any function $f : \mathbb{B}^N \rightarrow \mathbb{R}$ that can be expressed in the following form:

$$f(x) = \sum_{i=1}^{M} f_i(mask(x, s_i)) \tag{1}$$

In this equation f_i is a subfunction such that $f_i : \mathbb{B}^{|s_i|} \rightarrow \mathbb{R}$. Each s_i is a set of variables in x, such that $|s_i| \leq k$. The $mask$ function returns the values in x associated with each variable in s_i. The total number of subfunctions M is constrained to grow at $\mathcal{O}(N)$, and k is constant with respect to N.

This formulation can represent many problems of real-world interest, as well as the most commonly used combinatorial benchmark problems. In this work we examine 5 problems in particular: Concatenated Traps, Adjacent and Random NKq Landscapes, Ising Spin Glasses, and MAX-kSAT.

The Concatenated Trap problem [3] is a composition of k-order deceptive, separable subfunctions. In Mk Landscape terms, $M = N/k$ such that $\forall_i |s_i| = k$ and $\forall_{i \neq j} s_i \cap s_j = \emptyset$. Each f_i applies an identical subfunction based on the number of variables set to 1:

$$trap(t) = \begin{cases} k - 1 - t, & t < k \\ k, & t = k \end{cases} \tag{2}$$

While this problem is still in common use [6,8], most advanced methods can solve it trivially [5] and when expressed as an Mk Landscape it can be solved exactly in $\mathcal{O}(N)$ time [15]. We therefore include it only because its structure allows for straight forward algorithm analysis. Here we set $k = 5$ to ensure sufficient deceptiveness.

NKq Landscapes specify a class of randomly generated problem instances using 3 paramters: (1) the number of problem variables N, (2) the amount of variable epistasis K where $k = K + 1$, and (3) the number of unique subfunction values q. From these parameters a landscape is generated by creating $M = N$ subfunctions f_i, where f_i uses variable x_i and K others to look up a fitness value

in the range $[0..q-1]$ from a randomly generated table. As NKq Landscapes are structured as a sum of bounded subfunctions, they are a natural fit for Mk Landscapes and are the most studied problem class for Gray-Box optimization [2, 4,9,12,15,16]. We set $k = 3$ and $q = 2^{K+1} = 2^k = 8$ in line with previous work.

We use two common variants of NKq in our experiments that specify how the K additional variables in each subfunction are chosen: Adjacent and Random. In Adjacent NKq, f_i depends on variable indices $[i..(i+k) \bmod N]$. In Random NKq, f_i depends on x_i and a random set of K unique variables that does not include x_i. While Random NKq is NP-Hard, the structure of Adjacent NKq allows for a polynomial time solution [17].

Ising Spin Glasses are a type of MAX-CUT problem derived from statistical physics. Each atom in the glass (vertex) can be assigned a spin, with the goal being to find the set of assignments that minimize the energy between nearby atoms (edges). Similar to Adjacent NKq, the $2D \pm J$ subset of Ising Spin Glasses can be polynomially solved [10]. In this subset, the graph is defined as a square two-dimensional grid with periodic boundaries such that each edge weight is chosen from $\{-1, 1\}$. Each vertex is assigned a spin from $\{-1, 1\}$ with the energy in the glass equal to

$$\sum_{e_{ij} \in E} x_i e_{ij} x_j \tag{3}$$

where e_{ij} is the weight of the edge connecting vertex i to vertex j. In Mk Landscape terms this type of spin glass has $M = 2N$ and $k = 2$.

Our final problem is randomly generated maximum satisfiability or MAX-kSAT. This version of the canonical NP-Complete boolean satisfiability problem is formulated as the maximization of $M = 4.27N$ clauses, each containing exactly $k = 3$ unique literals [11]. A clause is satisfied if any of its literals match how a solution's variables are set.

3 Gray-Box Enumeration of Mk Landscapes

When considered as a black box, the process of finding all local optima in a landscape requires $\Omega(N2^N)$ time. This is because each of the 2^N solutions must be compared with each of its N neighbors. Extending this method to look for solutions that cannot be improved by flipping r or fewer bits requires $\Omega(N^r 2^N)$ time. However, by exploiting Gray-Box optimization methods, previous work [9] was able to find all r-bit local optima of Mk Landscapes in $\mathcal{O}(2^N)$ for small r.

The first major result in Gray-Box optimization was the proof that the list of fitness-improving moves from a solution can be updated after a bit flip in $\mathcal{O}(1)$ time [16]. The fitness effect of flipping x_j only changes after flipping x_i if there is a non-linear relationship between x_i and x_j. In an Mk Landscape, variables can only have a non-linear relationship if they appear together in at least one s_i. By definition the total number of non-linear relationships in an Mk Landscape is linear with N, meaning on average the amortized number of non-linear relationships per variable is $\mathcal{O}(1)$.

This result has been extended, with no increase in asymptotic costs, to include search for solutions that cannot be improved by flipping r or fewer bits [2]. Consider two variables x_i and x_j that do not appear in the same s_i. By definition the fitness effect of flipping both x_i and x_j is equal to the sum of flipping each independently. As a result, if neither individual flip is fitness-improving, flipping both together cannot be fitness-improving. By examining the graph of non-linear interactions between variables, this principle can be extended to any collection of r variables, with the number of useful collections growing at $\mathcal{O}(N)$ when r is a small constant. As a result, the time to update the list of improving moves after up to r bit-flips is still $\mathcal{O}(1)$.

These advances can be applied directly to the task of finding all local optima in a landscape [9]. Instead of requiring $\mathcal{O}(N)$ time to evaluate each solution and to check if it is a local optima, only $\mathcal{O}(1)$ time is needed. Consider an enumeration of the landscape that uses gray-codes, meaning that each transition between solutions requires exactly 1 bit flip. In Gray-Box optimization updating fitness and the list of improving moves after a single bit flip takes $\mathcal{O}(1)$ time. That property holds even when looking for r-bit fitness-improving moves. Therefore Gray-Box enumeration is able to find all local optima in $\mathcal{O}(2^N)$ time.

4 Hyperplane Elimination

Due to the limited non-linearity of the Gray-Box domain, it is possible to exclude large parts of the search space without missing any local optima. Consider the representation presented in the top of Fig. 1. In a Black-Box domain, enumeration would progress as a binary counter, treating *index* 0 (symbol A in the solution) as the least significant bit. This ordering ensures that before changing *index* i, all possible settings of *index* 0 through $i-1$ have been tested. This corresponds to examining the hyperplane where the lowest i positions can vary and all other positions remain fixed. The Gray-Box domain makes it possible to skip hyperplanes that cannot contain local optima. In Fig. 1, move m_i, which flips variable F, is a fitness improvement when enumeration starts (all variables set to 0). Due to the known relationships between variables, we know that the quality of m_i only depends on variables C, E, F, and H. Therefore, until one of those four variables is modified, the solution cannot be a local optimum. As a

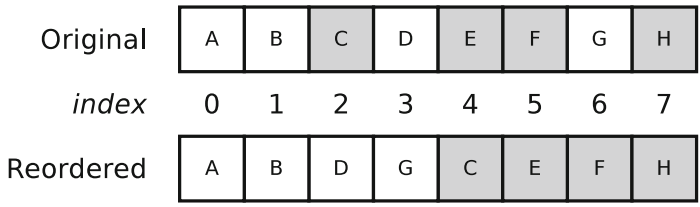


Fig. 1. Example reordering. The gray variables are all dependencies for move m_i which flips variable F. Reordering improves m_i's lowest *index* dependency from 2 to 4.

result, the hyperplane **0*00*0 cannot contain a local optima and can therefore be eliminated from consideration during enumeration. More generally, if at any point during enumeration there exists a fitness-improving move, no local optima can exist until at least one dependency of that move is modified. Any hyperplane that has all of a fitness-improving move's dependencies fixed cannot contain any local optima.

Algorithm 1. Find all local optima using Hyperplane Elimination.

Input: $solution \leftarrow \{0\}^N$
Input: *move_bin* (Count of improving moves at each minimum dependency)
Input: FlipBit (Function that flips *index* in *solution* and updates *move_bin*)
Output: *found* (list of all local optima)

```
 1: found ← [ ]
 2: index ← N − 1
 3: while index < N do
 4:     while index ≥ 0 and move_bin[index] = 0 do
 5:         index ← index − 1
 6:     if index = −1 then
 7:         found ← found + [solution]
 8:         index ← 0
 9:     while index < N and solution[index] = 1 do
10:         FlipBit(index)
11:         index ← index + 1
12:     if index < N then
13:         FlipBit(index)
```

This knowledge can be exploited to skip parts of the enumeration, as shown in Algorithm 1. Initially all variables are set to 0 and each fitness-improving move is put into a table *move_bin* based on that move's lowest *index* dependency. This is the first *index* that can be modified by enumeration that can potentially change the fitness effect of making that move. Algorithm 1 works by finding the highest *index* in *move_bin* that is not empty (Line 4) and then adding a 1 to that *index* in *solution*. Initially all bins could contain a fitness-improving move, so *index* starts at $N - 1$. If at any point all bins are empty, then the solution is added to the list of local optima *found*. Algorithm 1 then adds a 1 to *index*, using the loop on Line 9 to perform carry operations and Line 13 to create the new 1 value. Iteration stops when the carry exceeds the solution length.

When performing subsequent iterations, not all bins need to be checked. Instead, the highest *index* bin that must be tested is the highest *index* flipped by the previous iteration. This simplification is possible because the previous iterations have verified that all moves in higher *index* bins are not fitness-improving, and no action performed during that iteration can make them fitness-improving. Furthermore, *index* is always the location of the lowest index 1 bit in *solution*, meaning iteration can continue immediately from the found *index*. This is true by construction. Initially *solution* contains all 0s. Each time a 1 is inserted, its

position is equal to *index* and *index* is only increased by carry operations which reset a position to 0 before increasing *index*.

Extending Algorithm 1 to search for r-bit optima only requires adding the necessary r-bit moves to *move_bin*. As discussed in Sect. 3, we know the number of these moves is $\mathcal{O}(N)$ and can be kept updated in $\mathcal{O}(1)$ time per flip. Therefore, for small r, the cost of finding r-bit local optima is no more than a constant slower than finding 1-bit local optima. In fact, we will provide evidence in Sect. 7.2 that increasing r can actually decrease runtime.

5 Reordering Variables

Changing the order in which variables are indexed can provide further efficiency gains. When a move is fitness-improving, the amount of search space that is skipped depends on how high its lowest *index* dependency is. Therefore, by rearranging the order to make its lowest *index* dependency higher, more search space can be skipped. Figure 1 shows how changing the *index* order of variables improves m_i's lowest *index* dependency from two to four. Consider that before reordering m_i allows the hyperplane **000000 to be skipped by Algorithm 1, while after reordering the hyperplane that can be skipped is ****0000. Now, whenever m_i is a fitness improvement, four times as many solutions are skipped.

We perform this reordering in a greedy fashion, such that the move with the least-unmapped dependencies has all of its remaining dependencies mapped to the highest remaining indices. Before iteration, each move is binned based on its number of dependencies, and dictionaries are created to convert a move to its bin location and a variable to the moves that depend on that variable. Each of these requires $\mathcal{O}(N)$ time as for each move ($\mathcal{O}(N)$) you must consider all variables ($\mathcal{O}(N)$) in all subfunctions ($\mathcal{O}(1)$) it overlaps. The algorithm then iteratively queries the move bins in ascending order (requiring at most $kM/N = \mathcal{O}(1)$ amortized steps) until the first non-empty bin is found. All dependencies of that move are assigned positions, and all moves that depend on that variable are updated in the move bin (requiring at most $\mathcal{O}(1)$ amortized updates). Each iteration finishes a move, meaning after $\mathcal{O}(N)$ iterations the process is complete.

The quality of an ordering is determined by how many solutions Algorithm 1 can skip when using that ordering. Each time the loop on Line 4 stops with $index > 0$, 2^{index} solutions have been skipped. For practical purposes we will assume that all moves are equally likely to be fitness-improving.

Lemma 1. *All optimal orderings of variables are orderings of moves, such that all remaining dependencies of each move are sequentially assigned the highest remaining position in the enumeration ordering.*

Proof. Any ordering of variables that cannot be described by an ordering of moves must contain positions i and j with the following properties: (1) $i < j$ (2) i is the minimum dependency of a move m (3) no move with minimum dependency of i or higher depends on x_j. In this situation, swapping the order of x_i and x_j will raise m's minimum dependency by at least one, but it will

not lower the minimum dependency of any move. Therefore, any ordering which contains this property cannot be optimal. □

Moves can have their dependencies assigned in some permutation P of all possible moves, such that P_0 is the first move to have its dependencies assigned. $D_P(i, m)$ is a function that returns how many unassigned dependencies move m has after all moves in P before i have been assigned. A greedy solution to this problem is one such that P_i is set to be the move that minimizes $D_P(i, m)$.

Theorem 1. *All optimal orderings of variables are greedy orderings of moves.*

Proof. All non-greedy solutions must have some i such that $D_P(i, P_i) > D_P(i, P_{i+1})$. This is because $D_P(i, m)$ can only decrease as i increases, meaning that if some m has a lower value at i than P_i, this property must hold for some index between i and when m appears in P. Finally, all non-greedy solutions must have some $\hat{i}$ that is the maximum i value for which $D_P(i, P_i) > D_P(i, P_{i+1})$ is true.

Consider the optimal ordering $P^\star$ that is not greedy and that has the minimum value for $\hat{i}$. Swapping $P^\star_{\hat{i}}$ and $P^\star_{\hat{i}+1}$ cannot change the minimum dependency of any other moves. Performing the swap causes $P^\star_{\hat{i}}$ to have the minimum dependency $P^\star_{\hat{i}+1}$ had before the swap, while $P^\star_{\hat{i}+1}$'s minimum dependency is raised by at least $D_{P^\star}(\hat{i}, P^\star_{\hat{i}}) - D_{P^\star}(\hat{i}, P^\star_{\hat{i}+1})$. This value cannot be negative, meaning that we have now constructed a solution that either skips more solutions than $P^\star$ or has a lower $\hat{i}$ than $P^\star$, contradicting our assertions. □

Theorem 2. *Not all greedy orderings are optimal orderings.*

Proof. Consider a problem with move dependencies $m_0 = \{A, B\}$, $m_1 = \{C, D\}$, and $m_2 = \{C, D, E\}$. There are two greedy orderings, $[m_0, m_1, m_2]$ and $[m_1, m_2, m_0]$. The former skips $2^3 + 2^1$ solutions, while the latter skips $2^3 + 2^2$. This is because m_1 and m_2 overlap. Even though both are greedy, only the second is optimal. □

We suspect finding the optimal ordering of moves is NP-Hard and therefore rely on the greedy method as it is probabilistically optimal.

6 Complexity Classes for Simple Landscapes

Understanding the runtime complexity of Algorithm 1 requires knowledge of how often each move is fitness-improving. For most landscapes, this is intractable to do theoretically. However, for some restricted problem types it is possible to rigorously determine Algorithm 1's complexity class.

6.1 Linear Functions

A linear function is any $f : \mathbb{B}^N \to \mathbb{R}$ which contains no non-linear terms between variables. These functions are of the form:

$$f(x) = \sum_{i=0}^{N-1} w_i x_i \tag{4}$$

In Mk Landscape terms, $M = N$ and $k = 1$. The most well known linear function is OneMax, in which $w_i = 1$ for all i.

Theorem 3. *Algorithm 1 finds all local optima of any linear function where* $\forall_i w_i \neq 0$ *in* $\mathcal{O}(N)$ *time.*

Proof. When Algorithm 1 is applied to a linear function, an index i of *move_bin* is non-zero if and only if x_i disagrees with the global optimum. Initially *index* is set to $N - 1$, and is decreased by at least one in every step after the first. Each step will flip a bit that disagrees with the global optimum. No carry operations can occur before finding the global optimum, meaning the cost of each iteration is equal to the amount by which *index* is decreased. Therefore, Algorithm 1 requires $\mathcal{O}(N)$ time to reach the global optimum.

After finding the global optimum, *index* is set to 0. One iteration is then spent adding a 1 to *index* 0, which in the worst case requires $\mathcal{O}(N)$ carry operations. For all future iterations, *index* is the position of the highest fitness-improving move which is currently set to 1. In these iterations at least 1 carry operation must occur, and *index* cannot be decreased. Iteration ends when *index* exceeds N, which requires at most N carry operations. Therefore, Algorithm 1 requires $\mathcal{O}(N)$ time to reach termination after finding the global optimum. Combined with initialization and the time to find the global optimum, the total complexity is $\mathcal{O}(N)$, which is the lowest possible bound for this problem. □

6.2 k-Bound Separable Problems

A k-bound separable problem is any problem that is composed of non-overlapping subfunctions, each using k or fewer bits. Formally this means $\forall_i |s_i| \leq k$ and $\forall_{i \neq j} s_i \cap s_j = \emptyset$. Let l_i be the set of ways f_i can be set such that it contains no fitness improving moves.

Theorem 4. *Algorithm 1 with reordering finds all* $\prod_{i=0}^{M-1} |l_i|$ *local optima in a* k*-bound problem in at most* $2^k \sum_{j=0}^{M-1} \prod_{i=M-j}^{M-1} |l_i| + \mathcal{O}(M)$ *time.*

Proof. While there is no restriction on how variables are ordered in the problem, performing reordering will ensure that all variables that appear in the same s_i are consecutive. As such, it is possible to create a numbering of f_i such that if $i < j$ then all of s_i's variables appear before s_j's in enumeration ordering.

Algorithm 1 finds the f_i with the highest i that contains a fitness-improving move and enumerates its variables until it no longer contains an improving move. In total, enumerating f_i requires at most 2^k steps. Each f_i is considered sequentially until none contains an improving move, meaning the first local optimum is found in $\mathcal{O}(2^k M)$ time.

Once the first local optimum is found, Algorithm 1 proceeds by finding all local optima in larger and larger hyperplanes. Initially, f_0 is enumerated to find all local optima in the hyperplane where all $f_i, i > 0$ remain fixed. This process ends when a carry operation modifies a bit of f_1. At this point, f_1 is enumerated such that each time f_1 contains no improving moves f_0 is enumerated again. This finds all local optima in the hyperplane where $f_i, i > 1$ remain fixed. $T(i)$ represents the time required to find all local optima in the hyperplane with $f_j, j > i$ fixed. $T(0) = 2^k$ as all values of f_0 must be enumerated, and $T(i) = |l_i| * T(i-1) + 2^k$. Each setting in l_i exposes a new hyperplane that can contain local optima, which causes the recursive call to $T(i-1)$. The time required to find all local optima is therefore $T(M-1) = 2^k \sum_{j=0}^{M-1} \prod_{i=M-j}^{M-1} |l_i|$. □

When looking for only r-bit local optima of k-bound separable problems, Algorithm 1's complexity only increases by a constant. For $r < k$, the additional moves may reduce $|l_i|$ without increasing the cost by more than a constant. When $r \geq k$ there can only be one local optimum: the global optimum. Furthermore, as only non-linearly related subsets of variables must be checked for fitness improvements, no subsets can be added that are larger than k. Therefore, even if $r = N - 1$, the number of subsets only grows at $\mathcal{O}(N)$. Because all $|l_i| = 1$ in this case, the cost of finding the single global optimum only requires the $\mathcal{O}(M)$ time, which is optimal.

Theorem 5. *If $\forall_i |l_i| = c$ and $c > 1$ in a k-bound problem, Algorithm 1 with reordering finds all c^M local optima in at most $2^k c^M + \mathcal{O}(M)$ time.*

Proof. From Theorem 4, the number of local optima can be expressed as $\prod_{i=0}^{M-1} |l_i| = \prod_{i=0}^{M-1} c = c^M$. Similarly $2^k \sum_{j=0}^{M-1} \prod_{i=M-j}^{M-1} |l_i| = 2^k \sum_{j=0}^{M-1} c^{j-1} < 2^k c^M$. □

Concatenated Traps is a commonly used k-bound separable problem with $c = 2$ and $M = N/k$. Algorithm 1 requires $2^k 2^{N/k}$ time to find all $2^{N/k}$ local optima in this problem. Note also that in the degenerate case of $k = 1$, a k-bound separable problem is a linear function. As such, when $k = 1$ and $c = 1$, $2^k c^M + \mathcal{O}(M) = \mathcal{O}(N)$.

Importance of Reordering. The order of variables for k-bound separable problems has a significant impact on Algorithm 1's complexity. Instead of using the normal reordering procedure, consider an ordering such that if $i < j$, all variables in s_i appear before the second variable in s_j and after the first variable in s_j.

In order to flip the second variable in s_i, all of s_{i-1} must be enumerated. This is because only carry operations can result in *index* being set to the second

variable in s_i. Each time f_i contains a fitness-improving move, *index* is set to the first variable in s_i, which comes before all variables in s_{i-1}. As a result, the time to solve this ordering becomes $T(i) = 2^{k-1} * T(i-1) + 2^k$. Therefore, if $l_i < 2^{k-1}$ this ordering performs much worse than optimal.

7 Experiments

We compare three methods for finding all local optima: Gray-Box, Hyper, and Hyper-Reorder. The Gray-Box method refers to previous work [9] that does not perform hyperplane elimination. Hyper is Algorithm 1 without the order improvements discuss ed in Sect. 5. Hyper-Reorder is Algorithm 1 with reordering.

For each of the five types of Mk Landscape outlined in Sect. 2, we tested all algorithms for all N in [15..100] and r in [1..4]. For each problem size we generated 30 problem instances, with each r of each algorithm run once per instance. Variables were randomly ordered in the genome for problems with natural structure. For example, variables in the same s_i of an Adjacent NKq instance are not likely to be consecutive in the genome. However, s_i still overlaps s_{i+1} in all but 1 variable.

Each algorithm was given a maximum of four hours to find all r-bit local optima. Runs were distributed across a cluster using 2.5 GHz Intel Xeon E5-2670v2 processors. Any run which failed was rerun once to prevent cluster issues from introducing noise. For the same reason, any run which was over ten times slower than the next slowest run of the same configuration with a different seed was rerun. This resulted in 33 out of 36,000 runs being rerun. If the algorithm failed to complete in the alloted time for an instance, we declared it unsuccessful for that combination of r and N of that problem. All results we report are for successful combinations. For timing purposes, local optima were counted but not recorded. All of our code, data, and statistical analysis can be downloaded from our website.[1]

7.1 Finding Local Optima

We tested 932 task configurations (problem, N, r), with 681 completed by at least one method. Of those, Hyper-Reorder had the lowest mean time to find all local optima in all but 2. In both of those cases, Hyper-Reorder was within a tenth of a second of the best method. Figure 2 shows the variance in runtime for the largest problem size where all three methods were successful. In all cases it is clear that Hyper-Reorder outperforms the other methods.

More critically, Fig. 3 shows how each method scales with problem size. Not only does Hyper-Reorder outperform the alternatives on all sizes, the rate at which it scales to problem complexity is better. Table 1 shows the model fit for each method on each problem. This value is the slope of each line shown in Fig. 3. As expected, Gray-Box scales at almost exactly 2^N. Using hyperplane

[1] https://github.com/brianwgoldman/Enumerate-Local-Optima/releases.

elimination without reordering reduces this complexity, and reordering makes further improvements. In the final column we show the growth rate of the number of local optima. It is impossible for any method that enumerates all local optima to grow slower than this rate. However, Hyper-Reorder has an estimated slope better than optimal on Concatenated Traps, Adjacent NKq, and Ising Spin Glass. This is due to lower order terms dominating runtime for small problem sizes, as can be see in Fig. 3. For instance, fitting the curve only to problems where $N > 60$ makes Hyper-Reorder have $m = 0.199$ on Concatenated Traps and $m = 0.358$ on Adjacent NKq. For Concatenated Traps this matches our theoretical predictions that Hyper-Reorder scales at $2^k 2^{N/k}$ as $\frac{1}{k} = 0.2$.

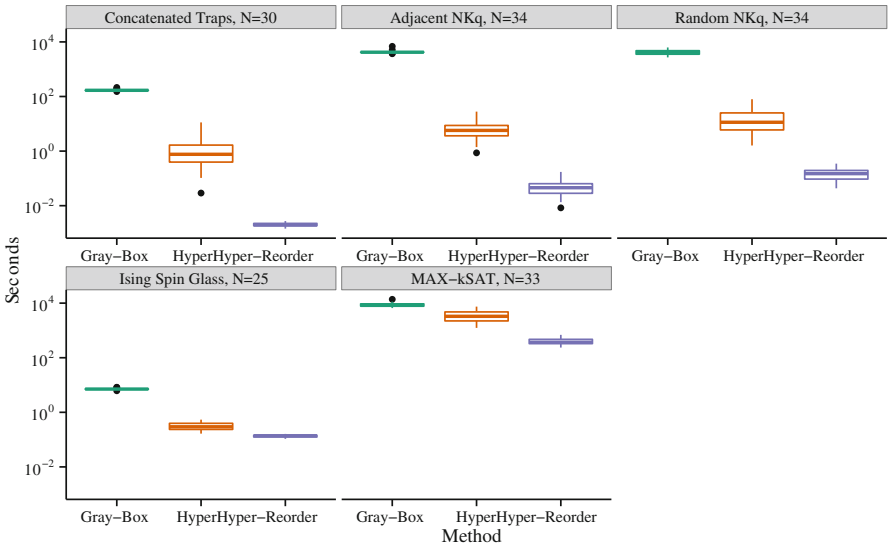


Fig. 2. Comparison of completion time variance for the largest size of each problem where all three methods were successful at finding all 1-bit local optima.

This reduction in growth complexity translates into a substantial increase in the size of problems which can be enumerated in reasonable time. For instance, using the previous best enumeration methods [9], the largest Adjacent NKq problem size that can be enumerated in 4 h is 34 bits. Given the same amount of time, Hyper-Reorder can enumerate problems with 77 bits. Using the regression model, we estimate solving a 77 bit problem using Gray-Box would require 900 million years. Similarly, Hyper-Reorder's largest Random NKq size of 69 bits would require Gray-Box 3 million years. We believe these larger problems are more likely to share characteristics with those where search algorithms are actually applied.

With the exception of MAX-kSAT, Hyper-Reorder's time required to find all local optima appears to be growing at the same rate as the total number

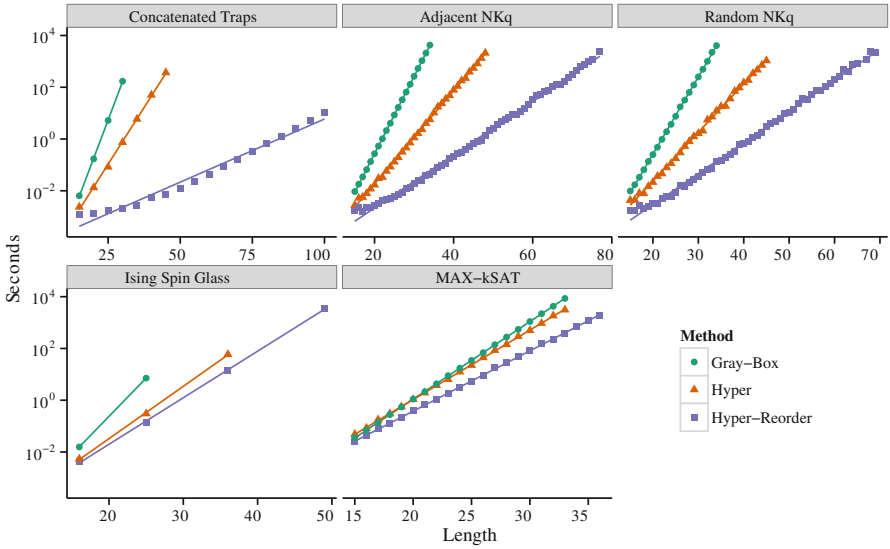


Fig. 3. Comparison of how each method scales with problem size when finding 1-bit local optima on log-linear scales. Each point is the mean runtime over 30 instances and each line the linear model. All confidence intervals too tight to see.

Table 1. Value of m when fitting a model to $y = 2^c 2^{mN}$ where y is number of seconds required to find all 1-bit local optima of a problem using N bits. Optimal refers to the same model where y is the number of local optima. All R^2 values over 0.95 except Hyper on Concatenated Traps which has 0.898.

	Gray-Box	Hyper	Hyper-Reorder	Optimal
Concatenated Traps	0.9809	0.5831	0.1620	0.2000
Adjacent NKq	0.9922	0.5972	0.3419	0.3603
Random NKq	0.9920	0.6229	0.3989	0.3531
Ising Spin Glass	0.9821	0.6722	0.5976	0.6015
MAX-kSAT	0.9949	0.8922	0.7737	0.5393

of local optima. We suspect some of this deficiency on MAX-kSAT is due to its significantly higher number of non-linear interactions. Concatenated Traps, Adjacent NKq, and Ising Spin Glass all have exactly 4 non-linear relationships per variable. For small problem sizes, the number of non-linear relationships per variable in Random NKq and MAX-kSAT both increase with problem size. Random NKq's average ranges between 5 and 6 on problems we tested. With $N = 15$, MAX-kSAT has, on average, nearly 12 non-linear relationships per variable. By $N = 30$, this grows to just over 17. Having each variable depend on over half the genome significantly limits the size of hyperplanes that can be eliminated. While

the number of non-linear relationships per variable is asymptotically constant, this growth on small problems is likely affecting runtime.

7.2 Finding r-Bit Local Optima

A major advantage of Gray-Box enumeration is that it can find r-bit local optima with only a constant increase in runtime. Figure 4 shows the estimated growth rate for each method as r increases. On Concatenated Traps, Adjacent NKq, Ising Spin Glass, and to a lesser extent Random NKq, none of our enumeration methods have an increase in growth complexity with increasing r. In all cases, using Hyper-Reorder has a lower growth rate than the other two methods using the same r. On every problem except MAX-kSAT, Hyper-Reorder's slowest r is faster than the fastest r of any other configuration.

Perhaps most striking in Fig. 4 is that on Adjacent NKq, Random NKq, and Ising Spin Glass, increasing r **reduces** Hyper-Reorder's growth complexity. Increasing r creates more moves which can be fitness-improving. As a result there are more hyperplanes that can potentially be detected and eliminated. On Concatenated Traps, no flip using $1 < r < k$ bits can be fitness-improving without a 1-bit flip being fitness-improving, meaning no additional hyperplanes can be skipped. For MAX-kSAT, increasing r likely compounds the high dependency problems discussed in Sect. 7.1.

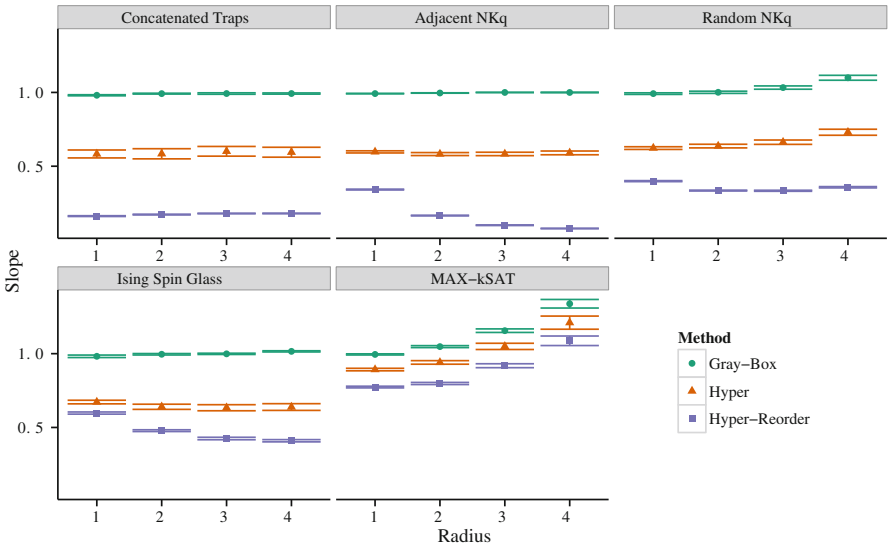


Fig. 4. Estimated slope with 95 % confidence intervals for each method to find r-bit local optima. Slope is m in the model $y = 2^c 2^{mN}$ where y is number of seconds required to find all local optima of a problem using N bits.

Increasing r also results in a substantial reduction in the total number of local optima, as shown in Fig. 5. For every problem except Concatenated Traps, increasing r not only reduces the number of local optima, it decreases the growth rate with respect to N. As there are fewer local optima to enumerate, Algorithm 1 requires less time to enumerate them all. In Concatenated Traps, each trap requires exactly k flips in order to move between local optima. Therefore, while $r < k$ the number of local optima cannot change, and once $r \geq k$ there is exactly one local optima.

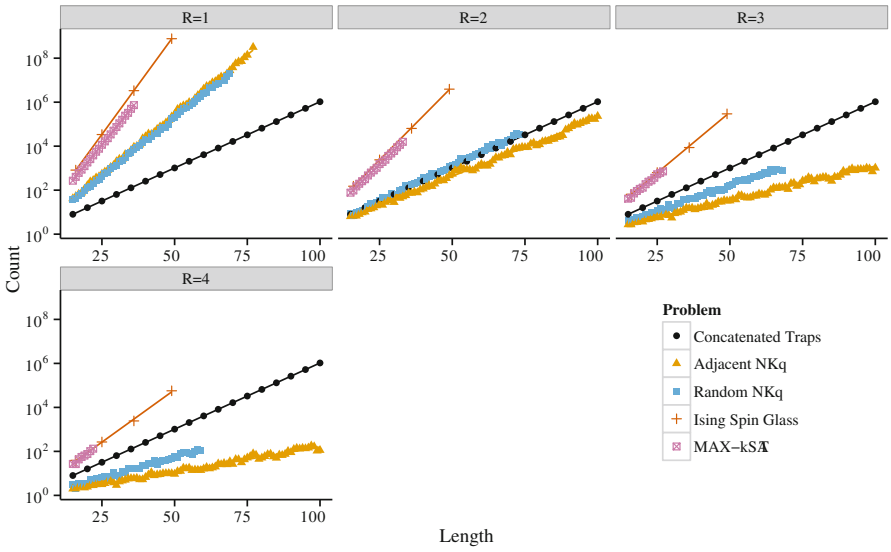


Fig. 5. Number of r-bit local optima for each size of each problem. Note that the number of local optima in Concatenated Traps does not change while $r < k$.

Previous work [9] noted that, counter-intuitively, Adjacent NKq has more local optima than Random NKq for $r = 1$ and $N = 20$, even though the former is generally easier to solve. However, many of these optima are "weak" in the sense that they are not also 2-bit local optima. Here we extend those observations for up to $N = 59$ and up to $r = 4$. Adjacent NKq tends to have more 1-bit local optima than Random NKq, most of which disappear as r increases. Furthermore, the growth rate of local optima between the two landscapes diverges as r increases.

MAX-kSAT and Ising Spin Glasses generally appear to contain far more local optima than any of the other problems we enumerated. With just $N = 49$, Ising Spin Glasses average nearly 800 million local optima. For comparison, Adjacent NKq with $N = 49$ averages fewer than 300 thousand, and with $N = 77$ Adjacent NKq still has fewer than 500 million. As with NKq, the polynomially-solvable Ising Spin Glass problem has more 1-bit local optima than the NP-Hard

MAX-kSAT. Unlike with NKq, this relationship generally does not change as r increases. However, both do still see large decreases in the number and growth rate of local optima. With $N = 49$ the number of 4-bit local optima in an Ising Spin Glass is on average under 70 thousand, a reduction of four orders of magnitude.

In our original experiments we tested problems up to size $N = 100$, which was sufficient to cause all three methods using $r = 1$ to fail all problems except for Hyper-Reorder on Concatenated Traps. For Gray-Box and Hyper, increasing r only decreased the largest size they could enumerate. However, using higher r values, Hyper-Reorder is able to significantly extend its range. On Random NKq, Hyper-Reorder was successful at finding 2-bit local optima for $N = 73$. On Adjacent NKq for all r tested except 1, Hyper-Reorder was successful on $N = 100$. To push Algorithm 1 to the extreme, we tested Adjacent NKq with $N = 200$ and $r = 4$, with all 30 runs successful. On average, these runs required 5.6 min to complete. However, there was substantial variance in both the runtime and number of local optima. Eighteen runs completed in under one minute, while the slowest run took 1.5 h. Three had fewer than 50 4-bit local optima, while five had over 100,000.

8 Conclusions and Future Work

By expressing a function as an Mk Landscape, we have developed a method to find all local optima of that function very efficiently. The structure imposed by an Mk Landscape allows for the identification of hyperplanes in the search space that cannot contain any local optima and can therefore be eliminated from enumeration. Combined with a greedy method to change the ordering of variables, we were able to show this method's runtime complexity scales with the number of local optima, unlike previous methods that scale with the number of possible solutions. When applied to NKq Landscapes and Ising Spin Glasses, this allowed for a dramatic increase in the size of problems which can be enumerated. We believe that these larger problem sizes will be more representative of the types of search spaces where optimization is performed.

We further refined this hyperplane elimination to find only solutions that cannot be improved by flipping r or fewer bits. For some problems this counterintuitively reduces search times, and on Adjacent NKq allowed us to find all 4-bit local optima of 200 bit problems.

The obvious next step is to use the results of hyperplane elimination in conjunction with landscape analysis techniques [9,13,14]. We are especially interested to see how landscape properties change for larger N and r. As to the algorithm itself, we suspect there may be room for improvement in enumeration ordering. While we are able to prove that all optimal orderings are greedy, the reverse is not true due to how ties are broken. Therefore, more gains may be possible by improving this ordering. Furthermore, we assumed that all moves are equally likely to be fitness-improving. It is likely possible to determine the true probability of a move being fitness-improving, resulting in potentially better ordering.

Acknowledgments. This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Boese, K.D., Kahng, A.B., Muddu, S.: A new adaptive multi-start technique for combinatorial global optimizations. Oper. Res. Lett. **16**(2), 101–113 (1994)
2. Chicano, F., Whitley, D., Sutton, A.M.: Efficient identification of improving moves in a ball for pseudo-boolean problems. In: Genetic and Evolutionary Computation Conference, Vancouver, BC, Canada, pp. 437–444. ACM, 12–16 July 2014
3. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. In: Foundations of Genetic Algorithms Conference, Vail, Colorado, 24–29 July 1992
4. Goldman, B.W., Punch, W.F.: Gray-box optimization using the parameter-less population pyramid. In: Genetic and Evolutionary Computation Conference, pp. 855–862. ACM (2015)
5. Goldman, B.W., Tauritz, D.R.: Linkage tree genetic algorithms: variants and analysis. In: Genetic and Evolutionary Computation Conference, Philadelphia, Pennsylvania, USA, pp. 625–632. ACM, 7–11 July 2012
6. Hsu, S.H., Yu, T.L.: Optimization by pairwise linkage detection, incremental linkage set, and restricted/back mixing: Dsmga-ii. In: Genetic and Evolutionary Computation Conference, pp. 519–526. ACM (2015)
7. Iclănzan, D., Daolio, F., Tomassini, M.: Learning inherent networks from stochastic search methods. In: Blum, C., Ochoa, G. (eds.) EvoCOP 2014. LNCS, vol. 8600, pp. 157–169. Springer, Heidelberg (2014)
8. Inoue, K., Hasegawa, T., Araki, Y., Mori, N., Matsumoto, K.: Adaptive control of parameter-less population pyramid on the local distribution of inferior individuals. In: Genetic and Evolutionary Computation Conference, pp. 863–870. ACM (2015)
9. Ochoa, G., Chicano, F., Tinós, R., Whitley, D.: Tunnelling crossover networks. In: Genetic and Evolutionary Computation Conference, pp. 449–456. ACM (2015)
10. Saul, L., Kardar, M.: The 2d±j ising spin glass: exact partition functions in polynomial time. Nucl. Phys. B **432**(3), 641–667 (1994)
11. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. Artif. Intell. **81**(1–2), 17–29 (1996)
12. Tintos, R., Whitley, D., Chicano, F.: Partition crossover for pseudo-boolean optimization. In: Foundations of Genetic Algorithms Conference, pp. 137–149. ACM (2015)
13. Tomassini, M., Verel, S., Ochoa, G.: Complex-network analysis of combinatorial spaces: the n k landscape case. Phys. Rev. E **78**(6), 066114 (2008)
14. Verel, S., Ochoa, G., Tomassini, M.: Local optima networks of nk landscapes with neutrality. IEEE Trans. Evol. Comput. **15**(6), 783–797 (2011)
15. Whitley, D.: Mk landscapes, nk landscapes, max-ksat: a proof that the only challenging problems are deceptive. In: Genetic and Evolutionary Computation Conference, pp. 927–934. ACM (2015)
16. Whitley, D., Chen, W.: Constant time steepest descent local search with lookahead for nk-landscapes and max-ksat. In: Genetic and Evolutionary Computation Conference, pp. 1357–1364. ACM (2012)
17. Wright, A.H., Thompson, R.K., Zhang, J.: The computational complexity of N-K fitness functions. IEEE Trans. Evol. Comput. **4**(4), 373–379 (2000)

Limits to Learning in Reinforcement Learning Hyper-heuristics

Fawaz Alanazi(✉) and Per Kristian Lehre

School of Computer Science, University of Nottingham, Nottingham, UK
{psxfa1,pszpl}@nottingham.ac.uk

Abstract. Learning mechanisms in selection hyper-heuristics are used to identify the most appropriate subset of heuristics when solving a given problem. Several experimental studies have used *additive* reinforcement learning mechanisms, however, these are inconclusive with regard to the performance of selection hyper-heuristics with these learning mechanisms. This paper points out limitations to learning with *additive* reinforcement learning mechanisms. Our theoretical results show that if the probability of improving the candidate solution in each point of the search process is less than 1/2 which is a mild assumption, then *additive* reinforcement learning mechanisms perform asymptotically similar to the *simple random* mechanism which chooses heuristics uniformly at random. In addition, frequently used adaptation schemes can affect the memory of reinforcement learning mechanisms negatively. We also conducted experiments on two well-known combinatorial optimisation problems, *bin-packing* and *flow-shop*, and the obtained results confirm the theoretical findings. This study suggests that alternatives to the *additive* updates in reinforcement learning mechanisms should be considered.

Keywords: Hyper-heuristics · Reinforcement learning · Runtime analysis

1 Introduction

The term 'hyper-heuristic' refers to a search methodology that automatically selects or generates heuristics to solve a given optimisation problem. One main goal of this line of research is to design general-purpose optimisation methods that can adapt to different problem instances and domains. There are two main categories of hyper-heuristics; *selection* hyper-heuristics and *generation* hyper-heuristics. This paper focuses on the former. Typically, a selection hyper-heuristic framework evolves an initial randomly generated solution by iteratively (i) selecting a heuristic from a repository of pre-chosen low-level heuristics; (ii) applying the selected heuristic, thus generating new offspring; (iii) evaluating the offspring and deciding whether it should replace the current candidate solution or be discarded. This process is iterated until a predetermined termination condition is satisfied. A growing body of literature has assessed the effectiveness of applications of hyper-heuristics on different optimisation problems. An overview can be found in Burke et al. [4].

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 170–185, 2016.
DOI: 10.1007/978-3-319-30698-8_12

Learning mechanisms in selection hyper-heuristics are heuristic selection techniques that specify which heuristic to apply in a given point of the search process. Reinforcement learning is sufficiently general in nature, and thus describes several learning mechanisms [14]. Conceptually, all reinforcement learning mechanisms aim to iteratively solve the appropriate heuristics identification task by trial-and-error interactions with the search environment. Traditional reinforcement learning hyper-heuristics associate every heuristic with a weight (rank), and each heuristic is subject to a positive or a negative reinforcement based on its performance [16]. This reinforcement increases or decreases the probability of the heuristic being chosen in the next points of the search process (e.g., each heuristic is chosen with probability proportional to its weight). All the weights of low-level heuristics are initialised to the same value and adapted during the search process by a pre-chosen weights adaptation scheme. The adaptation scheme determines how the weights of the heuristics should be updated. The *additive* weights adaptation scheme is the most frequently used adaptation scheme which combines slow additive and subtractive adaptation rates so that, if an improvement in the candidate solution is found, the weight of the selected heuristic is increased by a small value; otherwise, the weight is minimally degraded by of same value. Several experimental studies have used the reinforcement learning mechanisms on different search and optimisation problems. However, non-conclusive results about the performance of hyper-heuristics with the reinforcement learning mechanisms were obtained (see e.g. [3,5,8,16,18,19]).

This study investigates theoretically the learning behaviour of reinforcement learning mechanisms in hyper-heuristics. A theoretical foundation of learning mechanisms in hyper-heuristics is largely absent. Lehre and Özcan [15] showed that mixing low-level heuristics can be necessary for certain benchmark problems. He et al. [11], demonstrated that $(1 + 1)$ Evolutionary Algorithms (EAs) that combine several mutation operators can outperform the $(1 + 1)$ EAs that use any one of the operators alone. Alanazi and Lehre [1] showed that several classical learning mechanisms have roughly the same performance on both a benchmark problem and a general model of fitness landscapes. To the best of our knowledge, this is one of the first theoretical studies of reinforcement learning mechanisms within a hyper-heuristic framework. We perform rigorous runtime analyses (see e.g. [13]) to investigate how reinforcement learning mechanisms choose low-level heuristics. Additionally, we investigate empirically the performance of the *additive* reinforcement learning mechanism on a set of instances of two different problems.

The paper is organised as follows: Sect. 2 presents notation used in this paper; Sect. 3 describes briefly reinforcement learning hype-heuristics; Sect. 4 provides a theoretical investigation of the reinforcement learning mechanisms; Sect. 5 presents an empirical investigation of additive reinforcement learning mechanisms; Sect. 6 draws the conclusions of this study.

2 Notation

This paper uses the following notation. For any integer n, let $[n] := \{1, \ldots, n\}$. We use $s \sim \text{Unif}(\mathcal{S})$ to signify that s is sampled uniformly at random from $\mathcal{S}$.

Standard asymptotic notation (e.g. O, Ω, Θ) is used (see e.g. [6]). An event occurs with high probability (w.h.p.) with respect to a parameter n, if the probability of the event is bounded from below by at least $1 - O(1/n)$.

3 Reinforcement Learning Hyper-heuristics

We begin by briefly reviewing reinforcement learning hyper-heuristics [16]. The framework consists of a finite set of states $\mathcal{S}$, an objective function $f : \mathcal{S} \to \mathbb{R}$ mapping states to real numbers, a set $H := \{h_1, \ldots, h_m\}$ of m low-level heuristics, and an adaptation scheme $A : \mathcal{S} \times H \to \mathbb{R}$. We define low-level heuristics as follows.

Definition 1. *Given a finite set $\mathcal{S}$ and an integer $k \geq 1$, a low-level heuristic h of* arity *k is a random mapping*

$$h : \mathcal{S}^k \to \mathcal{S}$$

A low-level heuristic in this paper is a function that generates a new search point given one or more search points. Reinforcement learning mechanisms associate each low-level heuristic $i \in [m]$ with a positive weight. Denote by $w_i^{(t)}$ the weight of heuristic i in iteration t. Initially, $w_i^{(0)} = w_j^{(0)}$ for all $i, j \in [m]$. The weight of each heuristic is restricted to be within a user-defined interval $[w_{\min}, w_{\max}]$ such that $w_i^{(t)} \in [w_{\min}, w_{\max}]$ for all $i \in [m]$ and $t \geq 0$. At each iteration t, a heuristic selection strategy is used to decide which heuristic to apply based on a distribution $\mathbf{p}^{(t)} := (p_1^{(t)}, \ldots, p_m^{(t)})$. For example, the *roulette wheel* selection strategy chooses each heuristic with probability proportional to its weight, i.e., $p_i^{(t)} := \frac{w_i^{(t)}}{\sum_{j=1}^{m} w_j^{(t)}}$. The reinforcement learning hyper-heuristic is given in Algorithm 1. For our theoretical investigation, the *additive* adaptation scheme is considered (see line 11 in Algorithm 1). The adaptation rates α and β are usually fixed beforehand, however, they can also be functions of the current heuristic weight or of the heuristic performance. The move-acceptance strategy determines whether to accept or reject the new generated solution. We assume a maximisation problem in all algorithms in this paper.

4 Theoretical Analysis

This section theoretically investigates the learning behaviour of Algorithm 1. Due to the space restriction, all technical proofs are omitted. These proofs employ standard techniques from runtime analysis [9,10,12]. We use the *simple random* hyper-heuristic as a baseline hyper-heuristic with which Algorithm 1 is compared. The *simple random* hyper-heuristic [7] chooses low-level heuristics uniformly at random, and hence it does not learn (see Algorithm 2).

Obviously, the user-defined bounds of the weights ($w_{\min}$, $w_{\max}$) also imply bounds on the selection probabilities of low-level heuristics. Proposition 2 shows

Algorithm 1. Reinforcement Learning Hyper-heuristic

Given a finite set $\mathcal{S}$, and an objective function $f : \mathcal{S} \to \mathbb{R}$.
Let $H := \{h_1, \ldots, h_m\}$ be a set of m low-level heuristics, where $h_i : \mathcal{S} \to \mathcal{S}$.
Fix $w_{\min}$ and $w_{\max}$ such that $w_{\min}, w_{\max} \geq 0$.
Let α and $\beta \in [0, w_{\max}]$ be a rewarding and punishing rates respectively.
For all $i \in [m]$, let $w_i^{(0)} := w_{\min}$.
Let $s \sim \text{Unif}(\mathcal{S})$ be an initial solution generated uniformly at random.
Let τ be the maximum admissible number of iterations.
for $t = 0, 1, \ldots, \tau$ **do**
  Pick heuristic $i \in [m]$, with probability $p_i^{(t)} := \frac{w_i^{(t)}}{\sum_{j=1}^{m} w_j^{(t)}}$
  $s' := h_i(s)$
  $w_i^{(t+1)} := \begin{cases} \min(w_i^{(t)} + \alpha\ , w_{\max}) & \text{if } f(s') > f(s)\ \text{, and} \\ \max(w_i^{(t)} - \beta\ , w_{\min}) & \text{otherwise} \end{cases}$
  if move-acceptance(s, s') **then**
    $s := s'$
  end if
end for

Algorithm 2. Simple Random Hyper-heuristic

Given a finite set $\mathcal{S}$, and an objective function $f : \mathcal{S} \to \mathbb{R}$.
Let $H := \{h_1, \ldots, h_m\}$ be a set of m low-level heuristics, where $h_i : \mathcal{S} \to \mathcal{S}$.
Let τ be the maximum admissible number of iterations.
for $t = 0, 1, \ldots, \tau$ **do**
  Pick heuristic $i \in [m]$, with probability $p_i^{(t)} := \frac{1}{m}$
  $s' := h_i(s)$
  if move-acceptance(s, s') **then**
    $s := s'$
  end if
end for

that if these bounds are constant with respect to the number of low-level heuristics m (i.e. $\frac{w_{\max}}{w_{\min}} = O(1)$), then Algorithm 1 assigns asymptotically, w.r.t. the number of heuristics, the same selection probability to each low-level heuristic (i.e. similar to the simple random mechanism).

Proposition 2. *If $w_{\min}$ and $w_{\max}$ are constants with respect to m, then $\forall i \in [m]$ and $t \geq 0$ Algorithm 1 chooses low-level heuristic i with probability $p_i^{(t)} = \Theta\left(\frac{1}{m}\right)$.*

This implies that if these bounds are constants w.r.t. m, then no selection probability is larger than (c/m) for some constant $c > 0$. Typical low-level heuristics perform differently on different search states, and hence Algorithm 1 faces an obvious issue of exploration versus exploitation. The bounds of the weights maintain a non-zero selection probability for each heuristic in every iteration of the algorithm. These bounds, however, are predefined parameters that need to be tuned correctly beforehand.

Definition 3. *Let $\mathcal{S}$ be a finite set, and $f : \mathcal{S} \to \mathbb{R}$ be an objective function. Let $s' \in \mathcal{S}$ be the offspring of $s \in \mathcal{S}$ generated by the low-level heuristic i in iteration t. In Algorithm 1 and 2, the success probability of the low-level heuristic i in iteration t is*

$$q_i^{(t)}(s) := \Pr\left(f\left(h_i(s')\right) > f\left(s\right)\right)$$

The objective of reinforcement learning mechanisms is to learn the effectiveness and the differences between the success probabilities of low-level heuristics. The following proposition shows that if the success probabilities of heuristics are less than 1/2 and the most common settings for the adaptation rates are used, where they are chosen so that $\alpha \leq \beta$, then the probability that the total weights of low-level heuristics is much larger than the initial total weights is small. A consequence of this is that Algorithm 1 generates selection probabilities which are almost the same.

Proposition 4. *Consider Algorithm 1 with $\alpha \leq \beta$. Let $\Phi^{(t)} := \sum_{i=1}^{m} w_j^{(t)}$ be the total weights of low-level heuristics in iteration t. Let $q_i^{(t)}(s)$ be the success probability of heuristic $i \in [m]$ in iteration t. If there exists a constant $\varepsilon \in]0,1]$ such that $q_i^{(t)}(s) \leq \frac{1}{2} \cdot (1-\varepsilon)$ for all $s \in \mathcal{S}$, $i \in [m]$ and $t \geq 0$, then for every iteration $t \geq 0$*

$$\Pr(\Phi^{(t)} \geq \Phi^{(0)} + k \cdot \alpha) \leq 2(1-\varepsilon)^{k\alpha} \tag{1}$$

The proof idea is based on tracking the progress in the total weights of low-level heuristics using so-called drift analysis. It follows from Proposition 4 that if the success probabilities of low-level heuristics are less than 1/2, then the weight of each heuristic is, with high probability, not much larger than the minimum weight. Formally, for any $i \in [m]$ and $t \geq 0$ we have

$$\Pr(w_i^{(t)} \geq w_i^{(0)} + k \cdot \alpha) \leq 2(1-\varepsilon)^{k\alpha}. \tag{2}$$

Corollary 5 shows that if the success probabilities of heuristics are less than $\frac{1}{2}(1-\varepsilon)$, then reinforcement learning mechanisms with the *additive* adaptation scheme choose low-level heuristics almost uniformly at random.

Corollary 5. *Suppose that $w_{\min}$ and α are constants. If there exists a constant $\varepsilon \in]0,1]$ such that $q_i^{(t)}(s) \leq (1/2)(1-\varepsilon)$ for all $s \in \mathcal{S}$, $i \in [m]$ and $t \geq 0$, then for any $\delta > 0$, $i \in [m]$, and $t \geq 0$, with probability at least $1 - 2(1-\varepsilon)^{\alpha m^{\delta}}$, $p_i^{(t)} = O\left(m^{-(1-\delta)}\right)$ and $\mathbb{E}[p_i^{(t)}] = O\left(1/m\right)$.*

The success probabilities of heuristics are small (i.e. less than 1/2), so if α is no larger than β, the reinforcement learning mechanism with roulette wheel selection strategy performs asymptotically similar to the simple random mechanism. The weights of heuristics quickly converge to the minimum weight if their success probabilities are less than 1/2. As a result, Algorithm 1 assigns roughly the same probability to each heuristic. Corollary 5 also shows that the probability that

Algorithm 1 chooses roughly low-level heuristics with probability $\frac{1}{m}$ increases as the number of low-level heuristics m increases. Therefore, alternatives to the roulette wheel selection strategy should be considered.

The so-called *max* strategy has been used as an alternative to the *roulette wheel* selection strategy (see line 7 in Algorithm 1). It deterministically chooses the heuristic with the maximal weight (see e.g. [5,18]).

Proposition 6. *Suppose that Algorithm 1 uses the following heuristic selection strategy:*

$$p_i^{(t)} := \begin{cases} 1 & \text{if } i = \arg\max_{j \in [m]} \{w_j^{(t)}\}, \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

Let $\alpha := \beta$, *and* $k := \frac{w_{\max} - w_{\min}}{\alpha}$. *Assume that* $w_i^{(t)} := w_{\min} + \alpha$ *and* $w_j^{(t)} = w_{\min}$ $\forall j \in [m]$ *and* $j \neq i$. *Let* $T_{w_{\min}} := \min\{t' | w_i^{t+t'} = w_{\min}\}$. *If there exists a constant* $\varepsilon \in]0,1]$ *such that* $q_i^{(t)}(s) \leq \frac{1}{2}(1-\varepsilon)$ *for any* $s \in \mathcal{S}$, $t \geq 0$, *then with probability at least* $1 - \left(\frac{1+\epsilon}{1-\epsilon}\right)^{w_i^{(t)}-k}$, *the weight reaches* $w_i^{(t')} = w_{\min}$ *before* $w_i^{(t')} = w_{\max}$ *and*

$$\mathbb{E}[T_{w_{\min}} | w_i^{(t)} = w_{\min} + \alpha] \leq \frac{1}{\varepsilon} \left(w_i^{(t)} + k \cdot \left(\frac{1+\varepsilon}{1-\varepsilon} \right)^{w_i^{(t)}-k} \right)$$

We consider the scenario of the so-called *gambler's ruin* problem [9] to prove Proposition 6. When $w_i^{(t)} > w_{\min}$, Proposition 6 suggests that the expected number of iterations until the weight of heuristic i returns to the minimum weight, and hence the reinforcement learning mechanism starts choosing heuristics uniformly at random, is small if the success probability of heuristic i is smaller than $1/2$.

4.1 Runtime Analysis

Runtime analysis is a theoretical investigation that evaluates the efficiency of optimisation algorithms through their expected runtimes and probabilities of finding satisfactory solutions [2,13]. The runtime of an algorithm is the number of times the objective function is evaluated until the algorithm produces an optimal solution for the first time. We analyse the expected runtime of the reinforcement learning hyper-heuristic on a simple search scenario. We consider a frequently used theoretical benchmark function in runtime analysis, the so-called LeadingOnes problem.

Definition 7. *For all* $s \in \{0,1\}^n$,

$$\textsc{LeadingOnes}(s) := \sum_{i=1}^{n} \prod_{j=1}^{i} s_j$$

The LeadingOnes value of a bit-string is the number of consecutive, leading 1-bits in the bit-string. In order to improve the candidate solution, at least the left-most 0-bit should be flipped and the leading 1-bits should remain the same. We assume that there are m low-level heuristics, but only one of them has success probability larger than 0 and all the other heuristics have success probability 0. In such a simple search scenario, if the reinforcement learning mechanism is unable to identify the successful heuristic, then it might not be able to identify the appropriate heuristics in more complex problems. The successful low-level heuristic is a mutation operator that flips one bit uniformly at random, and hence its success probability is $1/n$. We use the *improve-or-equal* strategy as move-acceptance operator in both Algorithms 1 and 2.

Definition 8. *Let $\mathcal{S}$ be a finite set, and $f : \mathcal{S} \to \mathbb{R}$ be an objective function. Let s' be the offspring the candidate solution s. In the case of a maximisation problem, the* improve-or-equal *move-acceptance strategy accepts s' if and only if $f(s') \geq f(s)$.*

Suppose that q_i is the success probability of the i-th low-level heuristic. The following lemma shows that if $q_i < 1/8$, then the probability that its weight is larger than $(w_{\min})$ is $O(q_i)$.

Lemma 9. *Suppose that $\alpha = \beta$, if*

1. $\Pr\left(w_i^{(t+1)} = w_{\min} + \alpha | w_i^{(t)} = w_{\min}\right) = \frac{q_i}{m}$
2. $\Pr\left(w_i^{(t+1)} = w_i^{(t)} + \alpha | w_i^{(t)} > w_{\min}\right) = q_i$
3. $\Pr\left(w_i^{(t+1)} = w_i^{(t)} - \beta | w_i^{(t)} > w_{\min}\right) = 1 - q_i$

and $q_i < 1/8$, then

$$\Pr\left(w_i^{(t)} \geq w_{\min} + \alpha\right) = O(q_i)$$

Theorem 10. *The expected runtime of the reinforcement learning hyper-heuristic with both* roulette wheel *and* max *selection strategies with $\alpha = \beta$, and the prescribed settings above on* LeadingOnes *is $\Theta\left(mn^2\right)$.*

Theorem 11. *The expected runtime of the simple random hyper-heuristic with the prescribed settings above on* LeadingOnes *is $\Theta\left(mn^2\right)$.*

Additive reinforcement learning and simple random hyper-heuristics have asymptotically the same expected runtime on the LeadingOnes problem. The analysis shows that the expected runtime of both algorithms increase linearly with the number of low-level heuristics. We make no assumption about the number of low-level heuristics, so m can be a function of n. In this case, the expected runtime of both algorithms are asymptotically different from n^2.

4.2 Frequently Used Adaptation Rates

The adaptation rates in Algorithm 1 are usually set so that $\alpha := 1 =: \beta$ (see e.g. [5,8,18]). As shown above, this adaptation scheme leads reinforcement learning mechanisms to quickly forget how low-level heuristics performed in the recent past iterations if the success probabilities of the low-level heuristics are less than 1/2. The following adaptation scheme was suggested by Nareyek [16], and it has been argued to be the most appropriate adaptation scheme.

Definition 12. *Consider Algorithm 1 with* max *strategy, and let* $w_{\min} := 1$. *For all* $t \geq 0$, *update the weight of the selected heuristic* i *in the iteration* t *as follows.*

$$w_i^{(t+1)} = \begin{cases} \min(w_i^{(t)} + 1\ , w_{\max}) & \text{if } f(s') > f(s), \text{and} \\ \max(\sqrt{w_i^{(t)}}\ , w_{\min}) & \text{otherwise} \end{cases} \tag{3}$$

If Algorithm 1 uses the prescribed adaptation scheme in Definition 12, then the weight of the first rewarded low-level heuristic remains the one with maximal weight, because $\sqrt{x} > 1$ for all $x > 1$. Therefore, Algorithm 1 with the above adaptation scheme chooses the first rewarded heuristic (which is selected uniformly at random) continuously throughout the entire run and ignores the other heuristics. However, this adaptation scheme works due to the fact that the weight of the selected heuristic after sufficiently large number of punishments is eventually rounded to 1 by the computer, and the algorithm starts again choosing low-level heuristics uniformly at random.

5 Empirical Investigations

A set of experiments is conducted to investigate the performance of the reinforcement learning hyper-heuristic on several instances of two different combinatorial optimisation problems. We use common settings for the adaptation rates of the reinforcement learning mechanism, where the rates are set so that $\alpha := 1 =: \beta$. We implement the hyper-heuristics as extensions to the *Hyflex* framework [17]. *Hyflex* is a public object-oriented hyper-heuristic framework that provides several problem domains, each with various instances and a set of low-level heuristics. The performances of the reinforcement learning mechanism and simple random hyper-heuristics are examined on different instances of the *bin-packing* (BP) and *permutation flow-shop* (PF). To provide fair comparisons, each hyper-heuristic is allowed to evaluate the objective function $2 \cdot 10^6$ times. A single run is repeated 1000 times, each run with a different seed. Both hyper-heuristics are used to minimise the objective functions. The experimental results are presented in the following charts.

Figure 1 shows the results of the simple random and reinforcement learning hyper-heuristics on 6 instances of the *bin-packing* problem. As can be seen from the figure, the hyper-heuristics perform almost the same on all instances. Comparable results are obtained on different instances of the *permutation flow-shop*

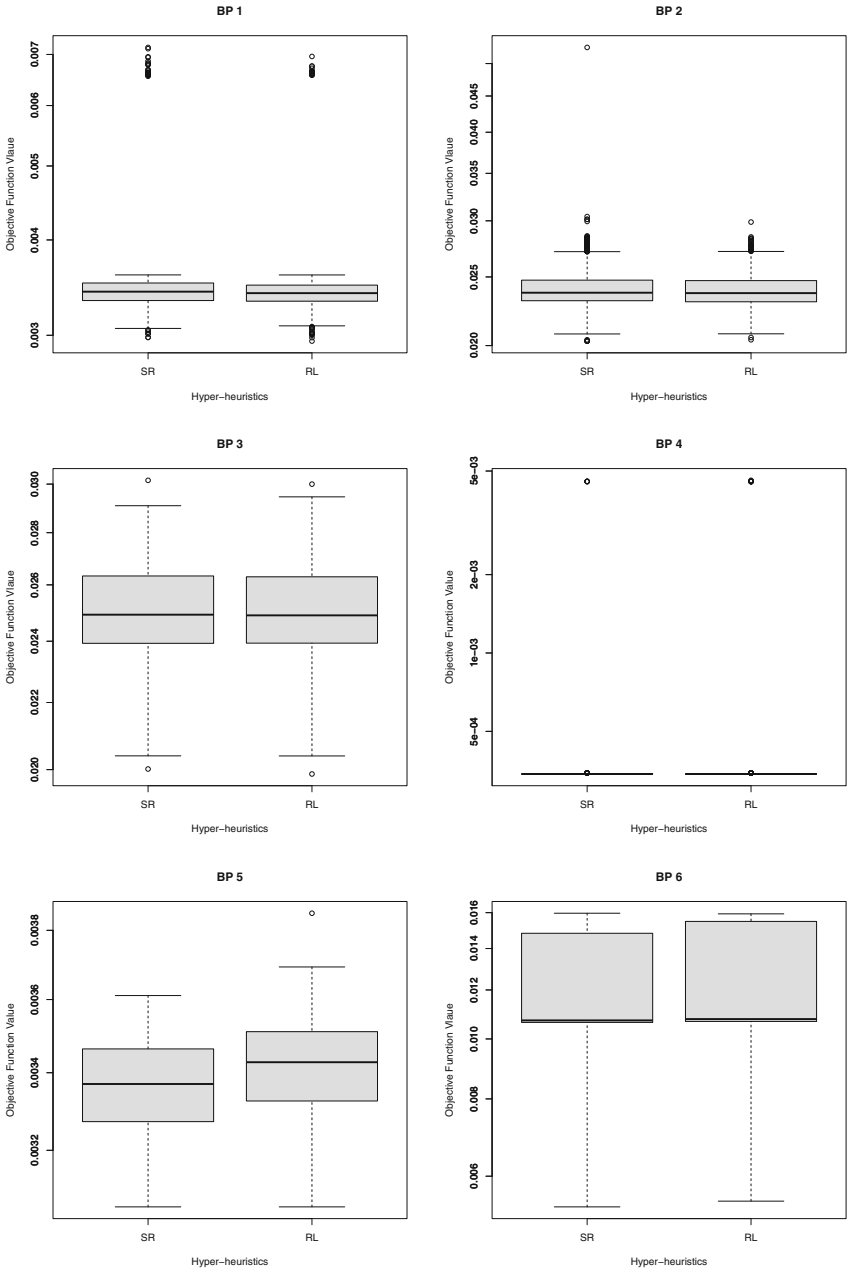


Fig. 1. The box plots show comparisons between the performance of simple random (SR), and reinforcement learning (RL) hyper-heuristics on six instances of the bin-packing (BP) problem.

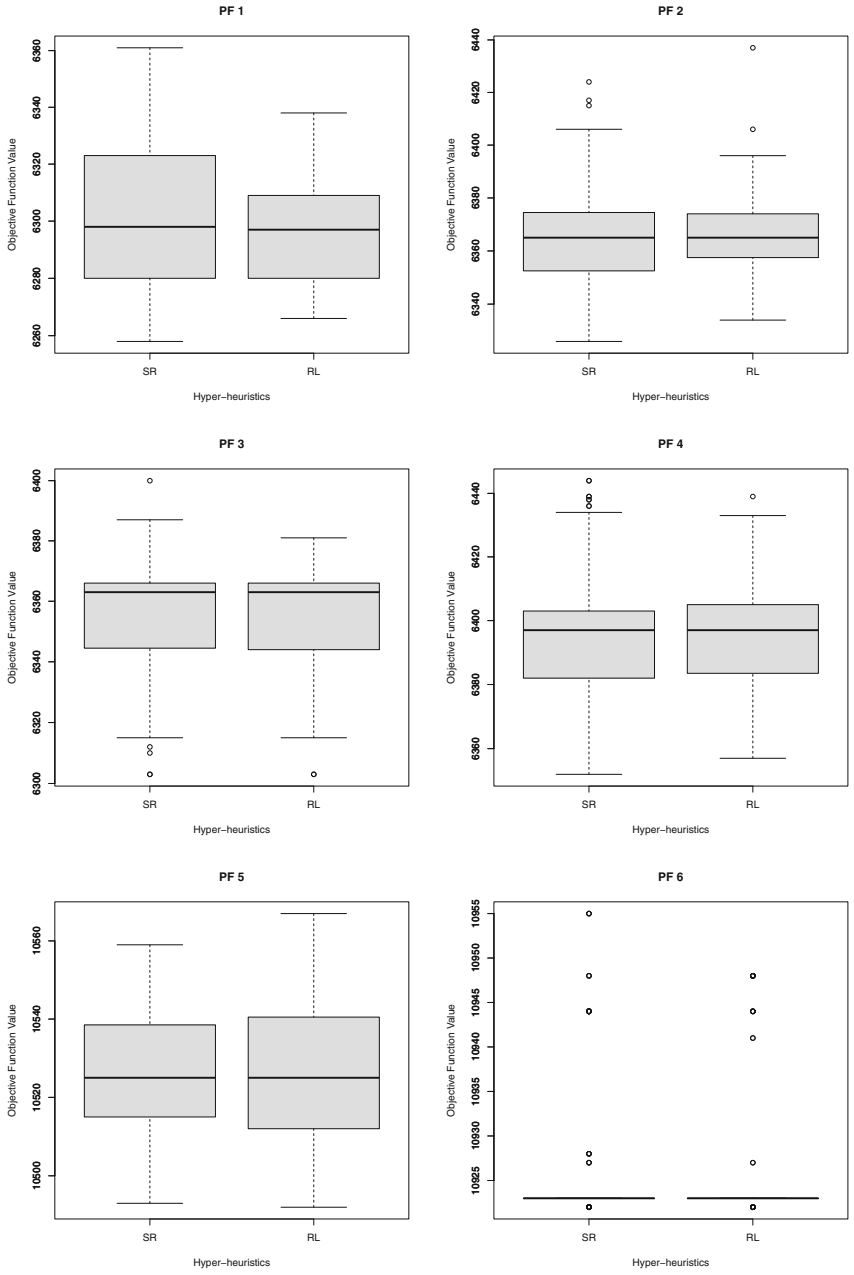


Fig. 2. The box plots show comparisons between the performance of simple random (SR), and reinforcement learning (RL) hyper-heuristics on six instances of the *permutation flow-shop* (PF) problem.

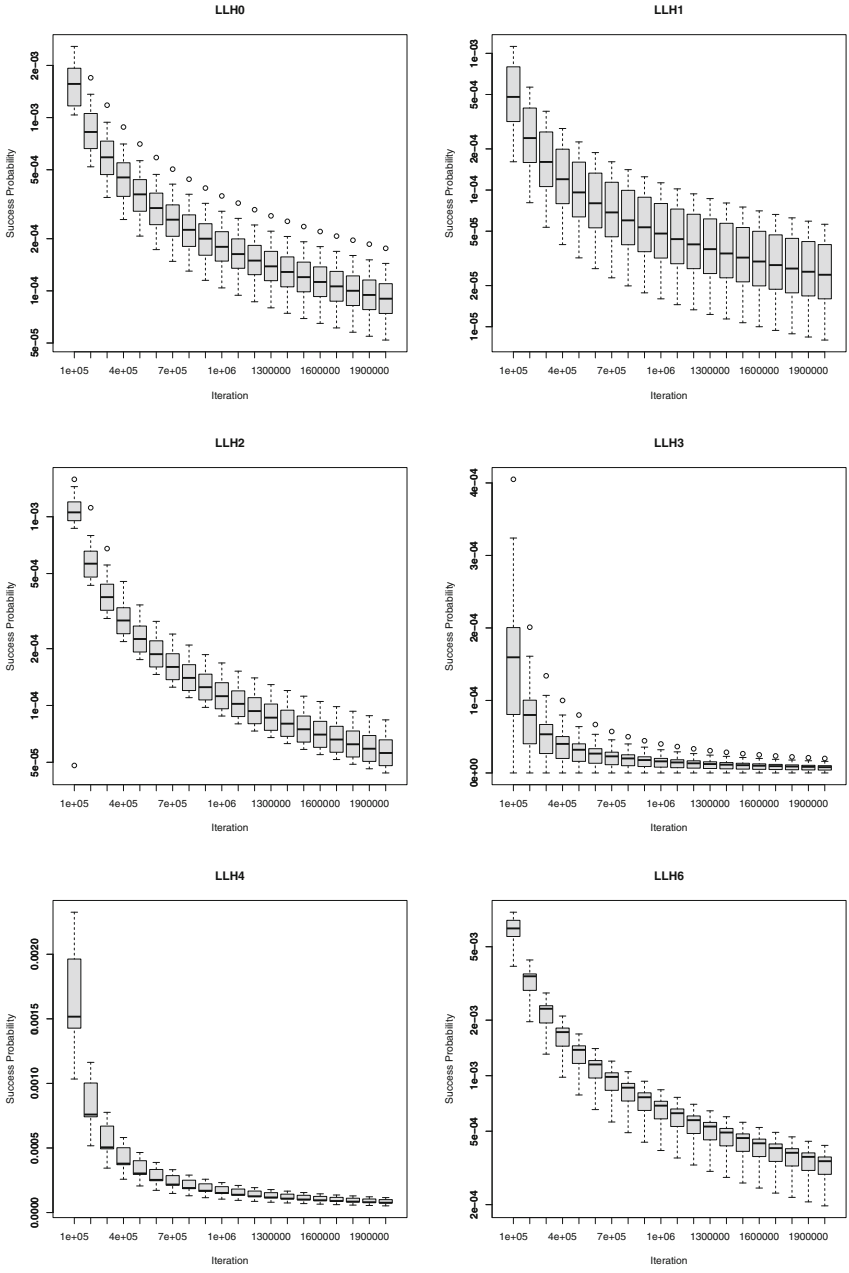


Fig. 3. The estimated success probabilities of low-level heuristics on the second instance (BP 2) of the bin-packing problem.

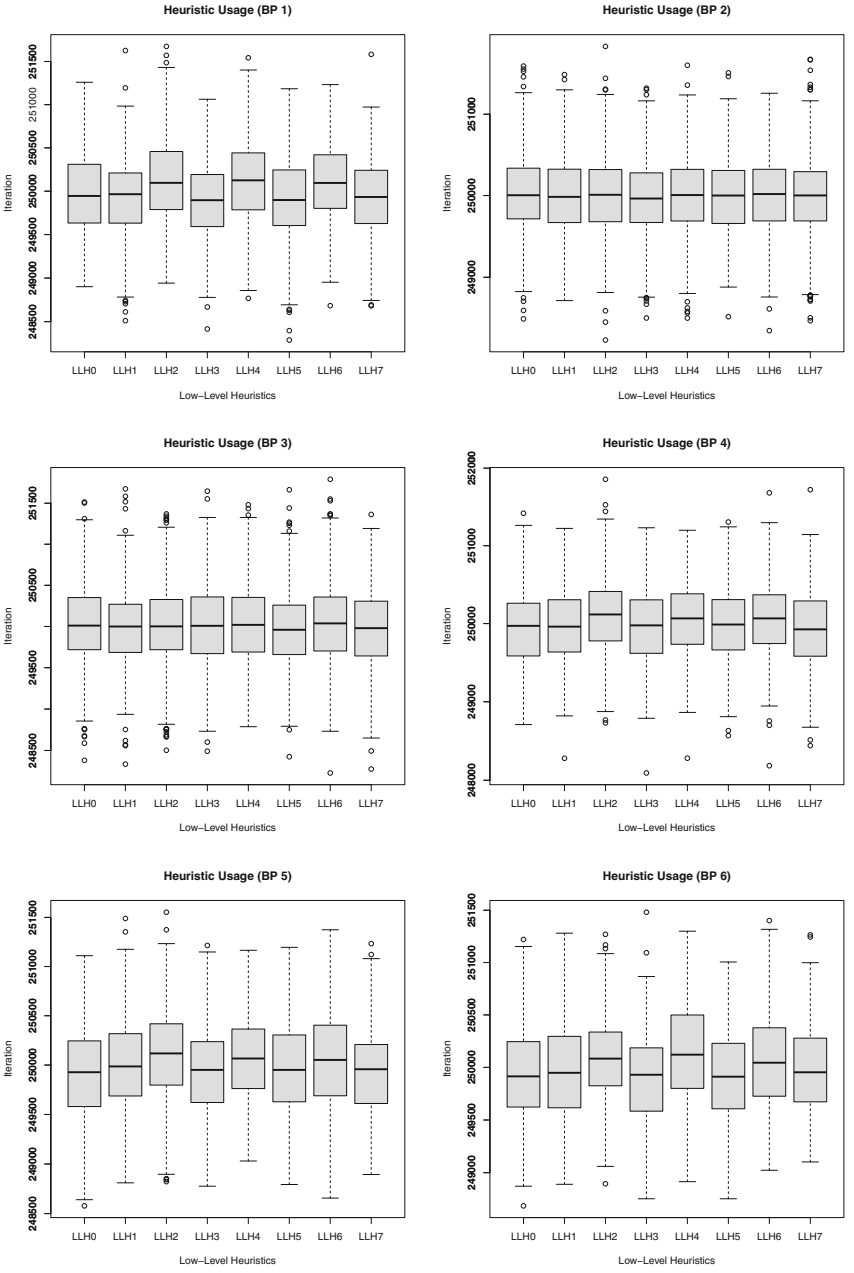


Fig. 4. Heuristics usage with the reinforcement learning hyper-heuristic on six instances of the *bin-packing* (BP) problem.

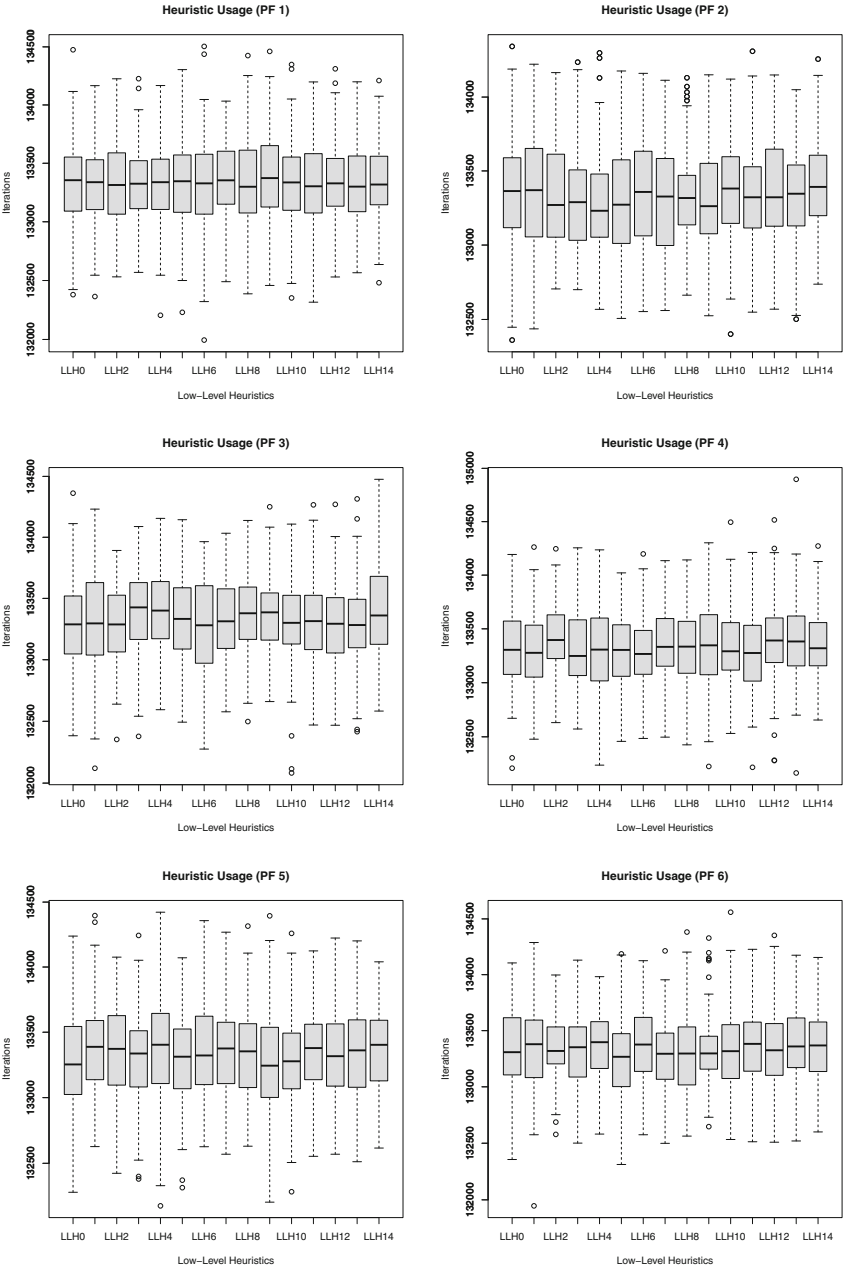


Fig. 5. Heuristics usage with the reinforcement learning hyper-heuristic on six instances of the *permutation flow-shop* (PF) problem.

scheduling, as shown in Fig. 2. This is consistent with the theoretical results in Sect. 4. We have shown that if the success probabilities of low-level heuristics are smaller than 1/2, then the additive reinforcement learning mechanism provides roughly the same probability of being chosen to each heuristic, similarly to the simple random. We track the success probabilities of low-level heuristics on the considered problems. A success probability of a heuristic can be estimated as a ratio of the number of times the heuristic has improved the candidate solution divided by the number of times it has been executed. The results show that the success probabilities of low-level heuristics are very small and varies between search states. Figure 3 shows the estimated success probabilities of 6 low-level heuristics out of 8 available heuristics on a randomly chosen instance of the *bin-packing* problem (the success probability of the remaining heuristics were zero during the entire running time). The estimated success probabilities of low-level heuristics in *permutation flow-shop* are also very small. Figure 4 shows that the total number of times each low-level heuristic was invoked by the *additive* reinforcement learning mechanism is almost the same in all instances of the *bin-packing* problem. The same observations can be seen in Fig. 5 where the reinforcement learning hyper-heuristic was applied to different instances of the *permutation flow-shop* problem. This shows that the *additive* reinforcement learning mechanism provides roughly the same number of invocations to all low-level heuristics (i.e. similar to simple random). This indicates that if the success probability of low-level heuristics are smaller than $(1/2)(1-\varepsilon)$ the *additive* reinforcement learning mechanisms almost choose low-level heuristics uniformly at random (as shown in Sect. 4).

6 Conclusion

The main goal of this paper was to determine the limitation of learning in *additive* reinforcement learning hyper-heuristics. We have shown that if the success probabilities of the low-level heuristics under consideration are less than 1/2, then these hyper-heuristics assign approximately the same probability of being chosen to each heuristic which is similar to using the simple random mechanism. Consistently with this, we have shown that both additive reinforcement learning and simple random hyper-heuristics have asymptotically the same expected runtime on the LeadingOnes problem. The experimental analysis shows that the estimated success probabilities of typical low-level heuristics are much smaller than 1/2. In addition, we found that several frequently used adaptation schemes either reduce the memory length of reinforcement learning mechanisms or lead them to pick systematically a specific heuristic. The study suggests that *additive* reinforcement learning mechanisms are not necessarily capable of distinguishing between the performances of the heuristics. In future work, we intend to identify learning mechanisms that adapt themselves to cope with the dynamic change in the success probabilities of low-level heuristics.

Acknowledgements. This research received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 618091 (SAGE).

References

1. Alanazi, F., Lehre, P.K.: Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In: IEEE Congress on Evolutionary Computation (CEC), pp. 2515–2523. IEEE (2014)
2. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics: Foundations and Recent Developments, vol. 1. World Scientific, River Edge (2011)
3. Bilgin, B., Özcan, E., Korkmaz, E.E.: An experimental study on hyper-heuristics and exam timetabling. In: Burke, E.K., Rudová, H. (eds.) PATAT 2006. LNCS, vol. 3867, pp. 394–412. Springer, Heidelberg (2007)
4. 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. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
5. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. J. Heuristics **9**(6), 451–470 (2003)
6. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to algorithms **5**(3), 55 (2001). MIT Press
7. Cowling, P.I., 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)
8. Dowsland, K.A., Soubeiga, E., Burke, E.: A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. Eur. J. Oper. Res. **179**(3), 759–774 (2007)
9. Feller, W.: An introduction to probability theory and its applications, vol. 1. Wiley, New York (1968)
10. Hajek, B.: Hitting-time and occupation-time bounds implied by drift analysis with applications. Adv. Appl. Prob. **14**(3), 502–525 (1982)
11. He, J., He, F., Dong, H.: Pure strategy or mixed strategy? In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 218–229. Springer, Heidelberg (2012)
12. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **127**(1), 57–85 (2001)
13. Jansen, T.: Analyzing Evolutionary Algorithms: The Computer Science Perspective. Springer, Heidelberg (2013)
14. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
15. Lehre, P.K., Özcan, E.: A runtime analysis of simple hyper-heuristics: to mix or not to mix operators. In: Proceedings of the Twelfth Workshop on Foundations of Genetic Algorithms XII, pp. 97–104. ACM (2013)
16. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In: Pardalos, P.M., Hearn, D.W. (eds.) Metaheuristics: Computer decision-making, pp. 523–544. Springer, Heidelberg (2004)
17. 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)

18. Özcan, E., Misir, M., Ochoa, G., Burke, E.K.: A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. Int. J. Appl. Metaheuristic Comput. (IJAMC) **1**(1), 39–59 (2010)
19. Remde, S., Cowling, P., Dahal, K., Colledge, N., Selensky, E.: An empirical study of hyperheuristics for managing very large sets of low level heuristics. J. Oper. Res. Soc. **63**(3), 392–405 (2012)

Modifying Colourings Between Time-Steps to Tackle Changes in Dynamic Random Graphs

Bradley Hardy(✉), Rhyd Lewis, and Jonathan Thompson

School of Mathematics, Cardiff University, Cardiff CF24 4AG, UK
{hardyB,lewisR9,thompsonJM1}@cardiff.ac.uk

Abstract. Many real world operational research problems can be formulated as graph colouring problems. Algorithms for this problem usually operate under the assumption that the size and constraints of a problem are fixed, allowing us to model the problem using a static graph. For many problems however, this is not the case and it would be more appropriate to model such problems using dynamic graphs. In this paper we will explore whether feasible colourings for one graph at time-step t can be modified into a colouring for a similar graph at time-step $t+1$ in some beneficial manner.

Keywords: Graph colouring · Dynamic graphs · Heuristics

1 Introduction

The graph colouring problem (GCP) aims to colour each vertex of a graph $G = (V, E)$ such that no adjacent vertices have the same colour and the number of colours used is minimised. The minimum number of colours required to colour a graph G is called the chromatic number of G, denoted by $\chi(G)$.

By considering the different aspects of a given problem instance and how they might relate to the components of a graph (vertices, edges and colours), one can reformulate many real world problems into a GCP. One example is frequency assignment [1] where each geographical site is represented by a vertex, an edge exists between two vertices if their respective sites are within a certain proximity of one another, and colours represent communication frequencies (e.g. radio frequencies). Other examples include exam timetabling [5,15], register allocation [3], designing seating plans [11] and grouping people in social networks [16].

Most GCP methods can only be applied to such problems under the assumption that the size and constraints of a problem are fixed (i. e. V and E are fixed in the associated graph $G = (V, E)$). However, in areas such as the frequency assignment problem [4] this is not always appropriate as sites can be added or removed from the communication network, or the location of sites can themselves move. The aim of this particular research, therefore, is to explore graph colouring on dynamic graphs. More specifically, we wish to look at methods which modify a feasible colouring for one graph into a colouring for a "similar" graph.

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 186–201, 2016.
DOI: 10.1007/978-3-319-30698-8_13

The rest of the paper will be structured as follows: Sect. 2 will formerly define dynamic graphs and their associated problems and Sect. 3 will then discuss the various search spaces for graph colouring problems. Section 4 will then outline a general approach and define the different modification methods used, Sect. 5 will contain the experimentation details, and in Sect. 6 we will present the results. Finally, Sect. 7 will summarise the findings of the experiments and discuss future work.

2 Dynamic Graph Colouring Problems

The importance of studying dynamic graphs and their associated problems has been highlighted by Harary and Gupta [7] who outlined many applications, especially in the area of computer science, and postulated that techniques applied to static graphs should be extended for dynamic graphs. However, there has been very little research regarding methods designed explicitly for dynamic graphs.

Two methods for finding colourings for dynamic graphs are given in [13,14]. The first of these proposes a genetic algorithm that uses the same population of colourings between time-steps for vertex dynamic graphs and the second proposes an agent-based approach for repairing colourings between time-steps for edge dynamic graphs. Both of these methods are only concerned with the quality of initial colourings, whereas this research will presume that optimisation can take place between time-steps.

We define a dynamic graph $\mathcal{G} = (G_0, G_1, \ldots, G_T)$ as a series of $T + 1$ static graphs where $G_t = (V_t, E_t) \in \mathcal{G}$ is the static graph defined for time-step $t \in \{0, 1, \ldots, T\}$. At every time-step, the objective in analogous to the static GCP. In terms of methodology, this means using heuristic methods to find a feasible k_t-colouring for each time-step t, where k_t is a good approximation of $\chi(G_t)$. Objectively, this is an attempt to minimise $\sum_{t=0}^{T} k_t$.

In this work we choose to split the concept of dynamic graphs into two cases: edge dynamic graphs and vertex dynamic graphs. In the edge dynamic graph colouring problem, changes can only occur on the edge set E_t; therefore $V_0 = V_1 = \cdots = V_T = V$ for all time-steps. For an edge dynamic graph $\mathcal{G}$, consider the graph $G_t = (V, E_t)$ for time-step t. To get to time-step $t+1$ we must define a set of deleted edges $E^-_{t+1} \subseteq E_t$ and a set of new edges $E^+_{t+1} \subseteq (\mathcal{E} \backslash E_t)$ where $\mathcal{E}$ is the set of all possible edges between vertices in V. The edge set for time-step $t + 1$ is then defined as $E_{t+1} = (E_t \backslash E^-_{t+1}) \cup E^+_{t+1}$.

In the vertex dynamic graph colouring problem, changes are applied to the vertex set V_t. This in turn affects the edge set E_t, as edges incident to deleted vertices will themselves need to be deleted. Similarly, new vertices will also require the addition of new edges unless the new vertex is intended to be isolated. For a vertex dynamic graph $\mathcal{G}$, consider the graph $G_t = (V_t, E_t)$ for time-step t. To get to time-step $t + 1$ we must define a set of deleted vertices $V^-_{t+1} \subseteq V_t$ and a set of new vertices V^+_{t+1}. Once these are defined, the set of deleted edges $E^-_{t+1} \subseteq E_t$ is defined to be the set of all edges incident to the deleted vertices (i. e. E^-_{t+1} contains all the edges $\{u, v\} \in E_t$ such that either $u \in V^-_{t+1}$ or $v \in V^-_{t+1}$).

The set of new edges E^+_{t+1} is a set of connecting edges from the set of new vertices to any of the vertices in V_{t+1} (i. e. E^+_{t+1} contains edges $\{u, v\} \in \mathcal{E}_{t+1}$ where $\mathcal{E}_{t+1}$ is the set of all possible edges between vertices in V_{t+1} and either $u \in V^+_{t+1}$ or $v \in V^+_{t+1}$). The vertex and edge sets for time-step $t + 1$ are then defined as $V_{t+1} = (V_t \backslash V^-_{t+1}) \cup V^+_{t+1}$ and $E_{t+1} = (E_t \backslash E^-_{t+1}) \cup E^+_{t+1}$ respectively.

In fact, edge dynamic graphs can be considered as a special case of vertex dynamic graphs where $|V^-_t| = |V^+_t| = |V_{t-1}|$ and $E^-_t = E_{t-1}$, $\forall t \in \{1, \ldots, T\}$. Another special case is on-line graph colouring, where exactly one vertex is added at each time-step (i. e. $V^-_t = \emptyset$ and $|V^+_t| = 1$, $\forall t \in \{1, \ldots, T\}$). On-line graph colouring has the additional constraint that, once coloured, a vertex cannot be transferred to a different colour class. Research concerning on-line graph colouring mainly consists of worst case behaviour analysis of algorithms [6,12].

3 Search Spaces of the GCP

In this paper we will approach dynamic graph colouring problems by adapting methods for the static problem. In general, the literature suggest three main search spaces for the static GCP: (i) *feasible colourings only*, where every vertex is coloured, there are no clashes (i. e. all adjacent vertices are coloured differently) and the number of colour classes is allowed to vary; (ii) *complete, improper colourings*, where every vertex is coloured but clashes are permitted; and (iii) *partial, proper colourings*, where no clashes occur but there may be "uncoloured" vertices.

The search space of feasible colourings only is rarely used in the literature as it is often difficult to determine which of two k-colourings is closer to becoming a colouring with $k - 1$ colour classes. One example of a heuristic method in this search space is a simulated annealing approach outlined in [9].

In the complete, improper search space a colouring $\mathcal{S} = \{S_1, \ldots, S_k\}$ is a partition of V into k disjoint subsets (i. e. $V = \bigcup_{i=1}^{k} S_i$ and $S_i \cap S_j = \emptyset$, $\forall i, j \in \{1, \ldots, k\}$ and $i \neq j$). S_i is called the ith colour class of the colouring $\mathcal{S}$ and the colouring function $c : V \to \{1, \ldots, k\}$ is defined such that $c(v) = i$ for all $v \in S_i$. One well-known algorithm that operates in this search space is TabuCol [8]. In this algorithm, to move from one colouring $\mathcal{S}$ to a neighbouring colouring $\mathcal{S}'$, a vertex v is transferred from its current colour class S_i to a different colour class S_j where $i \neq j$. Then $\mathcal{S}$ becomes $\mathcal{S}' = \{S'_1, \ldots, S'_k\}$ with $S'_i = S_i \backslash \{v\}$, $S'_j = S_j \cup \{v\}$ and $S'_l = S_l$, $\forall l \in \{1, \ldots, k\} \backslash \{i, j\}$. The vertex v to be moved can also be chosen exclusively from the set of currently clashing vertices (i. e. we can move $v \in S_i$ if and only if $\exists u \in S_i$ such that $u \neq v$ and $\{u, v\} \in E$). For a given solution $\mathcal{S}$, the associated cost function in this algorithm is given by

$$f(\mathcal{S}) = \sum_{i=1}^{k} |E_{(i)}| \tag{1}$$

where $E_{(i)}$ is the set of edges with both end points in S_i. This cost function is equivalent to the number clashes in the colouring. If $f(\mathcal{S}) = 0$ then the colouring $\mathcal{S}$ has no clashes and is therefore a feasible k-colouring.

In the partial, proper search space a colouring $\mathcal{S} = \{S_1, \ldots, S_k, S_{k+1}\}$ is defined by a partition of V into $k+1$ disjoint subsets. The first k subsets are independent sets (i. e. $E_{(i)} = \emptyset$, $\forall i \in \{1, \ldots, k\}$) and the remaining vertices $v \in V \backslash (\bigcup_{i=1}^{k} S_i)$ are placed in the additional subset S_{k+1} of "uncoloured" vertices, in which clashes are also permitted.

PartialCol [2] (a modification of TabuCol) is an example of an algorithm that operates in this search space. In this algorithm, to move from one colouring $\mathcal{S}$ to a neighbouring colouring $\mathcal{S}'$, we transfer an uncoloured vertex $v \in S_{k+1}$ to a colour class S_i where $i \leq k$ and move the set of vertices adjacent to v, $U_i \subseteq S_i$, to S_{k+1}. Then $\mathcal{S}$ becomes $\mathcal{S}' = \{S'_1, \ldots, S'_k, S'_{k+1}\}$ with $S'_i = (S_i \backslash U_i) \cup \{v\}$, $S'_{k+1} = (S_{k+1} \cup U_i) \backslash \{v\}$ and $S'_l = S_l$, $\forall l \in \{1, \ldots, k\} \backslash \{i\}$.

For a given solution $\mathcal{S}$, the associated cost function in this algorithm is given by

$$f(\mathcal{S}) = |S_{k+1}| \tag{2}$$

which is equivalent to the number of uncoloured vertices. An alternative cost function is

$$f(\mathcal{S}) = \sum_{v \in S_{k+1}} \deg(v) \tag{3}$$

where $\deg(v)$ is the degree of vertex v. If the vertices in S_{k+1} have low degrees then, in theory, they will be easier to move into colour classes without causing clashes. For both of these cost functions, if $f(\mathcal{S}) = 0$ then there are no uncoloured vertices and $\mathcal{S}$ is therefore a feasible k-colouring.

4 Methods

Our approach for solving a dynamic graph $\mathcal{G} = \{G_0, G_1, \ldots, G_T\}$ will follow the process outlined in Algorithm 1. Notice that for G_0 a method for the static GCP needs to be applied.

Algorithm 1. Generic DGCP Algorithm

Input: a dynamic graph $\mathcal{G} = (G_0, G_1, \ldots, G_T)$
Output: a set $\mathbf{S} = \{\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_T\}$ where $\mathcal{S}_t$ is a feasible colourings for $G_t \in \mathcal{G}$
1: $\mathcal{S}_0 \leftarrow$ Static GCP Algorithm (G_0)
2: **for** $t = 1$ **to** T **do**
3: $\quad \mathcal{S}_t \leftarrow$ Dynamic GCP Time-step Algorithm $(G_t, \mathcal{S}_{t-1})$ (i. e. Algorithm 2)
4: **return** $\mathbf{S} = \{\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_T\}$

For each time-step t, suppose a feasible colouring $\mathcal{S}_t$ for G_t has been found; that is, a colouring where all vertices are coloured and no clashes occur. This colouring might then be saved and possibly modified in some way to be used as a colouring for G_{t+1}. Using this modified colouring with $k \geq |\mathcal{S}_t|$ colour classes as a starting point, we then wish to find a feasible k-colouring for G_{t+1}. If we succeed, then we search for a feasible colouring with one fewer colour class and

so on until some stopping criteria (e.g. a time or iteration limit) is reached. It may of course be impossible to find a feasible k-colouring for G_{t+1}. In order to accommodate this eventuality, if some timing criteria is met and a feasible colouring (of any size) has not be found, then we increase k by 1, we allow the target number of colour classes to be increased indefinitely until a feasible colouring is found or the algorithm's stopping criteria is met. This process is outlined in Algorithm 2.

The focus of this particular piece of research is to explore the different methods for modifying a feasible colouring achieved in time-step t into an initial colouring for time-step $t+1$ (i. e. line 2 of Algorithm 2). The essential question to be answered is: can a feasible colouring for one graph G_t be used in some advantageous way to find a feasible colouring for a similar graph G_{t+1}?

Algorithm 2. Generic DGCP Time-step Algorithm

Input: a graph G_{t+1} and a feasible colouring $\mathcal{S}_t$ for G_t
Output: a feasible colouring $\mathcal{S}_{t+1}$ for G_{t+1}
1: $\mathcal{S}_{best} \leftarrow \emptyset$
2: $\mathcal{S}_{t+1} \leftarrow \mathcal{S}_t$ modified in some way (see Sects. 4.1 and 4.2)
3: $k \leftarrow |\mathcal{S}_{t+1}|$
4: **while not** stopping criterion **do**
5: attempt to make $\mathcal{S}_{t+1}$ a feasible k-colouring for G_{t+1}
6: **if** $\mathcal{S}_{t+1}$ is a feasible k-colouring for G_{t+1} **then**
7: $\mathcal{S}_{best} \leftarrow \mathcal{S}_{t+1}$
8: $k \leftarrow k - 1$
9: **if** $\mathcal{S}_{best} = \emptyset$ **and** a computation limit is reached **then**
10: $k \leftarrow k + 1$
11: $\mathcal{S}_{t+1} \leftarrow \mathcal{S}_{best}$
12: **return** $\mathcal{S}_{t+1}$

4.1 Modification for Edge Dynamic Graphs

For all of the following methods, the final feasible colouring $\mathcal{S}_t$ for G_t can be considered as a complete, improper colouring for G_{t+1} with $k = |\mathcal{S}_t|$ colour classes. We can do this because every vertex $v \in V$ will be coloured but the new edges E^+_{t+1} are likely to cause clashes. With this knowledge we can then apply one of the following modification methods.

(1) *Calculate the number of clashes*: Calculate the initial number of clashes and then pass $\mathcal{S}_t$ directly to the tabu search operator which will attempt to find a feasible k-colouring for G_{t+1}.

(2) *Uncolour clashing vertices*: By identifying pairs of clashing vertices in $\mathcal{S}_t$ and transferring one of the vertices in each of these pairs to a set of uncoloured vertices, one produces a partial, proper colouring $\tilde{\mathcal{S}}_{t+1}$ for G_{t+1}. $\tilde{\mathcal{S}}_{t+1}$ along with the set of uncoloured vertices can now be passed to the tabu search operator which will attempt to find a feasible k-colouring for G_{t+1}.

(3) *Solve clashing vertices*: In a similar manner to Method (2), clashing vertices are "uncoloured" to produce a partial, proper colouring $\tilde{\mathcal{S}}_{t+1}$ for G_{t+1}. An attempt is then made to re-insert each of these uncoloured vertices into a colour class in $\tilde{\mathcal{S}}_{t+1}$ such that no clashes are incurred. The remaining uncoloured vertices and any appropriate edges are then considered as a residual graph G'_{t+1} of G_{t+1}. This residual graph is passed to the constructive operator (specifically, the recursive largest first (RLF) algorithm [10]) which produces a feasible k'-colouring for G'_{t+1}. The feasible colouring for G'_{t+1} is then combined with $\tilde{\mathcal{S}}_{t+1}$ to produce a feasible colouring for G_{t+1} with $k + k'$ colour classes. The tabu search operator will then attempt to find a feasible $(k + k' - 1)$-colouring for G_{t+1}.

4.2 Modification for Vertex Dynamic Graphs

The final feasible colouring $\mathcal{S}_t$ achieved for G_t will be neither a complete, improper colouring or a partial, proper colouring for G_{t+1} as it will include the deleted vertices V^-_{t+1} and won't include the new vertices V^+_{t+1}. For each of the following methods, every deleted vertex $v \in V^-_{t+1}$ must first be removed from $\mathcal{S}_t$ in order to produce a partial, proper colouring $\tilde{\mathcal{S}}_{t+1}$ for G_{t+1} with $k = |\mathcal{S}_t|$ colour classes. We can then apply one of the following modification methods.

(4) *Randomly assign new vertices*: Each new vertex $v \in V^+_{t+1}$ is randomly assigned to a colour class in $\tilde{\mathcal{S}}_{t+1}$ to produce a complete, improper colouring for G_{t+1}. This can then be passed to the tabu search operator which will attempt to find a feasible k-colouring for G_{t+1}.

(5) *Uncolour new vertices*: Unlike Method (4), the new vertices V^+_{t+1} are not assigned to colour classes in $\tilde{\mathcal{S}}_{t+1}$. Instead the new vertices V^+_{t+1} are considered as a set of uncoloured vertices. Along with $\tilde{\mathcal{S}}_{t+1}$, this set of uncoloured vertices is passed to the tabu search operator which attempts to find a feasible k-colouring for G_{t+1}.

(6) *Solve new vertices*: An attempt is made to insert each of the new vertex $v \in V^+_{t+1}$ into an a colour class in $\tilde{\mathcal{S}}_{t+1}$ such that no clashes are incurred. The remaining new vertices and any appropriate edges are then considered as a residual graph G'_{t+1} of G_{t+1}. This residual graph is passed to the constructive operator (again, RLF) which produces a feasible k'-colouring for G'_{t+1}. The feasible colouring for G'_{t+1} is then combined with $\tilde{\mathcal{S}}_{t+1}$ to produce a feasible colouring for G_{t+1} with $k + k'$ colour classes. The tabu search operator will then attempt to find a feasible $(k + k' - 1)$-colouring for G_{t+1}.

5 Experimentation Details

In our experiments we considered dynamic random graphs. For each dynamic random graph we specify an initial number of vertices n, a desired density d, a change probability p and a number of time-steps T. To construct a sequence of graphs $\mathcal{G}$ we use the following methods.

For an edge dynamic graph consider the graph $G_t = (V, E_t)$. To construct G_{t+1}, every edge $\{u, v\} \in E_t$ is copied to the set of deleted edges E^-_{t+1} with

probability p and every currently non-existent edge $\{u, v\} \in \mathcal{E} \backslash E_t$ is copied to the set of new edges E^+_{t+1} with probability $\frac{dp}{1-d}$.

For a vertex dynamic graph, consider the graph $G_t = (V_t, E_t)$. To construct G_{t+1}, every vertex $v \in V_t$ is copied to the set of deleted vertices V^-_{t+1} with probability p and the set of new vertices is constructed such that $|V^+_{t+1}|$ is an integer between $np(1-p)$ and $np(1+p)$. Every edge $\{u, v\} \in \mathcal{E}_{t+1}$ with $u \in V_{t+1}$, $v \in V^+_{t+1}$ and $u \neq v$ is then added to the set of new edges E^+_{t+1} with probability d.

For both the edge and vertex dynamic graphs, the following parameters were used: $n = 500$, $d \in \{0.1, 0.5, 0.9\}$, $p \in \{0.005, 0.01, \ldots, 0.05\}$ and $T = 10$, and for each combination of these parameters, 20 graphs were produced. The RLF algorithm [10] was applied to obtain an initial colouring for G_0. Note that all results corresponding to these initial graphs are ignored; however, the colourings they produced were used in the modification methods for G_1.

In our case, each time-step was given a time limit of 10 s[1] (i. e. line 4 of Algorithm 2). If this time limit had been set much longer, say hours, then the advantage of modifying colourings between time-steps would obviously diminish.

TabuCol [8] and PartialCol [2] were used to find feasible colourings in the complete, improper search space and partial, proper search space respectively (i. e. line 5 of Algorithm 2). These algorithms use the neighbourhood moves outlined in Sect. 3 and, upon performing a move, the inverse moves are made "tabu" for $0.6 \times f(\mathcal{S}') + r$ iterations, where f is the cost function given in Eqs. (1) and (2) respectively, $\mathcal{S}'$ is the resultant colouring after the neighbourhood move, and r is a random integer from the set $\{0, 1, \ldots, 9\}$. This tabu tenure has been used in both [8] and [2].

During execution, the target number of colour classes is adjusted in the following way. Let k be the target number of colour classes, initially defined by the modification method being implemented. If a feasible k-colouring cannot be obtained within half of the allotted time limit then k is increased by 1. If a feasible k-colouring cannot then be obtained within half of this remaining time limit then k is again increased by 1, and so on (i. e. lines 9 and 10 of Algorithm 2). For example, say the target number of colour classes for G_t is initially set as $k = 23$, if a feasible 23-colouring cannot be found within 5 s then the tabu search operator attempts to find a feasible 24-colouring for G_t, if this cannot be found within a further 2.5 s then the tabu search attempts to find a feasible 25-colouring for G_t, and so on.

For a base-line comparison, the following control method was also implemented:

(0) *Reset*: The static graph $G_t \in \mathcal{G}$ for each time-step $t \in \{1, \ldots, T\}$ is considered without any information about colourings achieved in the previous time-steps. As with G_0, the RLF algorithm is applied to obtain an initial colouring for G_t (i. e., RLF replaces line 2 of Algorithm 2). Tabu search is then applied iteratively in an attempt to find colourings with fewer colour classes. The number

[1] All algorithms were programmed in C++ and executed on a 3.3 GHZ Windows 7 PC with an Intel Core i3-2120 processor and 8 GB RAM.

of colour classes in the final, feasible colouring achieved and the time required to obtain this colouring is then recorded.

Note that Methods (1) and (4) operate exclusively in the complete, improper search space, Methods (2) and (5) operate exclusively in the partial, proper search space, and Methods (0), (3) and (6) can operate in either search space as required. Because of this, only comparisons between methods designed for the same problem and operating in the same search space are compared. For example, for the edge dynamic GCP operating in the complete, improper search space only Methods (0), (1) and (3) are compared against one another.

In all of our results, unless otherwise stated, all statistical tests are Wilcoxon signed rank tests with significance level $\alpha = 0.05$.

6 Results

6.1 Initial Colourings for the Edge Dynamic GCP

Let us first consider the initial feasible colourings produced for the edge dynamic GCP. For all densities d and change probabilities p, Methods (1) and (2) were found to produce initial, feasible colourings with significantly fewer colour classes than both Methods (0) and (3). This is clearly illustrated in Fig. 1.

We have observed a significant increase in the time required by Methods (1) and (2) to achieve their initial, feasible colourings compared to Methods (0) and (3) for all values of d and p, as seen in Table 1. A main contributing factor to this may be found in the nature of the different methods: Methods (0) and (3) both start from feasible colourings whereas Methods (1) and (2) do not and therefore require more time to move to a feasible region of the search space. For similar reasons, as p increases so too does the time required by Methods (1) and (2) to achieve an initial, feasible colouring.

For $d = 0.1$ with $p = 0.005$, $d = 0.5$ with $p \leq 0.02$, and $d = 0.9$ with $p \leq 0.01$ Method (3) was found to produce initial, feasible colourings with significantly fewer colour classes than Method (0). However, for higher settings of p, specifically for $d = 0.1$ with $p \geq 0.01$, $d = 0.5$ with $p \geq 0.03$, and $d = 0.9$ with $p \geq 0.015$, the opposite holds. This is again clearly illustrated in Fig. 1. Hence we can conclude that for these high levels of p, modifying feasible colourings for G_t is of no benefit when attempting to achieve initial, feasible colourings for G_{t+1}.

Considering computational effort, we have found that the time required by Method (3) to achieve initial, feasible colourings is significantly less compared to Method (0) for $d \in \{0.5, 0.9\}$ with all values of p. Both Methods (0) and (3) employ RLF; however, Method (0) applies it to the whole graph $G_t = (V, E_t)$ at each time-step t as opposed to Method (3) which only applies it to a residual graph $G'_t = (V', E'_t)$ of G_t where $V' \subseteq V$ (which implies $|V'| \leq |V|$). We therefore see that applying Method (3) with low levels of p is advantageous with regards to both the number of colour classes in initial, feasible colourings and the time required to obtain them.

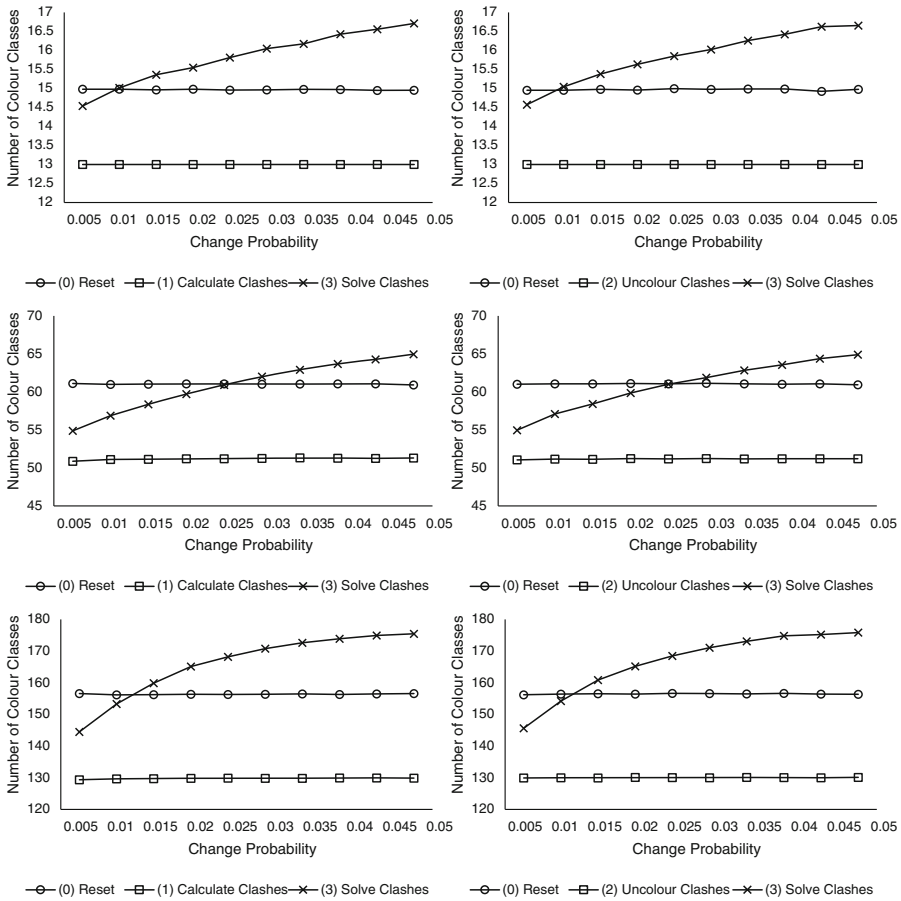


Fig. 1. Mean initial, feasible colourings for the edge dynamic GCP. Graphs on the left represents results from trials in the complete, improper search space and those on the right for trials in the partial, proper search space. From top to bottom, rows represent $d = 0.1, 0.5$, and 0.9 respectively.

6.2 Initial Colourings for the Vertex Dynamic GCP

Let us now consider initial colourings for the vertex dynamic GCP. It is first worth mentioning that a small change to the edge set of a graph will affect more vertices than a comparable change to its vertex set. It is therefore not surprising that the following results are similar to those presented in Sect. 6.1 but for higher values of p.

Comparable to Methods (1) and (2) for the edge dynamic problem, the initial, feasible colourings achieved by Methods (4) and (5) have significantly fewer colour classes than Methods (0) and (6) but require significantly more time to obtain them. The time required by Methods (4) and (5) also has a positive relationship with the change probability p. These observations can be seen in

Table 1. Median time (in seconds) required to obtain an initial, feasible colouring for the edge dynamic GCP (a 0^* entry implies that the recorded time is less than 10^{-3} s).

		$p(\%)$									
d	Method	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
0.1	(0)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*
	(1)	0^*	0.015	0.016	0.031	0.031	0.031	0.031	0.047	0.047	0.047
	(2)	0^*	0^*	0.015	0.015	0.015	0.015	0.015	0.016	0.016	0.016
	(3)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*
0.5	(0)	0.016	0.016	0.031	0.031	0.031	0.016	0.016	0.016	0.016	0.016
	(1)	1.692	2.246	2.777	2.948	3.182	3.268	3.363	3.791	4.181	3.713
	(2)	1.545	1.872	2.083	2.996	2.325	2.590	2.824	2.519	2.730	2.972
	(3)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*
0.9	(0)	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031
	(1)	5.008	5.125	5.335	5.140	5.288	5.421	5.366	5.171	5.327	5.304
	(2)	4.376	4.235	5.016	5.070	5.047	5.031	5.008	4.789	5.038	5.023
	(3)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*

Fig. 2 and Table 2. The reasons for this behaviour are the same as those given for Methods (1) and (2) in Sect. 6.1.

Again, as with Method (3) for the edge dynamic problem, Method (6) produces initial, feasible colourings with both significantly fewer and significantly more colour classes than Method (0) depending on the change probability p. However, Method (6) only produces initial, feasible colourings with significantly more colour classes for $d = 0.1$ with $p \geq 0.035$. In fact, for $d = 0.1$ with $p \leq 0.02$, and $d \in \{0.5, 0.9\}$ with all values of p, Method (6) achieves initial, feasible colourings with significantly fewer colour classes. This is clearly illustrated in Fig. 2.

As with Method (3), Method (6) requires significantly less time than Method (0) in all instances except for $d = 0.1$ with $p \leq 0.03$ (as seen in Table 2). This is again likely because Method (6) applies RLF to a smaller graph G'_t with $|V'_t| = |V_t^+| \approx np$ as opposed to applying it to G_t with $|V_t| \approx n$.

6.3 Final Colourings for the Edge Dynamic GCP

Next let us consider final colourings for the edge dynamic GCP. The Friedman test with $\alpha = 0.05$ shows that for $d = 0.1$ there is no significant difference between the number of colour classes in the final, feasible colourings achieved when applying Methods (0), (1), (2) and (3). However, Methods (1) and (2) both achieve final, feasible colourings with significantly more colour classes than those achieved by Method (0) for $d = 0.9$ with $p \geq 0.01$ and $p \geq 0.035$ respectively. Methods (1) and (2) also achieve final, feasible colourings with significantly more colour classes than those achieved by Method (3) for $d = 0.9$ with some values of p. This observation

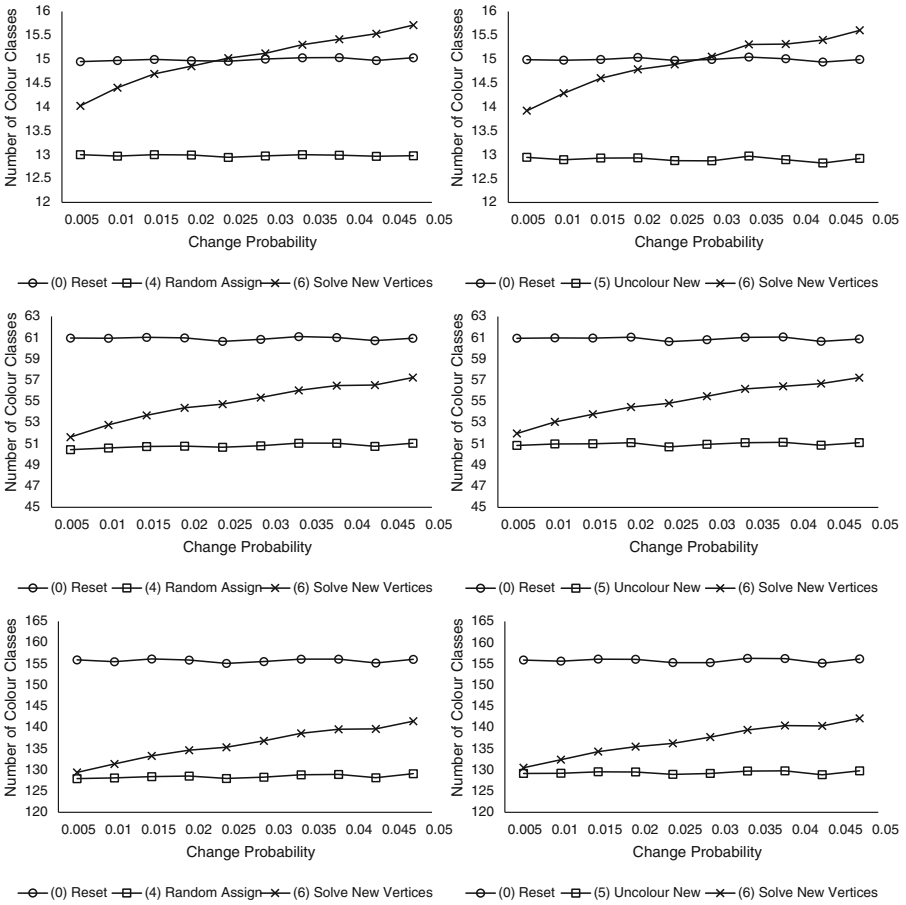


Fig. 2. Mean initial, feasible colourings for the vertex dynamic GCP. Graphs on the left represents results from trials in the complete, improper search space and those on the right for trials in the partial, proper search space. From top to bottom, rows represent $d = 0.1, 0.5$, and 0.9 respectively.

is likely due to the relatively large amount of time required by Methods (1) and (2) to find an initial, feasible colouring compared to Methods (0) and (3) (see Sect. 6.1 and Table 1). This "wasted" time then translates to time not being allocated to finding feasible colourings with fewer colour classes.

For $d = 0.5$ and some values of p, Method (3) was found to achieve final, feasible colourings with significantly fewer colour classes than Method (0). However, for $d = 0.9$ with $p \geq 0.04$ the opposite holds which is unsurprising as Method (3) produces initial, feasible colourings with significantly more colour classes under these parameter settings.

Table 2. Median time (in seconds) required to obtain an initial, feasible colouring for the vertex dynamic GCP (a 0^* entry implies that the recorded time is less than 10^{-3} s).

		$p(\%)$									
d	Method	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
0.1	(0)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0.015	0.015
	(4)	0^*	0^*	0.015	0.015	0.016	0.031	0.031	0.031	0.031	0.046
	(5)	0^*	0^*	0.015	0.015	0.015	0.016	0.015	0.016	0.016	0.016
	(6)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*
0.5	(0)	0.016	0.031	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016
	(4)	0.320	1.131	1.240	1.724	1.482	1.724	1.935	2.411	2.114	2.785
	(5)	0.663	0.983	1.537	1.529	1.630	1.537	1.973	1.794	1.841	2.340
	(6)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*
0.9	(0)	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031
	(4)	1.069	1.997	2.941	3.830	3.565	4.189	4.820	4.938	4.852	5.007
	(5)	1.163	1.731	2.644	3.222	2.387	3.416	3.339	3.424	2.816	3.346
	(6)	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*

The following time comparisons correspond only to trials where the number of colour classes in the final, feasible colourings achieved by the compared methods were equal to one another. This will also be the case in Sect. 6.4.

Method (1) was found to reach final, feasible colourings significantly faster than Method (0) for $d = 0.1$ with $p \leq 0.035$, and $d = 0.5$ with $p \leq 0.01$ as seen in Table 3. Similarly, Method (2) also achieves final, feasible colourings in significantly less time than Method (0) for $d = 0.1$ with all values of p, and $d = 0.5$ with $p = 0.005$. Both of these methods were also able to reach final, feasible colourings significantly faster than Method (3) for $d = 0.1$ with some values of p. These observations are likely due to the fact that the initial, feasible colourings achieved by Methods (1) and (2) are also the final, feasible colourings achieved for $d \in \{0.1, 0.5\}$ with low values of p.

On the other hand, Method (1) was found to require significantly more time than Method (0) to achieve final, feasible colourings for $d = 0.5$ with $p \geq 0.035$, and $d = 0.9$ with all values of p. The same was also found for Method (2) for $d = 0.9$ with most values of p. In a similar fashion, these two methods require significantly more time to achieve final, feasible colourings than Method (3) for $d \in \{0.5, 09\}$ with most values of p. This is probably due to the same arguments presented with regards to the number of colour classes in the final, feasible colourings achieved by these methods for $d = 0.9$.

Unlike Methods (1) and (2), Method (3) was not found to require significantly more time than Method (0) for any parameter settings. On the contrary, for $d = 0.1$ with $p \leq 0.035$, and $d = 0.5$ with $p \leq 0.02$, Method (3) requires significantly less time to achieve final, feasible colourings. It should be highlighted

Table 3. Median time (in seconds) required to obtain final, feasible colourings with the same numbers of colour classes for the edge dynamic GCP (a 0^* entry implies that the recorded time is less than 10^{-3} s).

			$p(\%)$									
d	S.S.	Method	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
0.1	C.I.	(0)	0.047	0.046	0.047	0.046	0.047	0.046	0.047	0.046	0.047	0.047
		(1)	0^*	0.015	0.016	0.031	0.031	0.031	0.031	0.047	0.047	0.047
		(3)	0.015	0.016	0.031	0.031	0.046	0.031	0.047	0.047	0.047	0.047
	P.P.	(0)	0.016	0.016	0.031	0.016	0.031	0.031	0.031	0.031	0.031	0.031
		(2)	0^*	0^*	0.015	0.015	0.015	0.015	0.015	0.016	0.016	0.016
		(3)	0^*	0.015	0.015	0.015	0.016	0.016	0.016	0.031	0.031	0.031
0.5	C.I.	(0)	3.478	2.996	3.034	3.128	2.442	2.855	2.528	3.136	2.941	2.754
		(1)	1.653	2.371	3.190	3.097	3.424	3.417	3.869	4.259	4.321	4.275
		(3)	1.077	1.794	2.130	2.683	2.239	2.652	2.754	2.465	3.284	2.762
	P.P.	(0)	2.933	2.910	3.081	2.870	2.278	2.730	2.636	2.559	2.309	2.676
		(2)	1.872	1.888	2.356	3.783	2.356	2.722	3.058	2.847	2.746	3.331
		(3)	1.435	2.160	2.060	2.356	2.246	2.699	2.169	2.442	2.168	2.598
0.9	C.I.	(0)	5.492	5.476	5.008	4.836	5.569	5.912	4.851	5.694	4.430	4.602
		(1)	6.225	7.122	7.691	7.074	7.964	7.550	7.535	8.455	7.176	7.488
		(3)	4.181	4.906	4.415	4.353	5.694	5.195	4.649	5.234	5.039	4.882
	P.P.	(0)	4.181	5.242	5.179	4.166	5.273	4.914	3.681	4.602	4.212	4.633
		(2)	5.141	5.616	5.975	6.365	6.365	5.741	6.038	5.506	5.452	5.866
		(3)	3.877	4.649	5.070	4.275	4.352	4.025	4.196	4.618	4.688	4.977

that these are similar parameter settings for which Method (3) is able to produce initial, feasible colourings with significantly fewer colour classes than Method (0).

6.4 Final Colourings for the Vertex Dynamic GCP

Finally, let us consider final colourings for the vertex dynamic GCP. As mentioned in Sect. 6.2, a small change to the edge set will usually affect more vertices than a comparable change to its vertex set.

Method (4) was found to achieve final, feasible colourings with significantly fewer colour classes than Method (0) for $d = 0.5$ with most values of p, and $d = 0.9$ with $p \leq 0.03$. Similarly, Method (5) was also found to achieve final, feasible colourings with significantly fewer colour classes than Method (0) for $d \in \{0.5, 0.9\}$ with some values of p. On the other hand, Method (4) achieves final, feasible colourings with significantly more colour classes than Method (6) for $d = 0.9$ with $p \geq 0.025$. Although Methods (4) and (5) require significantly more time to produce initial, feasible colourings (see Sect. 6.2 and Table 1) it is likely that Methods (0) and (6) still require more time to reach a feasible colouring with equivalent numbers of colour classes for low levels of p. This would imply that Methods (4) and (5) attempt to find feasible colourings with

Table 4. Median time (in seconds) required to obtain final, feasible colourings with the same numbers of colour classes for the vertex dynamic GCP (a 0^* entry implies that the recorded time is less than 10^{-3} s).

			$p(\%)$									
d	S.S.	Method	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
0.1	C.I.	(0)	0.046	0.046	0.047	0.047	0.047	0.047	0.047	0.047	0.047	0.062
		(4)	0^*	0^*	0.015	0.015	0.016	0.031	0.031	0.031	0.031	0.046
		(6)	0^*	0^*	0.015	0.016	0.031	0.031	0.031	0.046	0.031	0.047
	P.P.	(0)	0.016	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031
		(5)	0^*	0^*	0.015	0.015	0.016	0.016	0.016	0.031	0.031	0.031
		(6)	0^*	0.015	0.015	0.015	0.016	0.016	0.016	0.031	0.031	0.031
0.5	C.I.	(0)	5.141	3.682	4.321	4.610	4.212	3.619	3.884	3.713	3.666	3.612
		(4)	0.515	1.224	1.996	1.747	2.738	2.699	3.713	2.551	4.103	4.470
		(6)	0.328	1.084	1.045	1.303	2.028	1.740	1.981	2.020	2.013	2.247
	P.P.	(0)	2.652	3.073	3.276	3.151	2.980	2.504	2.964	2.457	2.933	2.855
		(5)	1.505	1.264	2.574	2.621	2.341	3.066	2.566	2.551	3.167	2.980
		(6)	0.203	1.068	1.092	1.529	1.544	1.420	1.381	2.192	2.387	2.293
0.9	C.I.	(0)	5.702	6.069	6.318	6.146	6.053	6.021	6.209	5.843	5.881	6.186
		(4)	1.428	5.007	5.007	4.399	5.460	5.148	5.507	6.069	6.381	6.459
		(6)	1.786	2.090	3.019	3.307	4.033	3.681	3.667	4.227	3.791	4.142
	P.P.	(0)	4.867	5.281	4.680	5.492	5.585	4.267	4.181	4.665	4.602	4.462
		(5)	2.964	3.362	4.665	5.194	4.446	4.196	6.255	5.585	5.054	5.281
		(6)	1.373	2.013	2.566	1.981	3.261	3.330	3.416	2.745	3.884	4.189

fewer colour classes earlier than Methods (0) and (6). Further analysis should be conducted in order to investigate the validity of this proposition.

Unlike Method (3) for the edge dynamic problem, Method (6) was only found to reach final, feasible colourings with the same or significantly fewer colour classes than Method (0). Both Methods (0) and (6) start each time-step from a feasible colouring; however, Method (6) achieves initial colouring with significantly fewer colour classes than Method (0) for most combinations of d and p (see Sect. 6.2 and Fig. 2). Method (6) will therefore attempt to find feasible colourings with fewer colour classes earlier than Method (0).

It was found that Methods (4) and (5) achieve final, feasible colourings in significantly less time than Method (0) for $d = 0.1$ with all values of p, and $d \in \{0.5, 0.9\}$ with $p \leq 0.01$. Additionally, Method (4) was found to achieve final, feasible colourings in significantly less time for $d = 0.5$ with $p \leq 0.04$, and $d = 0.9$ with $p \leq 0.025$ also. This can be seen in Table 4. The reason for these observations is likely to be the same as that given with regards to the number of colour classes in the final, feasible colourings achieved with low levels of p.

On the contrary, Methods (4) and (5) require significantly more time to achieve final, feasible colourings than Method (6) for $d \in \{0.5, 0.9\}$ with most values of p. In comparison to Method (0), Method (6) starts from a feasible colouring with significantly fewer colour classes for $d \in \{0.5, 0.9\}$ with all values

of p (see Sect. 6.2 and Fig. 2). This will likely translate to Method (6) attempting to find feasible colourings with fewer colour classes before Methods (4) and (5) are able to produce initial, feasible colourings.

Method (6) was also found to require significantly less time to achieve final, feasible colourings than Method (0) for all values of d with most values of p (again, see Table 4). This is probably due to the same argument given earlier with regards to the number of colour classes in the final, feasible colourings achieved by Method (6) compared to Method (0).

7 Conclusions and Future Work

In this paper we have presented several methods for modifying feasible colourings from one time-step of a dynamic random graph in order to help find a feasible colouring for the next time-step.

Our experiments have shown that, for both edge and vertex dynamic graphs, initial colourings with significantly fewer colour classes can be achieved by initially modifying a feasible k-colouring for G_t into an infeasible k-colouring for G_{t+1} and then passing this directly to the tabu search operator. However, there is a significant trade off with respect to the time required to achieve an initial, feasible colouring when these modification methods are applied. These methods were also found to achieve final, feasible colourings with the significantly more colour classes for some edge dynamic problems but significantly fewer colour classes for some vertex dynamic problems. The time required to achieve comparable final colourings via these methods is dependent on p.

It has also been shown that reducing a feasible colouring for G_t into a partial, proper colouring for G_{t+1} and then applying a constructive algorithm to the residual graph induced by the "uncoloured" vertices can also achieve initial, feasible colourings with significantly fewer colour classes when p is small enough. These modification methods were also shown to produce initial, feasible colourings in significantly less time for $d \in \{0.5, 0.9\}$. Finally, these methods also resulted in final, feasible colourings with the same or significantly fewer colour classes and require equal or significantly less time to do so.

Note that in this piece of work, all changes between time-steps of dynamic graphs have occurred completely at random; however, for some real world applications there may be some level of predictability. More specifically, we might have some knowledge of how edges and vertices are likely to change in the future. We wish to extend this research and explore how this sort of information can be used to our advantage in order to produce more robust colourings.

References

1. Aardal, K.I., Van Hoesel, S.P., Koster, A.M., Mannino, C., Sassano, A.: Models and solution techniques for frequency assignment problems. Ann. Oper. Res. **153**(1), 79–129 (2007)

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. Chaitin, G.J.: Register allocation & spilling via graph coloring. ACM Sigplan Not. **17**(6), 98–101 (1982)
4. Dupont, A., Linhares, A.C., Artigues, C., Feillet, D., Michelon, P., Vasquez, M.: The dynamic frequency assignment problem. Eur. J. Oper. Res. **195**(1), 75–88 (2009)
5. Erben, W.: A grouping genetic algorithm for graph colouring and exam timetabling. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 132–156. Springer, Heidelberg (2001)
6. Gyárfás, A., Lehel, J.: On-line and first fit colorings of graphs. J. Graph Theor. **12**(2), 217–227 (1988)
7. Harary, F., Gupta, G.: Dynamic graph models. Math. Comput. Model. **25**(7), 79–87 (1997)
8. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing **39**(4), 345–351 (1987)
9. 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)
10. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. J. Res. Nat. Bur. Stand. **84**(6), 489–506 (1979)
11. Lewis, R.: Constructing wedding seating plans: a tabu subject. In: Proceedings of the International Conference on Genetic and Evolutionary Methods (GEM). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), p. 1 (2013)
12. Lovász, L., Saks, M., Trotter, W.T.: An on-line graph coloring algorithm with sublinear performance ratio. Ann. Discrete Math. **43**, 319–325 (1989)
13. Monical, C., Stonedahl, F.: Static vs. dynamic populations in genetic algorithms for coloring a dynamic graph. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, pp. 469–476. ACM (2014)
14. Preuveneers, D., Berbers, Y.: ACODYGRA: an agent algorithm for coloring dynamic graphs. Symbolic Numer. Algorithms Sci. Comput. **6**, 381–390 (2004)
15. Qu, R., Burke, E.K., McCollum, B.: Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. Eur. J. Oper. Res. **198**(2), 392–404 (2009)
16. Tantipathananandh, C., Berger-Wolf, T., Kempe, D.: A framework for community identification in dynamic social networks. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 717–726. ACM (2007)

Particle Swarm Optimisation with Sequence-Like Indirect Representation for Web Service Composition

Alexandre Sawczuk da Silva(✉), Yi Mei, Hui Ma, and Mengjie Zhang

School of Engineering and Computer Science,
Victoria University of Wellington, Wellington, New Zealand
{Sawczualex,Yi.Mei,Hui.Ma,Mengjie.Zhang}@ecs.vuw.ac.nz

Abstract. Automated Web service composition, which refers to the creation of a complex application from pre-existing building blocks (Web services), has been an active research topic in the past years. The advantage of having an automated composition system is that it allows users to create new applications simply by providing the required parameters, instead of having to manually assemble the services. Existing approaches to automated composition rely on planning techniques or evolutionary computing (EC) to modify and optimise composition solutions *directly* in their tree/graph form, a complex process that requires several constraints to be considered before each alteration. To improve the search efficiency and simplify the checking of constraints, this work proposes an *indirect* Particle Swarm Optimisation (PSO)-based approach. The key idea of the indirect approach is to optimise a service queue which is then decoded into a composition solution by using a planning algorithm. This approach is compared to a previously proposed graph-based direct representation method, and experiment results show that the indirect representation can lead to a greater (or equivalent) quality while requiring a lower execution time. The analysis conducted shows that this is due to the design of the algorithms used for building and evaluating the fitness of solutions.

Keywords: Web service composition · Particle swarm optimisation · Quality of Service · Candidate representation

1 Introduction

Software developers around the world are well acquainted with *Web services*, which may be defined as applications that provide operations and/or data and are accessible via the network using communication protocols [7]. The modular nature of Web services has led users to think of them as building blocks for more complex applications, selected and integrated as needed from a repository of available candidates in a process known as *Web service composition* [5]. As the number of candidates in the repository grows and as composition tasks become

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 202–218, 2016.
DOI: 10.1007/978-3-319-30698-8_14

more complex, performing the selection and integration of services manually becomes increasingly difficult [12]. Additionally, if the repository contains multiple candidates with equivalent functionality but different quality attributes, then manually choosing the ideal alternative to include in a composition may become infeasible [8]. To overcome these challenges, researchers have been investigating the development of techniques to perform *automated Web service composition* [15]. By using these techniques, the *composition requestor* would be able to simply specify the inputs and outputs of the desired application, and an *automated composition system* would then correctly select and integrate services into a correct composition solution.

There are normally two tasks to be considered in Web service composition: maintaining the *correctness* of solutions, i.e. ensuring atomic services are connected in a way that can be executed at run time, and optimising solutions according to their overall *Quality of Service*. To accomplish this, there are typically three different kinds of methods: the first group focuses on creating a correct composition [18]; the second group optimises the quality of compositions assuming that an abstract workflow is already known [25]; the third group attempts to address both of these concerns simultaneously, creating a correct workflow and at the same time optimising the quality of the services included in the composition [20]. However, simultaneously accomplishing these two tasks increases the complexity of these approaches, since the optimisation must also respect a number of interrelated constraints [22].

The overall goal of this paper is to investigate an indirect representation to the problem of Web service composition, proposing a Particle Swarm Optimisation (PSO)-based approach [6] that represents each solution candidate as a queue of services. Each queue is decoded into the corresponding composition workflow by using a specific graph-planning algorithm, verifying the correctness of the connections between services. To the best of our knowledge, the idea of encoding/decoding solutions has not been used before in the field of Web service composition. This work accomplishes three objectives:

1. It identifies suitable encodings for representing a queue of services within a PSO particle vector.
2. It proposes decoding algorithms that efficiently create composition workflows from a service queue as efficiently as possible.
3. It compares the indirect approach with a state-of-the-art direct composition approach to verify that there is indeed a performance gain without a loss of solution quality.

The remainder of this paper is organised as follows: Sect. 2 provides the fundamental background on the Web service composition problem, including a literature review; Sect. 3 describes the indirect representation proposed in this paper; Sect. 4 describes the experiments conducted to test the performance of the novel PSO methods; Sect. 5 presents the results of these experiments; Sect. 6 concludes the paper.

2 Background

2.1 Problem Description and Example

The fundamental idea of Web service composition is to combine Web services into a structure that accomplishes a more complex task. A *Web service* $S = (input(x_1, x_2, ..., x_n),\ output(y_1, y_2, ..., y_n),\ QoS(time,\ cost,\ availability,\ reliability))$ requires a set of *inputs*, produces a set of *outputs*, and has an associated set of *quality attributes*. The fundamental elements in the composition problem are a *service repository* $SR = \{S_1, ..., S_m\}$ containing the services, and a *composition request* $R = (input(i_1, i_2, ..., i_n), output(o_1, o_2, ..., o_n))$ which specifies the overall inputs that should be made available when executing the composition as well as the overall outputs the composition should produce. The objective of this problem is to create a service composition with the best possible overall quality attributes, optimised according to a set of *objective functions* $f_1, f_2, ..., f_n$, where $1 \leq i \leq n$ and $i \in QoS$. There are three fundamental *constraints* required in a composition solution: firstly, the inputs of each service must be fully satisfied by predecessor services in the composition $(\forall x : input(x) \supseteq output(pred(x)))$; secondly, the outputs of the starting node of a composition must be the composition requests overall inputs $(output(s) = input(R))$; thirdly, the inputs of the ending node of a composition must be the composition requests overall outputs $(input(e) = output(R))$. The travel problem, a well-known Web service composition example which has been extensively described in the literature [19,21], is shown in Fig. 1 as a concrete example of this problem description.

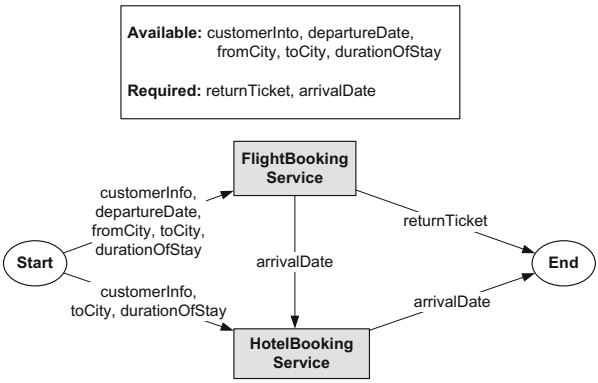


Fig. 1. Example of a solution to a Web service composition task [20].

2.2 Quality of Service and Composition Constructs

When creating compositions, it is necessary to pay attention to the Quality of Service (QoS) properties of each selected service, i.e. a QoS-aware composition

approach is needed. These non-functional criteria may be quite in certain field, e.g. in finance. There exist many Web service quality properties, from security levels to service throughput [14]. Based on the properties selected in previous works [9,26], in this paper we consider four of them: the probability of a service being available (A) upon request, the probability of a service providing a reliable response to a request (R), the expected service time limit between sending a request to the service and receiving a response (T), and the execution cost to be paid by the service requestor (C). The higher the probabilities of a service being available and of it producing a reliable response, the higher its quality with regard to A and R; conversely, the services with the lowest response time and execution cost have the highest quality with regard to T and C. The configuration of services in a composition is dictated by constructs used in building a workflow showing how services connect to each other [27]. This work considers two composition configurations, sequence and parallel, that are recognised by Web service composition languages such as BPEL4WS [4,26]. These two constructs are described as follows.

Sequence Construct. Services in a composition construct are chained sequentially, so that the outputs of a preceding service fulfil the inputs of the subsequent one, as shown in Fig. 2. The total time (T) and total cost (T) for a sequence can be calculated simply by adding the individual time and cost values of the composing services. The availability (A) and reliability (R) of individual services are expressed as probabilities, therefore the total composition values can be calculated by multiplying the individual service values.

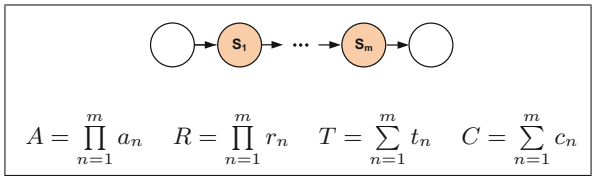


Fig. 2. Sequence construct and formulae for calculating its QoS properties [26].

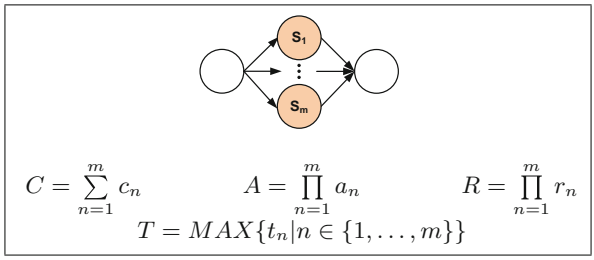


Fig. 3. Parallel construct and formulae for calculating its QoS properties [26].

Parallel Construct. The inputs of components in a parallel construct are satisfied independently, which allows these services to be executed simultaneously. The output produced by this construct can then be provided to any subsequent services, as shown in Fig. 3. The overall QoS values for the parallel construct are calculated using the same formulae applied to the sequence construct, with the exception of the total time (T), which is simply the time of the component service that takes the longest to execute.

2.3 Related Work

In addition to planning [17] and traditional optimisation strategies [28], a wide variety of Evolutionary Computing (EC) approaches have been applied to the problem of Web service composition [16,24]. One of the earliest works in this area [3] applies genetic algorithms to optimise the overall Quality of Service (QoS) of a composition. Its objective is to select a set of concrete services that fulfil the required functionality of their abstract counterparts, ensuring that the selected set results in a composition with the best possible quality. Even though this approach takes QoS into account, it is not capable of performing *fully automated* composition, which is when the composition workflow is automatically deduced at the same time that the services to include in the composition are identified. Several works employ particle swarm optimisation (PSO) to solving the problem of service composition [13,25,29], but similarly to genetic algorithms they focus exclusively on semi-automated composition.

Another approach [18] employs Genetic Programming (GP) to perform fully automated Web service composition, representing solutions as trees with candidate services as the leaf nodes and composition constructs as the inner nodes. A context-free grammar is used to generate new individuals at the beginning of the evolutionary process, as well as ensuring structural correctness during the crossover and mutation operations. Despite its favourable experimental results, this approach has the shortcoming of neglecting the Quality of Service of compositions and optimising candidates according to workflow topology measures such as the length of the longest path in the composition and the number of atomic services included.

Finally, some approaches both create the composition workflow and optimise the quality of the overall composition [20,26]. These works accomplish this by relying on variable-size solution representations (trees or directed acyclic graphs) and by measuring the quality of candidate compositions through the fitness function. In [26], the fitness function is responsible for penalising solutions that are not *functionally correct*, i.e. solutions that contain services whose inputs have not been entirely fulfilled; in [20], candidate initialisation and genetic operators are restricted to only produce functionally correct solutions. While these approaches do consider both workflow creation and quality improvement simultaneously, they perform operations to the solution workflows directly, which requires quite complex constraint checks.

3 PSO with Indirect Representation

The core idea explored in this work is to optimise solutions indirectly, using a representation that is then decoded into the final composition. Given that the verification of correctness constraints is very time-consuming, the indirect representations, which are usually simpler than the direct one, are expected to simplify or even remove the computationally expensive tasks of constraint checking and solution repairing, and thus improve the search efficiency. However, the implication of using this approach is that its efficiency and the quality of the solutions produced is affected by two factors: the specific representation used during the optimisation process, and the decoding process used for translating the particular representation into an actual service composition. In the context of an evolutionary computing approach, the use of an indirect representation requires candidates to be decoded before the usual fitness evaluation step.

The representation investigated in this work uses a linear format that is meant to represent a sequence of services, i.e. a service queue. The composition solution is then decoded from this queue by using an algorithm that adds services one by one to the solution according to the queue's ordering. PSO is the technique chosen for the indirect optimisation, since the linear structure of its particles naturally lends itself to representing a service queue. This approach follows the usual PSO steps [6], though with some particularities shown in Algorithm 1. Firstly, the size of particles is determined based on the number of candidate services being considered for the composition, with each candidate service being mapped to an index of the particle's position vector (each position holds a weight between 0.0 and 1.0, inclusive). Secondly, solutions must be built using a graph-building algorithm before their fitness can be calculated; a queue of services is generated from the particle's position vector and used as the input for the algorithm, which decodes a corresponding solution graph from it. Finally, the particle's fitness can be calculated from this corresponding solution graph. Two PSO variations are presented in this work, and they are discussed separately in the subsections below.

3.1 A Simple Forward Decoding PSO

The first PSO method proposed investigates a forward-decoding strategy for a particle vector of weight values, used for sorting all available services in a queue. In this method, a simple service queue is constructed using the particle's position vector before it can be translated into a composition graph. This is done by checking the service-to-index mapping for the particles' position vector. Each service is placed on the queue with an associated weight, which is retrieved by accessing the position vector with the index mapped to that service. This queue is then sorted according to these weights, placing the services with the highest weight at the head of the queue, and those with the lowest weight at the tail. Note that if two or more services have the same weight, then the ordering between them may vary.

Algorithm 1. Steps of the PSO-based Web service composition technique.

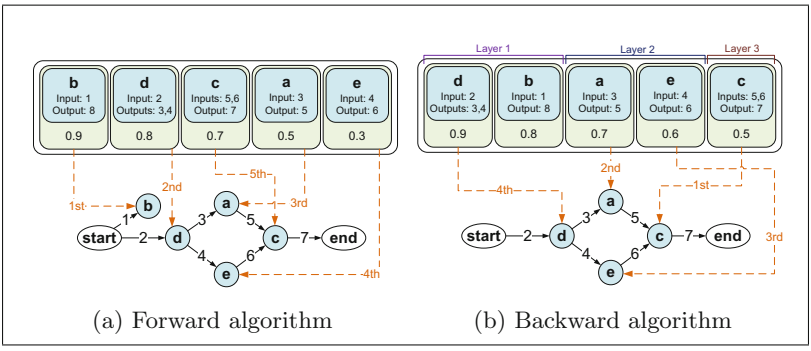
```
1:  Randomly initialise each particle in the swarm;
2:  while max. iterations not met do
3:      forall the particles in the swarm do
4:          Create queue of services using the particle's position vector;
5:          Build the corresponding composition graph using the queue;
6:          Calculate the fitness of the resulting graph;
7:          if fitness value better than pBest then
8:              Assign current fitness as new pBest;
9:          else
10:             Keep previous pBest;
11:     Assign best particle's pBest value to gBest, if better than gBest;
12:     Calculate the velocity of each particle according to the equation:
13:         v_id = v_id + c1 * rand() * (p_id − x_id) + c2 * rand() * (p_gd − x_id);
14:     Update the position of each particle according to the equation:
15:         x_id = x_id + v_id;
```

Graph-Building Algorithm. By determining the service queue represented in a particle, it is then possible to build a composition graph from that service ordering, based on the Graphplan technique discussed in [2]. The graph is built in a forward way – from the *start* node towards the *end* node – to prevent the formation of cycles, which may lead to the addition of nodes that do not contribute to reaching the *end* (i.e. dangling nodes). To address this, after the graph has been constructed, it is submitted to a function that removes these redundant nodes.

Algorithm 2. Generating a composition graph from a queue

```
    Input  : I, O, queue
    Output: composition graph G
1:  Create start node with outputs I and end node with inputs O;
2:  Create graph G containing the start node;
3:  Create set of available outputs containing start outputs;
4:  while available outputs do not satisfy end inputs do
5:      Get next candidate from queue;
6:      if candidate inputs are satisfied by available outputs then
7:          Connect node to graph;
8:          Remove it from queue and go back to the queue's beginning;
9:  Connect end node;
10: Remove dangling nodes from graph G;
11: return G;
```

As shown in Algorithm 2, the values initially required are the composition task inputs (I), task outputs (O), and a *queue* of services as its input; this leads to the creation of a composition graph G. Firstly, the *start* node is added to the graph G, its outputs are added to a set that records all available outputs from the nodes currently in the graph. Then, the following steps are repeated until the available outputs can be used to fulfil all of the inputs of *end*: the next node of the queue is retrieved as a candidate; if all of its inputs can be fulfilled, the candidate is connected to the graph, its outputs are added to the set of available outputs, and it is removed from the queue; otherwise, the candidate in the next queue position is considered. Finally, *end* is connected to the graph, any dangling nodes are removed, and G is returned. This process is illustrated in Fig. 4a, where node b is dangling. Note that different sequences may lead to completely different solution topologies, thus preventing this approach from being overly restrictive.



(a) Forward algorithm (b) Backward algorithm

Fig. 4. Approaches for decoding a particle solution.

Fitness Calculation. The fitness for a candidate graph is calculated using a function that evaluates its overall QoS values, considering the four attributes discussed in Subsect. 2.2. Note that when using a forward graph-building method, the fitness function can only be calculated once the entire graph has been decoded (as dangling nodes must not be included QoS calculations). The QoS attributes are combined using a commonly used weighted sum [23], according to the function $fitness_i = w_1 A_i + w_2 R_i + w_3(1 - T_i) + w_4(1 - C_i)$, where $\sum_{i=1}^{4} w_i = 1$. A, C, and R are calculated using each atomic service in the graph according to the formulae shown in Figs. 2 and 3; T, on the other hand, is determined by adding the individual times of the services that form the longest path in the graph, from start to end. The time is calculated based on the longest graph path because this allows us to handle both parallel and sequence constructs at the same time. The output of the fitness function is within the range [0, 1], with 1 representing the best possible fitness and 0 representing the worst. To ensure that the final result of the sum is within this range, the values of A, C, R and T

must all be normalised between 0 and 1 (for time and cost, the maximum value used in the normalisation is the highest individual service value in the repository multiplied by the total number of services in the repository) [23]. The weights in the function are specified by the user.

3.2 Layered Backward Decoding PSO

A variation to the PSO-based method proposed in the previous section, which considers the use of service layers, was also developed with the objective of further improving the efficiency of the indirect technique. This variation was implemented because the use of layers allows solutions to be decoded using a backwards algorithm, which is more computationally effective. A *layer* in this context refers to a group of discovered services whose inputs that can be completely fulfilled by a set of outputs (either given or from previous layers). The same steps used in the non-layered method are also employed here, with three fundamental differences: firstly, before initialising the population candidate services are mapped to an index in the particle's position vector according to the position information from the layers; secondly, the decoding and evaluation of the solution represented by a particle are performed in a single step by using a new algorithm that does not require the creation of a graph; finally, a graph representing the global best solution is created after the optimisation process has finished running. Each of these steps is explained below.

Layer Identification and Particle Mapping. The unique feature of the PSO method proposed here is that it identifies the *composition layer* to which a service belongs. Before the optimisation process begins, the service repository is run through a discovery process [23] that identifies the services that could be possibly used in the composition. As shown in Algorithm 3, this filtering process requires the set of inputs (I) and the set of outputs (O) from the overall composition task, in addition to the service repository (R); given these inputs, it produces a list of candidate service layers that are relevant to the composition (L). The algorithm keeps track of all available outputs so far, and uses them to discover additional layers: if a previously undiscovered service has all of its inputs satisfied by the set of available outputs, then it is added to the current layer and its outputs are added to the set of available outputs. The discovery continues until no additional layers are found, and the final step verifies whether the desired composite output O can in fact be achieved using the services in the repository. Once the composition layers are identified, the particle mapping takes place. As shown in in Fig. 5, each service layer is mapped to contiguous particle indices, effectively segmenting particles according to the layers. This segmentation facilitates the solution decoding process to be discussed in the following subsection, as the connectivity information can be reused during the optimisation process.

Algorithm 3. Discovering relevant service composition layers [23].

Input : I, O, R
Output: service layers L
Initialise output set with I;
Discover services satisfied by output set;
while *at least one service discovered* **do**
    Add services as the next layer in L;
    Add the outputs of these services to the output set;
    Discover additional services satisfied by the updated output set;
if *Output set satisfies* O **then**
    return L;
else
    Report no solution;

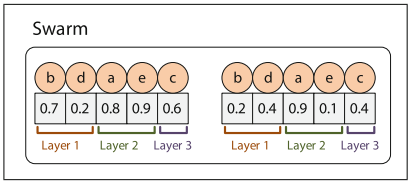


Fig. 5. Mapping of services to particles according to layers.

Solution Decoding and Fitness Calculation. The solution decoding step employed in the layered PSO method is fundamentally different from that of the simple PSO method. Since particles are segmented by layers, it becomes possible to build graph solutions backwards (i.e. from the graph's end node towards the graph's start node) without leading to cycles being formed, provided that only services from previous layers are used to fulfil the input of a service in the current layer. Another difference is that the solution decoding process shown here does not produce a graph structure at the end of its execution, and instead calculates the fitness of the solution at the same time that the solution is identified. As discussed earlier, before the decoding process can begin it is necessary to order candidate services according to the weights contained in the particle. Instead of generating a single queue, however, we generate one individual queue for each of the layers mapped to that particle, creating a series of sorted layers (L). These sorted layers are then provided in conjunction with the solution's *end* node as the input to Algorithm 4, which calculates the corresponding fitness (f) to the particle's solution.

The algorithm works by keeping track of all service inputs that need to be satisfied, initialising it to contain all the inputs required by the end node. Then, the algorithm is executed from the last layer towards the first layer, each time performing the same series of steps. Firstly, all the inputs in set to satisfy that correspond to services in the current layer are identified. Then, all previous

sorted layers are merged into a single service queue that is then used to fulfil the current inputs. As each service is selected from the queue to satisfy a given input, its QoS values are added to the QoS totals and all of its inputs are placed in the next-to-satisfy set. The *time* aspect of QoS is calculated by keeping track of the longest total time required by the services in previously processed layers, and by updating this total time with each new service addition. After all layers are satisfied, the fitness is calculated using the same fitness calculation as the previous PSO and the algorithm returns the result. A simplified depiction of the order in which this process is carried out is shown in Fig. 4b.

Algorithm 4. Algorithm for decoding solutions and calculating their fitness.

Input : *end*, sorted layers L
Output: fitness f

```
Initialise variables to keep track of QoS:
cost = 0, availability = 1, reliability = 1;
Set all inputs of end as the next to satisfy, associating each input with time 0
and layer |L| + 1;
for all sorted layers in L, from end to first do
    Identify the inputs from next-to-satisfy set that correspond to services in
    this layer;
    while not all of these inputs have been satisfied do
        Find the next service from the previous layers with the highest
        weight;
        if service outputs satisfy at least one input then
            Update running QoS values with service QoS (add to running
            cost, multiply with running availability and reliability);
            Add the inputs of service to the set of next inputs to satisfy, each
            associated with (service time + highest time from satisfied
            inputs) and current layer position;
Find the total composition time as the highest from the remaining set of
next inputs to satisfy;
Calculate fitness f using total QoS values;
return f;
```

Construction of Final Graph. As the decoding algorithm described in the previous subsection does not create a directed acyclic graph out of every candidate solution, at the end of the run it is necessary to build a solution graph out of the overall fittest particle by using Algorithm 5. This algorithm has a very similar structure to the decoding one, but instead of calculating QoS values it connects services from earlier layers whose outputs fulfil the inputs of services in later layers. As before, the algorithm goes through all composition layers, though in this case the final step is to connect the *start* node to any service with inputs that are still unfulfilled. Finally, the composition graph G is returned.

Algorithm 5. Algorithm for building final graph solution.

Input : *start*, *end*, sorted layers L
Output: final graph G

```
1:  Create graph G containing the end node;
2:  Set all inputs of end as the next to satisfy, associating each input with the
    end node and layer |L| + 1;
3:  forall the sorted layers in L, from end to first do
4:      Identify the inputs from next-to-satisfy set that correspond to services in
        thWeb servicesis layer;
5:      while not all of these inputs have been satisfied do
6:          Find the next service from the previous layer with the highest weight;
7:          if the outputs of this service satisfy at least one of the inputs for this
            layer then
8:              Add service node to graph;
9:              Add edges connecting this service node to the nodes whose
                inputs it satisfies;
10:             Add the inputs of the service node to the set of next inputs to
                satisfy, each associated with the service node and current layer
                position;
11: Add start node to graph;
12: Add edges connecting start node the associated nodes of all remaining
    inputs in the next-to-satisfy set;
13: return G;
```

4 Experiment Design

Experiments were conducted to evaluate the performance of the PSO-based indirect composition methods in comparison to a graph-based direct composition approach [20], with the hypothesis that the indirect representation will produce solutions with equivalent quality but requiring shorter execution times. The graph-based approach was chosen for the comparison because of its flexibility, as it can also simultaneously optimise the quality of solutions and ensure their correctness. All experiments were conducted on a personal computer with 8 GB RAM and an Intel Core i7-4770 CPU (3.4 GHz). The graph-based and PSO methods were compared using the datasets and tasks from WSC-2008 [1] and WSC-2009 [10], which contain service descriptions and their associated QoS attributes. 30 independent runs were carried out for each approach using each dataset. The parameters were chosen based on common settings from the literature [6,11]. For both PSO methods, 100 iterations were run for a swarm of 30 particles, having both c_1 and c_2 as 1.49618, w as 0.7298 and all weights in the fitness function were 0.25. For the graph-based approach, a population of size 500 was evolved for 51 generations, with crossover probability of 0.8, both mutation and reproduction with a probability of 0.1, tournament selection with a tournament of size 2, and all fitness function weights as 0.25.

5 Results

Results are presented and discussed in the following subsections, where the solution fitness and execution time means are shown accompanied by the standard deviation. A Wilcoxon signed-rank test at 95 % confidence level was conducted to ascertain whether the differences between the two PSO methods and the graph-based approach are statistically significant, and the symbols ↑ and ↓ are used to indicate values significantly larger than the graph-based approach and significantly smaller than the graph-based approach, respectively.

5.1 Solution Fitness

The fitness results presented in Table 1 generally show that the solution fitness produced by the PSO-based methods is equivalent to that of the graph-based approach. However, it must also be noted that the fitness of PSO solutions is significantly higher for a number of datasets (08-3, 08-6, 08-7, 09-4), whereas this is not the case for the graph-based approach. Thus, these results indicate that the PSO-based methods are preferable when the focus of the composition process is on the quality of the resulting solutions. This difference in quality was investigated by manually comparing the solutions produced by the graph-based approach to those produced by layered PSO for dataset WSC-08-6, where the difference is the most pronounced. For most runs with this dataset, the layered PSO produced many solutions with fitness 0.4980, whereas the graph-based approach did not produce any solutions with fitness higher than 0.4976. When comparing the topology of the solutions, an interesting observation was made: some of the layered PSO solutions included more services (42) and edges (107) than the graph-based solutions (40 services, 113 edges), but still had better overall quality. A hypothesis for this outcome is that the operators used by the graph-based approach discourage extensive exploitation of specific solution topologies, since its operators are fundamentally based on topological changes; the indirect approach, on the other hand, updates solutions by changing particle weights that may or may not influence the decoded solution topology. This flexibility potentially allows larger structures to be exploited by the layered PSO for longer, until an area with promising quality is found.

5.2 Execution Time

With regard to the execution time of the two techniques, shown in Table 2, an interesting pattern is observed: while the time required by the simple PSO-based method is consistently higher than that of the graph-based approach, the time required by the layered PSO-based method is consistently lower. This is the case for two reasons: firstly, the layered PSO decodes solutions backwards (from end to start), meaning it does not explore paths that do not ultimately connect the beginning and the end of a composition; secondly, during the decoding process the layered PSO goes through the services in the particle roughly $|layers| \times |services|$ in the worst case, whereas the worst case for the simple

Table 1. Mean solution fitness results for the two PSO-based methods and the graph-based composition approach.

Dataset (# servs.)	Simple PSO	Layered PSO	Graph-based
WSC-08-1 (158)	0.4928 ± 0.00118	0.4928 ± 0.00119	0.4916 ± 0.0000517
WSC-08-2 (558)	0.5949 ± 0.0130	0.5936 ± 0.0140	0.5993 ± 0.00
WSC-08-3 (608)	0.4894 ± 0.000289 ↑	0.4902 ± 0.000201 ↑	0.4879 ± 0.000145
WSC-08-4 (1041)	0.5120 ± 0.00256	0.5141 ± 0.000720	0.5088 ± 0.00124
WSC-08-5 (1090)	0.4971 ± 0.000139	0.4971 ± 0.0000993 ↑	0.4969 ± 0.0000414
WSC-08-6 (2198)	0.4979 ± 0.000124 ↑	0.4980 ± 0.000124 ↑	0.4976 ± 0.0000229
WSC-08-7 (4113)	0.4993 ± 0.0000381 ↑	0.4993 ± 0.0000403 ↑	0.4991 ± 0.0000191
WSC-08-8 (8119)	0.4994 ± 0.00000919	0.4994 ± 0.0000303	0.4994 ± 0.00000219
WSC-09-1 (572)	0.5630 ± 0.0125	0.5727 ± 0.0173	0.5664 ± 0.00991
WSC-09-2 (4129)	0.4993 ± 0.0000255	0.4993 ± 0.0000526	0.4993 ± 0.00000804
WSC-09-3 (8138)	0.5064 ± 0.00219	0.5058 ± 0.00293	0.5060 ± 0.00121
WSC-09-4 (8301)	0.4993 ± 0.0000473 ↑	0.4994 ± 0.0000508 ↑	0.4992 ± 0.0000103
WSC-09-5 (15211)	0.4996 ± 0.0000105	0.4996 ± 0.0000130	0.4996 ± 0.00000531

PSO is $|services|!$. These two key differences cause the layered method to check for significantly less potential service connections, which accounts for the time difference. A simple test was carried out to confirm this supposition, running the simple and layered PSO methods once each with WSC2008-8 (using the same settings as before) and counting how many times each of those methods checked for potential service connections during the particle decoding process. The simple PSO count was 362,045,030 after finishing the run, while the layered PSO count was 2,126,349. From the results in Table 2, we see that the layered-to-simple execution time ratio for WSC2008-8 is roughly 1:70, while the layered-to-simple count ratio is roughly 1:170. The fact that these two execution aspects have similarly high ratios lends credence to the conjecture that the time difference is correlated with the number of times the decoding algorithms check for potential connections during a run.

5.3 Discussion

The results displayed in this section show that the indirect approach performs well with a PSO implementation, though it was designed to work in a general sense. Thus, we theorise that other solution representations, decoding algorithms, and optimisation techniques in an indirect context could also yield superior results to those produced by direct approaches. Fundamentally, the indirect approach facilitates the checking of correctness constraints in a solution by restricting it to the decoding step, as opposed to incorporating it into the search process. This separation reduces the risk of overly constraining the search process, since invalid solutions can be easily prevented when using a robust decoding algorithm. The potential disadvantage of the indirect approach is that

Table 2. Mean execution time in seconds for the two PSO-based methods and the graph-based composition approach.

Dataset (# servs.)	Simple PSO	Layered PSO	Graph-based
WSC-08-1 (158)	2.1 ± 0.5	0.3 ± 0.1	3.2 ± 0.4
WSC-08-2 (558)	$4.1 \pm 2.1 \uparrow$	$0.4 \pm 0.1 \downarrow$	2.6 ± 0.4
WSC-08-3 (608)	$24.7 \pm 5.9 \uparrow$	$0.9 \pm 0.1 \downarrow$	14.3 ± 1.1
WSC-08-4 (1041)	$16.4 \pm 6.5 \uparrow$	0.5 ± 0.1	6.1 ± 0.6
WSC-08-5 (1090)	$29.1 \pm 8.2 \uparrow$	$0.9 \pm 0.1 \downarrow$	10.1 ± 1.3
WSC-08-6 (2198)	$195.6 \pm 45.7 \uparrow$	$3.8 \pm 0.2 \downarrow$	21.9 ± 1.5
WSC-08-7 (4113)	$202.4 \pm 61.0 \uparrow$	$3.2 \pm 0.4 \downarrow$	52.6 ± 3.7
WSC-08-8 (8119)	$539.0 \pm 145.0 \uparrow$	$7.4 \pm 0.6 \downarrow$	75.2 ± 13.8
WSC-09-1 (572)	3.8 ± 1.3	$0.4 \pm 0.1 \downarrow$	3.2 ± 0.5
WSC-09-2 (4129)	$168.6 \pm 38.8 \uparrow$	$3.2 \pm 0.3 \downarrow$	18.1 ± 1.6
WSC-09-3 (8138)	260.0 ± 99.3	$5.0 \pm 1.1 \downarrow$	23.3 ± 0.6
WSC-09-4 (8301)	$1378.4 \pm 577.5 \uparrow$	$21.0 \pm 3.3 \downarrow$	65.1 ± 2.9
WSC-09-5 (15211)	$2124.0 \pm 580.0 \uparrow$	$11.9 \pm 2.1 \downarrow$	151.1 ± 17.8

the chosen solution representation may lead to a very large search space, therefore care is needed when making this design decision.

6 Conclusions

This work introduced two PSO-based QoS-aware Web service composition methods that rely on an indirect solution representation, as opposed to the direct representations used by current works in the area. The key idea of these methods is to optimise a queue of candidate atomic services, identifying the sequence that leads to the construction of a composition with the highest possible quality. In order to evaluate the quality of a candidate, the queue is fed into algorithms that decode the underlying solution and calculate its fitness. These PSO-based methods were compared to a graph-based approach with direct solution representation, with results showing that the quality of the solutions produced by PSO generally matches or surpasses those produced by the graph-based approach but with shorter execution times. Additionally, the indirect representation successfully handles the complexities faced when simultaneously optimising the quality and ensuring the correctness of service compositions. Future works in this area should investigate ways of further improving and simplifying the decoding algorithms and solution representations, as well as considering alternative methods such as genetic algorithms and multi-objective optimisation.

References

1. Bansal, A., Blake, M.B., Kona, S., Bleul, S., Weise, T., Jaeger, M.C.: WSC-08: continuing the web services challenge. In: 2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, pp. 351–354. IEEE (2008)
2. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1), 281–300 (1997)
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 7th Annual Conference on Genetic and Evolutionary Computation, pp. 1069–1075. ACM (2005)
4. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semant. Sci. Serv. Agents World Wide Web **1**(3), 281–308 (2004)
5. Dustdar, S., Papazoglou, M.P.: Services and service composition-an introduction (services und service komposition-eine einführung). IT - Inf. Technol. (vormals it+ ti) **52**(2), 86–92 (2008)
6. Eberhart, R.C., Shi, Y.: Particle swarm optimization: developments, applications and resources. In: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, pp. 81–86. IEEE (2001)
7. Gottschalk, K., Graham, S., Kreger, H., Snell, J.: Introduction to web services architecture. IBM Syst. J. **41**(2), 170–177 (2002)
8. Grønmo, R., Jaeger, M.C.: Model-driven semantic web service composition. In: 12th Asia-Pacific Software Engineering Conference, APSEC 2005, p. 8. IEEE (2005)
9. Jaeger, M.C., Mühl, G.: Qos-based selection of services: The implementation of a genetic algorithm. In: 2007 ITG-GI Conference on Communication in Distributed Systems (KiVS), pp. 1–12. VDE (2007)
10. Kona, S., Bansal, A., Blake, M.B., Bleul, S., Weise, T.: WSC-2009: a quality of service-oriented web services challenge. In: IEEE Conference on Commerce and Enterprise Computing, CEC 2009, pp. 487–490. IEEE (2009)
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT Press, Cambridge (1992)
12. Lécué, F., Léger, A.: A formal model for semantic web service composition. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 385–398. Springer, Heidelberg (2006)
13. Ludwig, S., et al.: Applying particle swarm optimization to quality-of-service-driven web service composition. In: 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA), pp. 613–620. IEEE (2012)
14. Menasce, D.: QoS issues in web services. IEEE Internet Comput. **6**(6), 72–75 (2002)
15. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Comput. **8**(6), 51–59 (2004)
16. Pejman, E., Rastegari, Y., Esfahani, P.M., Salajegheh, A.: Web service composition methods: a survey. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1 (2012)
17. Pistore, M., Barbon, F., Bertoli, P.G., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: Bussler, C.J., Fensel, D. (eds.) AIMSA 2004. LNCS (LNAI), vol. 3192, pp. 106–115. Springer, Heidelberg (2004)

18. Rodriguez-Mier, P., Mucientes, M., Lama, M., Couto, M.I.: Composition of web services through genetic programming. Evol. Intell. **3**(3–4), 171–186 (2010)
19. Sheng, Q.Z., Qiao, X., Vasilakos, A.V., Szabo, C., Bourne, S., Xu, X.: Webservices composition: a decades overview. Inf. Sci. **280**, 218–238 (2014)
20. da Silva, A.S., Ma, H., Zhang, M.: GraphEvol: a graph evolution technique for web service composition. In: Chen, Q., Hameurlain, A., Toumani, F., Wagner, R., Decker, H. (eds.) DEXA 2015. LNCS, vol. 9262, pp. 134–142. Springer, Heidelberg (2015)
21. Tang, M., Ai, L.: A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. In: 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)
22. Venkatraman, S., Yen, G.G.: A generic framework for constrained optimization using genetic algorithms. IEEE Trans. Evol. Comput. **9**(4), 424–435 (2005)
23. Wang, A., Ma, H., Zhang, M.: Genetic programming with greedy search for web service composition. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013, Part II. LNCS, vol. 8056, pp. 9–17. Springer, Heidelberg (2013)
24. Wang, L., Shen, J., Yong, J.: A survey on bio-inspired algorithms for web service composition. In: IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 569–574. IEEE (2012)
25. Wang, W., Sun, Q., Zhao, X., Yang, F.: An improved particle swarm optimization algorithm for qos-aware web service selection in service oriented communication. Int. J. Comput. Intell. Syst. **3**(sup01), 18–30 (2010)
26. Yu, Y., Ma, H., Zhang, M.: An adaptive genetic programming approach to qos-aware web services composition. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 1740–1747. IEEE (2013)
27. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: Proceedings of the 12th International Conference on World Wide Web, pp. 411–421. ACM (2003)
28. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng. **30**(5), 311–327 (2004)
29. Zhao, X., Song, B., Huang, P., Wen, Z., Weng, J., Fan, Y.: An improved discrete immune optimization algorithm based on pso for qos-driven web service composition. Appl. Soft Comput. **12**(8), 2208–2216 (2012)

Particle Swarm Optimization for Multi-Objective Web Service Location Allocation

Boxiong Tan, Yi Mei, Hui Ma(✉), and Mengjie Zhang

Victoria University of Wellington, Wellington, New Zealand
titantbx1989215@gmail.com, {yi.mei,hui.ma,mengjie.zhang}@ecs.vuw.ac.nz

Abstract. Web service location allocation problem is an important problem in the modern IT industry. In this paper, the two major objectives, i.e. deployment cost and network latency, are considered simultaneously. In order to solve this new multi-objective problem effectively, we adopted the framework of binary Particle Swarm Optimization (PSO) due to its efficacy that has been demonstrated in many optimization problems. Specifically, we developed two PSO variants, one with weighted-sum fitness function (WSPSO) and the other with dominance-based fitness function. Concretely, it uses the fast Non-dominate Sorting scheme, and thus is called NSPSO. The experimental results showed that both PSO variants performed better than NSGA-II, which is the one of the most commonly used multi-objective genetic algorithms. Furthermore, we have found that NSPSO achieved a more diverse set of solutions than WSPSO, and thus covers the Pareto front better. This demonstrates the efficacy of using the dominance-based fitness function in solving multi-objective Web service location allocation problem.

Keywords: Web service location allocation · Particle swarm optimization · Combinatorial optimization

1 Introduction

The *Web Service Location Allocation Problem* (WSLAP) is a significant problem that is important for many modern IT enterprises. Given a set of *Web services* and *candidate locations*, WSLAP is to assign each Web service to at least one location (one or more copies) to optimize certain objective such as the total deployment cost and response time. To accommodate business agility, it is usually preferred to use existing applications instead of developing them from scratch. To this end, a contemporary approach is to package the software resources as Web services (e.g. in the service oriented architecture [6,18]), which are well-defined, self-contained modules that provide sltandard business functionality and are independent of the state or context of other services [20]. It has been demonstrated that the Web service technology has the advantages of convenience, low cost and capacity to be composed into high-level business processes

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 219–234, 2016.
DOI: 10.1007/978-3-319-30698-8_15

[1]. This provides possibility of combining coarse-grained Web servies to build complex applications using standards such as WS-BPEL [17].

In practice, the Web services are generally located in some physical places (e.g. servers) by Web server providers, and can be called by users from various locations. In this situation, how to select proper locations for the Web services becomes an important problem. Therefore, one needs to assign the given Web services to proper locations. In WSLAP, in addition to the functionality requirement (e.g. the system can response to any type of requests), there are a number of *Quality of Service* (QoS) objectives for the Web service providers to consider to become competitive in the market. QoS, also known as non-functional requirements to Web services, is the degree to which a Web service meets specified requirements or user needs [25]. The common QoS measures include deployment cost, response time, security and availability. Web service location allocation has significant impact on two QoS measures, i.e. deployment cost and response time. Therefore, in this paper, we study Web service location allocation with two objectives, minimizing the deployment cost and network latency.

It is obvious that the two objectives are conflicting with each other. For example, to reduce the deployment cost, one needs to reduce the number of Web services deployed. This will increase the network latency due to the lack of services nearby. The deployment cost and network latency have been considered separately in literature [9,12]. However, to the best of our knowledge, there is no study trying to minimize the cost and response time simultaneously.

In our study, we aim to solve the Multi-Objective WSLAP (MO-WSLAP) that minimizes the cost and response time simultaneously. Instead of providing a single solution, we expect to provide a set of trade-off solutions, which are so-called *Pareto optimal* solutions. Evolutionary algorithms are chosen to solve the problem since they maintain a population of individuals during the search, and thus are able to provide a set of solutions in a single run. To be more specific, the framework of binary Particle Swarm Optimization (PSO) was adopted here because it has been successfully applied to many real-world optimization problems.

In summary, our goals in the paper are given as follows.

1. The total deployment cost and network latency simultaneously are considered, which leads to a Multi-Objective WSLAP (MO-WSLAP);
2. Two binary PSO approaches are designed for solving the MO-WSLAP, considering different multi-objective fitness assignment schemes;
3. The efficacy of using binary PSO to solve the MO-WSLAP are verified by comparing with a well known multi-objective optimization algorithm (NSGA-II).

The rest of the paper is organized as follows: Sect. 2 introduces the background, including the problem description and related work. Then, the PSO algorithms proposed for solving MO-WSLAP is described in Sect. 3. The experimental studies are conducted in Sect. 4. Finally, the conclusions and future work are given in Sect. 5.

2 Background

2.1 Problem Description

In WSLAP, a set of *user centres* $\mathcal{U} = \{U_1, \ldots, U_m\}$ and a set of *candidate locations* $\mathcal{A} = \{A_1, \ldots, A_n\}$ are given. A user centre indicates a centre city of a user-concentrated area. A candidate location is the geographic location that is suitable to deploy the Web services, e.g. the locations of the existing Web server hosting providers. There is a pool of Web services $\mathcal{W} = \{W_1, \ldots, W_s\}$. Each Web service $W_i \in \mathcal{W}$ must be deployed to at least one location. Note that a Web service can have multiple copies that are located in different locations. For each Web service $W_i \in \mathcal{W}$ and each candidate location $A_j \in \mathcal{A}$, there is a deployment cost C_{ij} induced by deploying service W_i at location A_j. For each user centre $U_k \in \mathcal{U}$ and each candidate location $A_j \in \mathcal{A}$, there is a *latency* L_{jk}, which affects the *response time* from the location A_j to the user centre U_k. It mainly depends on the distance between the two geographical locations. For each Web service $W_i \in \mathcal{W}$ and each user centre $U_k \in \mathcal{U}$, there is an *invocation frequency* F_{ik}, indicating the frequency of the service W_i invoked by the users from U_k. Given all the above information, the problem is to design a plan to deploy the services, so that each service is deployed in one or more locations, and the *total deployment cost* f_1 and *network latency* f_2 of the system is minimized. The deployment cost and network latency can be calculated as follows:

$$f_1 = \sum_{i=1}^{s} \sum_{j=1}^{n} C_{ij} x_{ij}, \tag{1}$$

$$f_2 = \sum_{i=1}^{s} \sum_{k=1}^{m} F_{ik} r_{ik}, \tag{2}$$

where x_{ij} takes 1 if service W_i is assigned in location A_j, and 0 otherwise. r_{ik} stands for the response time of service W_i to the user centre U_k, which is calculated as

$$r_{ik} = \min\{L_{jk} \mid j \in \{1, \ldots, n\} \text{ and } x_{ij} = 1\}. \tag{3}$$

Then, the problem can be stated as follows:

$$\min f_1 = \sum_{i=1}^{s} \sum_{j=1}^{n} C_{ij} x_{ij}, \tag{4}$$

$$\min f_2 = \sum_{i=1}^{s} \sum_{k=1}^{m} F_{ik} r_{ik}, \tag{5}$$

$$s.t. : \sum_{j=1}^{n} x_{ij} \geq 1, \ \forall\, i \in \{1, \ldots, s\}, \tag{6}$$

$$x_{ij} \in \{0, 1\}, \ \forall\, i \in \{1, \ldots, s\}, \ \forall\, j \in \{1, \ldots, n\}. \tag{7}$$

Eqs. (4) and (5) are the two objectives to be minimized. Eq. (6) indicates that each service must be deployed in at least one location. Eq. (7) gives the domain of the decision variables x_{ij}.

2.2 Related Work

Most of the previous work treated WSLAP as a single objective problem. In [1,22], integer linear programming techniques were used to solve the problem. In particular, the work in [22] solved the problem by employing greedy and linear relaxation.

Researches on network virtualization [2,8] employed greedy algorithms to allocate virtual machines (VMs) in the data center so that the requirements of network bandwidth are met. [14] presented a multi-layer and integrated fashion through a convex integer programming formulation.

The major drawback of greedy algorithm is that it is easy to be stuck at local optima. On the other hand, it is well known that integer linear programming has a high complexity and thus does not scale well. It can only be used in small or medium sized problem instances. In this case, heuristics and meta-heuristics such as genetic algorithms are promising to achieve better solutions within a short time.

Huang [9] proposed an enhanced Genetic Algorithm (GA)-based approach for the problem. However, only network latency was considered in the paper. Kessaci [12] proposed a new multi-objective genetic algorithm called MOGA-CB for minimizing the cost of VMs instance and response time. A framework called Green Monster was proposed in [19] to dynamically move Web services across Internet data centres for reducing their carbon footprint while maintaining their performance. Green monster applied a modified version of NSGA-II [7] with an additional local search process.

In summary, although there have been a number of works trying to solve WSLAP in different ways, no work exists to consider minimizing the deployment cost and network latency simultaneously. Therefore, in this paper, we formulate the multi-objective model and attempt to solve it with PSO.

3 Particle Swarm Optimization for Multi-Objective Web Service Location Allocation

PSO was proposed by Kennedy and Eberhart in 1995 [10]. It is a simple yet powerful optimization algorithm that mimics the flock behavior to search in the solution space. It has been successfully applied to various optimzation problems. Thus, we adopt the PSO framework to solve the MO-WSLAP in this paper. The generic PSO framework is given in Algorithm 1.

In line 4, the personal best location of each particle is the location with the best objective value that the particle found so far. In line 5, the global best location is the best location among the personal best locations of all the particles.

Algorithm 1: The generic framework of PSO

```
1  Randomly generate an initial swarm and the velocities;
2  repeat
3      foreach particle i in the swarm do
4          Update the personal best location p_i;
5          Update the global best location g;
6      end
7      foreach particle i in the swarm do
8          Update particle velocity v_i;
9          Update and evaluate particle location x_i;
10     end
11 until termination criterion is met;
12 return the global best g;
```

In line 8, the standard way of updating each dimension v_{id} of the velocity $\boldsymbol{v}_i$ is as follows:

$$v_{id} \leftarrow w \cdot v_{id} + c_1 \cdot r_{1i} \cdot (p_{id} - x_{id}) + c_2 \cdot r_{2i} \cdot (g_d - x_{id}), \tag{8}$$

where w is the inertia weight, c_1 and c_2 are the acceleration factors, and r_{1i} and r_{2i} are random variables sampled from uniform distribution between 0 and 1. v_{id}, x_{id}, p_{id} and g_d stand for the value in dimension d of $\boldsymbol{v}_i$, $\boldsymbol{x}_i$, $\boldsymbol{p}_i$ and $\boldsymbol{g}$ respectively. In MO-WSLAP, the decision variables x_{ij} are binary variables, i.e. they can only take 0 or 1. In this case, we employ the Binary PSO (BPSO) [11]. In line 9, the location is updated as follows:

$$x_{id} = \begin{cases} 1, & \text{if } rand() < \frac{1}{1+e^{-v_{id}}}, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

where $rand()$ is a value sampled from the uniform distribution between 0 and 1.

When solving MO-WSLAP, problem-specific solution representation, fitness evaluation and constraint handling must be designed. They will be described one by one in this section.

3.1 Solution Representation

In MO-WSLAP, the decision variables are x_{ij}, where $i = 1, \ldots, s$, and $j = 1, \ldots, n$. That is, the decision variables form a $s \times n$ matrix. Note that PSO was designed for vector-based solutions. Therefore, we simply adopt the flatten matrix representation that transforms the matrix $\boldsymbol{X}_{s \times n}$ into a $(s \times n)$–dimensional vector $\boldsymbol{y}$. The element x_{ij} in $\boldsymbol{X}$ corresponds to the $(n \cdot (i-1) + j)^{th}$ element in $\boldsymbol{y}$. For example, given a 3×3 matrix as follows:

$$\boldsymbol{X} = \begin{bmatrix} 0\ 1\ 0 \\ 0\ 0\ 1 \\ 1\ 0\ 0 \end{bmatrix},$$

the flatten matrix (vector) is $\boldsymbol{y} = (0, 1, 0, 0, 0, 1, 1, 0, 0)$. A $(s \times n)$–dimensional velocity vector $\boldsymbol{v}$ is defined accordingly, each for an element $y \in \boldsymbol{y}$. The vector $\boldsymbol{y}$ is used in the update phase. Then, during the fitness evaluation, $\boldsymbol{y}$ is first decoded into the original matrix $\boldsymbol{X}$.

3.2 Fitness Evaluation

After decoding $\boldsymbol{y}$ into the corresponding matrix $\boldsymbol{X}$, the total deployment cost and network latency can be directly calculated by Eqs. (1) and (2). In MO-WSLAP, since the two objectives are considered simultaneously, it is important to normalize them so that they have the same scale. To this end, the lower and upper bounds of both the total deployment cost and network latency are calculated. Specifically, for the total deployment cost, the lower bound $f_{1,\min}$ is obtained by deploying each Web service once in the location that leads to the minimal deployment cost, while the upper bound $f_{1,\max}$ is obtained by employing each Web service in all the locations. For the network latency, the lower bound $f_{2,\min}$ is achieved by deploying all the Web services in all the locations, and the upper bound $f_{2,\max}$ is obtained by an exhaustive search in which each Web service is allocated in only one location. The search space to find $f_{2,\max}$ is $s \times n$.

Based on the bounds of f_1 and f_2, the linear normalization is conducted to obtain the normalized objective values, which are denoted as $\hat{f}_1$ and $\hat{f}_2$.

There are various ways for fitness assignment in evolutionary multi-objective optimization [3,5,7,13,15,16]. In this paper, two different fitness assignment schemes are selected and compared. The first one is the weighted sum aggregating function, which is the most straightforward way that combines multiple objectives into a single one. The resultant PSO is called the Weighted Sum PSO (WSPSO). In WSPSO, the fitness function is defined as follows:

$$fitness = \lambda \cdot \hat{f}_1 + (1 - \lambda) \cdot \hat{f}_2. \tag{10}$$

That is, particle $\boldsymbol{y}_1$ is considered to be better than particle $\boldsymbol{y}_2$, if $fitness(\boldsymbol{y}_1) < fitness(\boldsymbol{y}_2)$. Note that the fitness value depends not only on $\hat{f}_1$ and $\hat{f}_2$, but also the weight coefficient λ. In the experimental studies, λ is simply set to 0.5 as a rule of thumb. That is, the two objectives are of the same importance. However, in practice this value can be obtained from service providers.

The second strategy is the dominance-based fitness assignment scheme, and the resultant PSO is called the Non-Dominated PSO (NSPSO) [15]. Given a set of objective functions, solution x_1 is said to *dominate* solution x_2, if (1) x_1 is no worse than x_2 in all the objectives, and (2) x_1 is better than x_2 in at least one objective. Based on the dominance relation, Deb et al. [7] designed a fast non-dominated sorting procedure, which is efficient in sorting a population of individuals from the best to the worst. The basic idea is to first divide the individuals into different fronts. The first front consists of all the individuals that are not dominated by any other individuals in the population. Then, each subsequent front includes the individuals that are dominated by no other individuals than those in the previous fronts. Within the same front, a *crowding*

distance measure is designed to measure the crowdedness around each individual. Then, the individual with a larger crowding distance is considered to be in a less crowded region, and thus be better. Details of the fast non-dominated sorting can be found in [7].

NSPSO adopts the fast non-dominated sorting in fitness assignment. The differences between NSPSO and the standard PSO framework are mainly twofold. First, in each generation, instead of replacing the original particle, the update of each particle creates a new particle. After all the particles have been updated, all the original particles and newly created particles are combined together and sorted by the fast non-dominated sorting. Then, a new swarm is formed by selecting the first particles in the sorted set. Second, instead of choosing the location with the best fitness value, the global best location is chosen to be the one in the first front and with the least crowding distance value. If there are multiple such locations, one is randomly selected.

3.3 Constraint Handling

In MO-WSLAP, each service must be allocated to at least one location. However, during the search process, the constraint can be violated, and there might exist some services that are not allocated to any location. That is, infeasible particle may occur. There are a variety of strategies to handle the constraints [4]. Here, we employ the simplest constraint handling approach, which is to ignore all the infeasible particles. To this end, all the infeasible particles are assigned the highest possible objective values (1 after normalization). Specifically,

$$\hat{f}_u(\boldsymbol{y}) = \begin{cases} \hat{f}_u(\boldsymbol{y}), & \text{if } \boldsymbol{y} \text{ is feasible,} \\ 1, & \text{otherwise.} \end{cases}, \ u = 1, 2. \tag{11}$$

where $\hat{f}_u(\boldsymbol{y})$ stands for the u^{th} normalized objective value.

4 Experimental Studies

To verify the efficacy of the proposed WSPSO and NSPSO, we tested them on the real-world network datasets, and compared with NSGA-II [7].

4.1 Datasets

The network dataset provided by WS-DREAM [23,24] is used in the experiments. The dataset includes 339 user centres and 5825 candidate locations, along with the latency matrix between the user centres and the candidate locations. From Sect. 2.1, it is known that apart from the latency matrix, a MO-WSLAP instance consists of the other two matrices, i.e. the deployment cost matrix $\boldsymbol{C} = (C_{ij})_{s\times n}$ and invocation frequency matrix $\boldsymbol{F} = (F_{ik})_{s\times m}$. The matrices were generated as follows.

Deployment Cost: The deployment cost can include the fixed deployment fees (e.g. monthly rent) and variable fees (e.g. extra charges for exceeded storage and other limits) [21]. For the sake of simplicity, here we only considered the fixed deployment fees, which is independent of the location of the service. For each service, the deployment cost was randomly generated from a normal distribution with mean of 100 and standard deviation of 20.

Invocation Frequency: For each user centre and each service, the invocation frequency was randomly generated from a uniform distribution between 1 and 120.

In the experiments, we randomly generated a number of services, and selected different subsets of the latency matrix to form a set of different problem instances. The features of the generated instances are given in Table 1.

Table 1. The features of the generated problem instances.

Instance	#Services	#Locations	#Centres
Instance 1	20	5	10
Instance 2	20	10	10
Instance 3	50	15	20
Instance 4	50	15	40
Instance 5	50	25	20
Instance 6	50	25	40
Instance 7	100	15	20
Instance 8	100	15	40
Instance 9	100	25	20
Instance 10	100	25	40
Instance 11	200	25	40
Instance 12	200	25	80
Instance 13	200	40	40
Instance 14	200	40	80

4.2 Experiment Settings

In both WSPSO and NSPSO, the population size was set to 50, and the maximal number of generations was set to 50. c_1 and c_2 were set equally to 2, and w was set to 1. In addition, NSGA-II was taken into account for comparison. NSGA-II uses the tournament selection to select parents, and single point crossover and flip mutation operators to generate offsprings. In NSGA-II, the population size and maximal number of generations was set the same as that of the PSO approaches to make a fair comparison. The crossover and mutation rates were set to 0.8 and 0.2 respectively. For each instance and each algorithm, 40 independent runs were conducted.

4.3 Performance Measures

The Hypervolume and Inverted Generational Distance (IGD) indicators were chosen as the performance measures. The hypervolume is the area of the region in the objective space dominated by the given set of solutions. A larger hypervolume value indicates a better solution set. The IGD value is defined as the average distance from a Pareto-optimal solution to the closest solution in the given solution set. A smaller IGD value indicates that the given solution set is closer to the true Pareto front, and thus is better. The optimal IGD value is 0. The performance measures were calculated based on the normalized objective values. For the hypervolume, the nadir point was set to $(1, 1)$. For the IGD, since the true Pareto front is unknown, we selected the non-dominated solutions among the final solutions obtained by all the runs of WSPSO, NSPSO and NSGA-II as the approximation of the true Pareto front.

4.4 Results and Discussions

Performance of WSPSO: First, we analyse the performance of WSPSO, since it can be seen as the basic PSO version for solving MO-SWLAP. Table 2 shows the mean and standard deviation of the hypervolume of WSPSO on the tested instances. From the table, one can see that as the problem size increases, the performance of WSPSO decreased (the hypervolume value dropped).

Table 2. The mean and standard deviation of the hypervolume of WSPSO on the tested instances.

Instance 1	**Instance 2**	**Instance 2**	**Instance 4**	**Instance 5**
0.89 ± 0.016	0.61 ± 0.01	0.69 ± 0.007	0.71 ± 0.008	0.67 ± 0.007
Instance 6	**Instance 7**	**Instance 8**	**Instance 9**	**Instance 10**
0.63 ± 0.005	0.63 ± 0.006	0.65 ± 0.008	0.62 ± 0.005	0.58 ± 0.005
Instance 11	**Instance 12**	**Instance 13**	**Instance 14**	
0.55 ± 0.003	0.56 ± 0.003	0.56 ± 0.004	0.57 ± 0.003	

Figure 1 shows the convergence curve of WSPSO on Instance 4. All the other instances showed a similar pattern. From the figure, it is clear that WSPSO converged well during the search process. More importantly, it seems that 50 generations is not sufficient for WSPSO to converge since the convergence curve is still dropping at generation 50. Note that MO-WSLAP is essentially a multi-objective optimization problem, and thus the weight sum fitness value is not comprehensive. Therefore, it is necessary to observe the temporal behavior of WSPSO in terms of both objectives rather than the aggregated fitness.

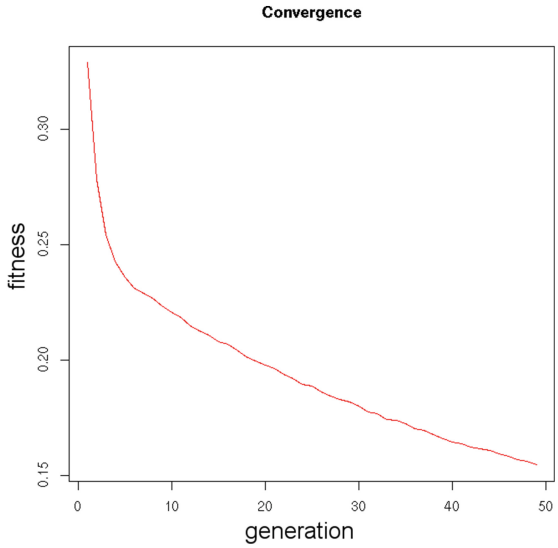


Fig. 1. The convergence curve (average fitness of the global best over 40 independent runs) of WSPSO on Instance 4.

Figure 2 shows the evolution trajectory of all the particles in the 2D objective space at different stages of the search process on Instance 4. From the figure, one can see that the particles in generation 50 do not dominate those in generation 10. Instead, they have a different trade-off between the total deployment cost and network latency. More specifically, from generation 10 to generation 50, the total deployment cost is reduced, while the network latency increases. It can be seen that as early as in generation 10, the network latency already achieved the best value (0.025 after normalization). In contrast, the total deployment cost is much harder to improve (around 0.5 after normalization). Thus, much more effort has been put in improving the total deployment cost with a slight sacrifice in the network latency. Therefore, the downward convergence curve in Fig. 1 does not indicate that the particles are improved in both objectives. Instead, the total deployment cost is improved and the network latency is deteriorated, but to less extent.

Finally, although WSPSO is a single-objective optimization algorithm, there are a swarm of particles maintained in the last generation, from which one may still obtain the non-dominated set. To observe the distribution of the particles in the final swarm, we draw the scatter plot of the particles in the last generation of one run in the objective space on Instance 4, as shown in Fig. 3. All the other instances showed a similar pattern. It can be seen that there is still a non-dominate set in the final swarm, and the best particle with the minimal fitness value is the one with the minimal total deployment cost. This partly shows the capability of WSPSO in obtaining a set of trade-off solutions for MO-WSLAP.

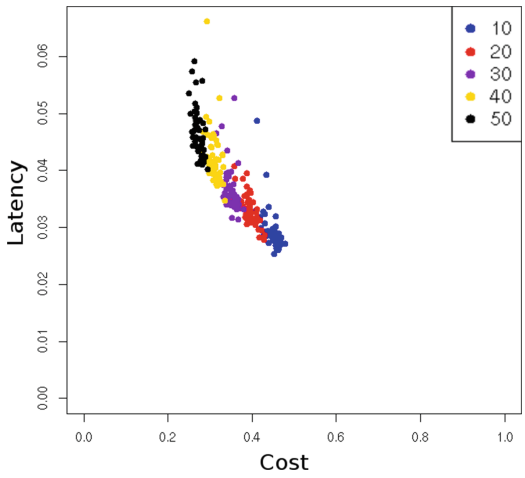


Fig. 2. The objective values of the particles at generation 10, 20, 30, 40 and 50 of WSPSO on Instance 4.

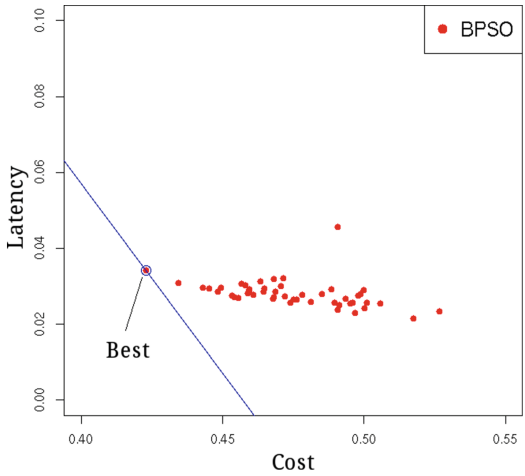


Fig. 3. The distribution of the final swarm in the objective space for Instance 4.

Performance of NSPSO: Then we evaluate the performance of NSPSO by comparing with WSPSO and NSGA-II. Tables 3 and 4 shows the mean and standard deviation of the hypervolume and IGD values of WSPSO, NSGA-II and NSPSO on the tested instances. The Wilxocon's rank sum test was conducted between each pair of the three compared algorithms with significance level of 0.05. For each instance, if one algorithm is significantly better than the other two, then the corresponding entry is marked in bold.

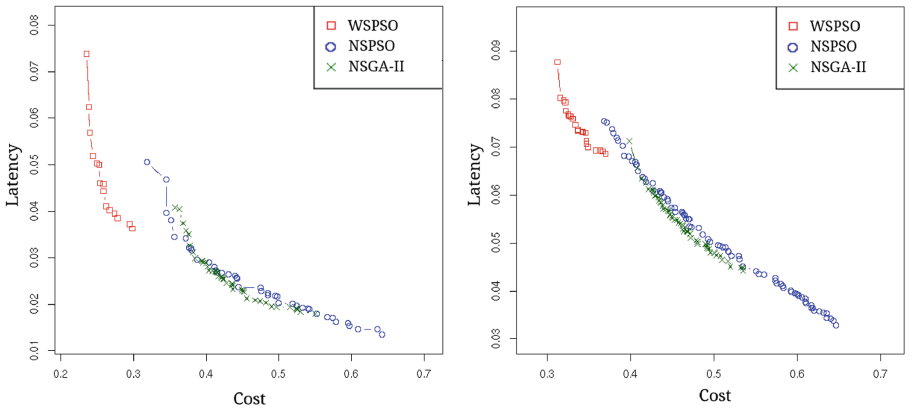
From the tables, one can see that in terms of hypervolume, WSPSO performed significantly the best on the first 10 instances. NSPSO performed signif-

Table 3. The mean and standard deviation of the hypervolume of WSPSO, NSGA-II and NSPSO on the tested instances. If an algorithm significantly outperform the other two algorithms with significance level of 0.05, then the corresponding entry is marked in bold.

Instance	WSPSO	NSGA-II	NSPSO
Instance 1	**0.895 ± 0.0132**	0.828 ± 0.0129	0.757 ± 0.0187
Instance 2	**0.615 ± 0.0105**	0.606 ± 0.0123	0.596 ± 0.0108
Instance 3	**0.690 ± 0.0072**	0.594 ± 0.0074	0.615 ± 0.0111
Instance 4	**0.713 ± 0.0077**	0.606 ± 0.0074	0.627 ± 0.0112
Instance 5	**0.668 ± 0.0077**	0.575 ± 0.0047	0.605 ± 0.0093
Instance 6	**0.626± 0.0049**	0.554 ± 0.0063	0.587 ± 0.0070
Instance 7	**0.633 ± 0.0060**	0.564 ± 0.0048	0.600 ± 0.0083
Instance 8	**0.652 ± 0.0070**	0.577 ± 0.0051	0.614 ± 0.0079
Instance 9	**0.620 ± 0.0052**	0.555 ± 0.0039	0.597 ± 0.0089
Instance 10	**0.585 ± 0.0049**	0.535 ± 0.0049	0.579 ± 0.0079
Instance 11	0.554 ± 0.0030	0.520 ± 0.0026	**0.571 ± 0.0087**
Instance 12	0.562 ± 0.0031	0.523 ± 0.0033	**0.578 ± 0.0094**
Instance 13	0.563 ± 0.0028	0.529 ± 0.0029	**0.585 ± 0.0077**
Instance 14	0.566 ± 0.0031	0.533 ± 0.0026	**0.588 ± 0.0089**

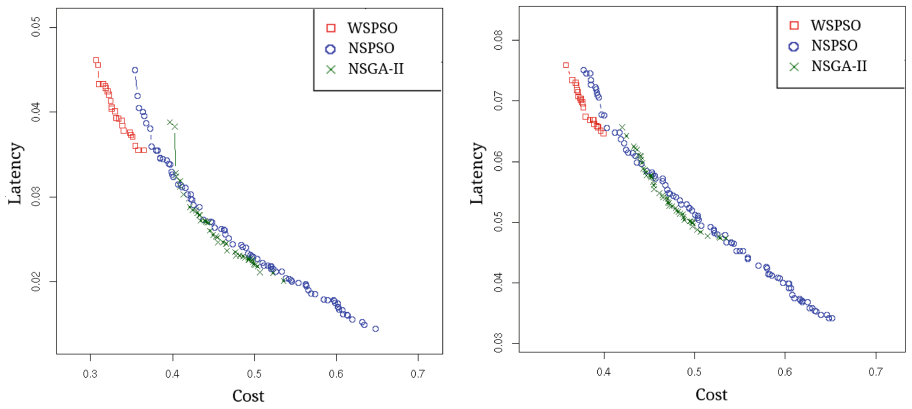
Table 4. The mean and standard deviation of the IGD of WSPSO, NSGA-II and NSPSO on the tested instances. If an algorithm significantly outperform the other two algorithms with significance level of 0.05, then the corresponding entry is marked in bold.

Instance	WSPSO	NSGA-II	NSPSO
Instance 1	0.194 ± 0.0176	**0.066 ± 0.0114**	0.092 ± 0.0110
Instance 2	0.146 ± 0.0131	**0.049 ± 0.0047**	0.057 ± 0.0035
Instance 3	0.146 ± 0.0089	0.036 ± 0.0029	**0.023 ± 0.0028**
Instance 4	0.133 ± 0.0089	0.038 ± 0.0028	**0.026 ± 0.0026**
Instance 5	0.110 ± 0.0060	0.037 ± 0.0030	**0.017 ± 0.0022**
Instance 6	0.123 ± 0.0068	0.036 ± 0.0040	**0.014 ± 0.0014**
Instance 7	0.128 ± 0.0091	0.037 ± 0.0036	**0.011 ± 0.0015**
Instance 8	0.114 ± 0.0068	0.041 ± 0.0036	**0.013 ± 0.0018**
Instance 9	0.100 ± 0.0045	0.039 ± 0.0025	**0.007 ± 0.0016**
Instance 10	0.104 ± 0.0046	0.041 ± 0.0037	**0.008 ± 0.0013**
Instance 11	0.077 ± 0.0038	0.036 ± 0.0018	**0.005 ± 0.0007**
Instance 12	0.074 ± 0.0033	0.035 ± 0.0021	**0.004 ± 0.0005**
Instance 13	0.076 ± 0.0027	0.044 ± 0.0024	**0.004 ± 0.0005**
Instance 14	0.069 ± 0.0023	0.039 ± 0.0022	**0.004 ± 0.0005**



(a) Instance 4 (b) Instance 6

Fig. 4. The best results of WSPSO, NSGA-II and NSPSO.



(a) Instance 8 (b) Instance 10

Fig. 5. The best results of WSPSO, NSGA-II and NSPSO.

icantly better on the large scale instances (from 11 to 14). In terms of IGD, it is clear that NSPSO obtained significantly better values than WSPSO and NSGA-II on 12 out of the total 14 instances (except the first two smallest instances, where NSGA-II performed better). Overall, NSPSO performed much better than the other two compared algorithms in terms of IGD. To better understand the inconsistency between the relative performance in terms of hypervolume and IGD, we plot the best results (the non-dominated solutions among those obtained in all the runs) of the compared algorithms in the objective space.

Figures 4, 5, and 6 show the results on Instances 4, 6, 8, 10, 12 and 14. From the figures, one can see that in general, NSPSO covered a much wider region than WSPSO and NSGA-II, especially when the problem size becomes larger. For the

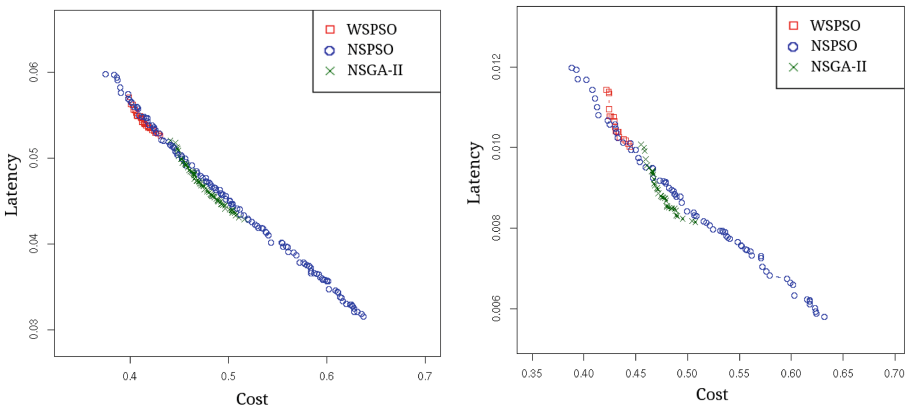


Fig. 6. The best results of WSPSO, NSGA-II and NSPSO.

small sized Instance 4, the advantage of NSPSO is not obvious, as it cannot reach the region of WSPSO, which has a smaller deployment cost. Nevertheless, NSPSO still obtained the smallest IGD value due to the better coverage. As the problem size increases, the advantage of NSPSO becomes more and more obvious. As shown in Fig. 5, on Instances 8 and 10, NSPSO nearly covered the entire area obtained by NSGA-II and the area that is slightly worse than that of WSPSO where the deployment cost is between 0.3 and 0.4. More importantly, NSPSO reached the area where the network latency is lower than 0.02 and 0.045 respectively, which was never reached by WSPSO or NSGA-II. As a result, the solutions obtained by NSPSO played a major role in the approximated Pareto front. Since the IGD value highly depends on the approximated Pareto front, it is not surprising that NSPSO obtained significantly better IGD value. The solutions obtained by WSPSO and NSGA-II covered different areas in the objective space, which are non-overlapping with each other. The results of NSGA-II had a better spread, and thus contributed more in the approximated Pareto front. Thus, NSGA-II obtained better IGD values than WSPSO. Fig. 6 shows a similar pattern as Fig. 5. In this figure, the advantage of NSPSO is more obvious, as it nearly covered the areas of both NSGA-II and WSPSO. Therefore, it achieved both significantly better hypervolume and IGD values.

In summary, we have the following major observations.

- Both the proposed WSPSO and NSPSO performed well in solving MO-WSLAP. WSPSO performed better in terms of hypervolume, and NSPSO performed better in terms of IGD;
- Both the proposed WSPSO and NSPSO performed better than NSGA-II, which is the most commonly used multi-objective optimization algorithms;
- In MO-WSLAP, the relative performances of the compared algorithms are inconsistent in hypervolume and IGD. This is mainly due to the different

optimization difficulties of the two objectives. Specifically, the total deployment cost is much harder to optimize than the network latency. As a result, the final solutions still have different scales in the two objectives after normalization. For example, as shown in Fig. 6b, for the final solutions, the deployment cost ranges from 0.4 to 0.6 and the network latency ranges from 0.006 to 0.012 on Instance 14. This makes the hypervolume emphasize more on the improvement in the deployment cost rather than the network latency.

5 Conclusions and Future Work

In this paper, the Multi-Objective Web Service Location Allocation Problem (MO-WSLAP) is investigated. In MO-WSLAP, the total deployment cost and network latency are to be minimized simultaneously. Two PSO variants are proposed for solving MO-WSLAP. One uses the weighted sum fitness evaluation, and the other uses the dominance-based fitness assignment. Both the proposed PSO methods performed better than the compared NSGA-II. Moreover, NSPSO performed better than WSPSO especially on larger instances. This demonstrates the efficacy of using dominance-based fitness assignment in solving real-world instances.

In the future, we plan to further improve the search scheme of PSO by employing more domain knowledge of the problem (e.g. the different optimization difficulties of the two objectives), and develop new PSO algorithms that can overcome the weaknesses of both WSPSO and NSPSO. We expect that the new PSO approach will be able to cover the areas of both WSPSO and NSPSO, and may reach a wider area in the objective space.

References

1. Aboolian, R., Sun, Y., Koehler, G.J.: A location allocation problem for a web services provider in a competitive market. Eur. J. Oper. Res. **194**(1), 64–77 (2009)
2. Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Towards predictable datacenter networks. In: ACM SIGCOMM (2011)
3. Coello, C., Pulido, G., Lechuga, M.: Handling multiple objectives with particle swarm optimization. IEEE Trans. Evol. Comput. **8**(3), 256–279 (2004)
4. Coello, C.A.C.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput. Methods Appl. Mech. Eng. **191**(11), 1245–1287 (2002)
5. Coello, C.A.C.: Evolutionary multi-objective optimization: a historical view of the field. IEEE Comput. Intell. Mag. **1**(1), 28–36 (2006)
6. Dan, A., Johnson, R.D., Carrato, T.: Soa service reuse by design. In: Proceedings of the 2nd International Workshop on Systems Development in SOA Environments, pp. 25–28. SDSOA 2008, ACM (2008)
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
8. Guo, C., Lu, G., Wang, H., Yang, S., Kong, C., Sun, P., Wu, W., Zhang, Y.: Secondnet: a data center network virtualization architecture with bandwidth guarantees. In: ACM CONEXT 2010. Association for Computing Machinery, Inc. (2010)

9. Huang, H., Ma, H., Zhang, M.: An enhanced genetic algorithm for web service location-allocation. In: Decker, H., Lhotská, L., Link, S., Spies, M., Wagner, R.R. (eds.) DEXA 2014, Part II. LNCS, vol. 8645, pp. 223–230. Springer, Heidelberg (2014)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
11. Kennedy, J., Eberhart, R.: A discrete binary version of the particle swarm algorithm. In: 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, vol. 5, pp. 4104–4108 (1997)
12. Kessaci, Y., Melab, N., Talbi, E.G.: A pareto-based genetic algorithm for optimized assignment of vm requests on a cloud brokering environment. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 2496–2503 (2013)
13. Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the pareto archived evolution strategy. Evol. Comput. **8**(2), 149–172 (2000)
14. Larumbe, F., Sanso, B.: Optimal location of data centers and software components in cloud computing network design. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 841–844 (2012)
15. Li, X.: A non-dominated sorting particle swarm optimizer for multiobjective optimization. In: Cantú-Paz, E., et al. (eds.) Genet. Evol. Comput. - GECCO 2003. LNCS, vol. 2723, pp. 37–48. Springer, Heidelberg (2003)
16. Mei, Y., Tang, K., Yao, X.: Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. IEEE Trans. Evol. Comput. **15**(2), 151–165 (2011)
17. Organization for the advancement of structured information standards (OASIS): Web Services Business Process Execution Language (WS-BPEL) Version 2.0 (2007)
18. Papazoglou, M.P., Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues. VLDB J. **16**(3), 389–415 (2007)
19. Phan, D.H., Suzuki, J., Carroll, R., Balasubramaniam, S., Donnelly, W., Botvich, D.: Evolutionary multiobjective optimization for green clouds. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 19–26. GECCO 2012, ACM (2012)
20. Ran, S.: A model for web services discovery with QoS. SIGecom Exch. **4**(1), 1–10 (2003)
21. Sun, Y.: A Location model for web services intermediaries. Ph.D. thesis, aAI3120151 (2003)
22. Sun, Y., Koehler, G.J.: A location model for a web service intermediary. Decis. Support Syst. **42**(1), 221–236 (2006)
23. Zhang, Y., Zheng, Z., Lyu, M.: Exploring latent features for memory-based QoS prediction in cloud computing. In: 2011 30th IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 1–10 (2011)
24. Zheng, Z., Zhang, Y., Lyu, M.: Distributed QoS evaluation for real-world web services. In: 2010 IEEE International Conference on Web Services (ICWS), pp. 83–90 (2010)
25. Zhou, J., Niemela, E.: Toward semantic QoS aware web services: issues, related studies and experience. In: IEEE/WIC/ACM International Conference on Web Intelligence. WI 2006, pp. 553–557 (2006)

Sim-EDA: A Multipopulation Estimation of Distribution Algorithm Based on Problem Similarity

Krzysztof Michalak(✉)

Department of Information Technologies, Institute of Business Informatics, Wroclaw University of Economics, Wroclaw, Poland
krzysztof.michalak@ue.wroc.pl

Abstract. In this paper a new estimation of distribution algorithm Sim-EDA is presented. This algorithm combines a multipopulation approach with distribution modelling. The proposed approach is to tackle several similar instances of the same optimization problem at once. Each subpopulation is assigned to a different instance and a migration mechanism is used for transferring information between the subpopulations. The migration process can be performed using one of the proposed strategies: two based on similarity between problem instances and one which migrates specimens between subpopulations with uniform probability. Similarity of problem instances is expressed numerically and the value of the similarity function is used for determining how likely a specimen is to migrate between two populations. The Sim-EDA algorithm is a general framework which can be used with various EDAs.

The presented algorithm has been tested on several instances of the Max-Cut and TSP problems using three different migration strategies and without migration. The results obtained in the experiments confirm, that the performance of the algorithm is improved when information is transferred between subpopulations assigned to similar instances of the problem. The migration strategy which transfers specimens between the most similar problem instances consistently produces better results than the algorithm without migration.

Keywords: Estimation of distribution algorithms · Multipopulation algorithms · Combinatorial optimization

1 Introduction

The algorithm presented in this paper combines a multipopulation approach based on the island model [2,27] with an estimation of distribution mechanism used in the PBIL algorithm [1] (additionally adapted in the case of the TSP). The multipopulation approach can be useful when the problem requires finding more than one good solution. Such a situation may occur in multimodal problems in which, in many areas of the search space, solutions can exist which

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 235–250, 2016.
DOI: 10.1007/978-3-319-30698-8_16

are equally good or almost equally good in terms of the objective function. In such a case it might be preferred, from the decision-making point of view, to maintain several good solutions corresponding to various areas of the search space, rather than to select just one that outperforms the others by possibly just a small value. Multimodal problems have been so far solved in the literature using methods such as species conservation [11], algorithms using the island model [2] and methods based on small-world topologies [8]. It is also possible to apply EDAs to multimodal problems, even though the authors of the paper [17] note "the poor performance of most EDAs for globally multimodal problem optimization". They propose to combine models able to encode conditional dependencies (namely Bayesian Networks) with niching to improve the performance of EDAs for this class of problems. However, contrary to this paper, they use a clustering algorithm for grouping solutions instead of multiple populations with migration. In the paper [5] EDAs are combined with the island model with migration, however, problem instance similarity is not used for controlling the migration. Instead, three different topologies are used (star, ring and broadcast) to determine which subpopulations receive the migrating specimens.

Similarly as in the case of multimodal problems, in multiobjective optimization it is usually not enough to obtain a single good solution. Because, typically, the objectives in a multiobjective optimization problem are conflicting it is not possible to find one solution with the best values of all of them. Instead, a common approach is to try to approximate an entire Pareto front of non-dominated solutions and report the possible choices to the decision maker. The paper [23] presents various ways of applying multipopulation algorithms to multiobjective optimization problems. Discussed approaches include cooperating subpopulations and the multi-start approach [9,12] in which independent instances of the optimization algorithm are started from different points or with different parameters.

Multipopulation algorithms are also often used for dynamic optimization problems. Because the environment changes, it is not desirable for the entire population to converge close to a single optimum. Multipopulation approaches allow some part of the population to improve the solution near the optimum while some part is left to explore other areas of the search space. A self-adaptive Differential Evolution algorithm named jDE [3], Shifting Balance GA (SBGA) [4], Forking Genetic Algorithms (FGAs) [24], and the multinational GA (MGA) [26] are among the methods that use the multipopulation approach for dynamic problems. Multipopulation approach has also been used with EDAs in the case of dynamic optimization problems by some authors [25,29].

In genetic algorithms a population of specimens which represent solutions of a given problem found by the algorithm is processed. These specimens undergo a process during which genetic operators are used to produce new specimens. The Estimation of Distribution Algorithms, on the other hand, work by modelling a probability distribution that describes properties of good solutions of the problem. Instead of applying genetic operators, the probabilistic model is updated based on the existing population, and then a new population is generated by sampling the updated model. In various EDAs different probabilistic models are

used, ranging from one-dimensional distributions as in UMDA [20,28] to decision trees [31], one- and multidimensional Gaussian models [7,30], Boltzmann Machines [21] and Bayesian Networks [18].

The algorithm proposed in this paper combines the multipopulation island model with an estimation of distribution algorithm to solve a set of similar problem instances using migration based on similarities between these instances. In this paper the Sim-EDA algorithm is applied to two optimization problems: the Max-Cut problem and the Travelling Salesman Problem. In a recent paper [22] the Max-Cut problem is used in image segmentation. Solving several problem instances at once would in this case mean performing image segmentation on a set of similar, but not identical images. In the case of the TSP solving several similar problem instances may be necessary for example when analyzing the influence of various changes in the network of roads on the length of transport routes. This task could be performed, for example, when simulating a situation in a city when certain streets become blocked or more crowded due to construction works. In a recent paper [19] the idea of solving many similar problems has been applied to a real world problem in genetics. In the aforementioned paper a structural transfer approach is used in which structural information is extracted and used to bias model construction. In this paper information is transferred in the form of solutions which partake in the migration process. The examples described above show that in certain areas there is a need for solving several similar optimization problem instances at once. Improving the optimization results using the Sim-EDA algorithm may be beneficial to optimization tasks in these areas.

The rest of this paper is structured as follows. Section 2 describes the Sim-EDA algorithm. Section 3 contains a description of the test problems used for experimental verification. In Sect. 4 parameters used for experiments are detailed and the results are presented. Section 5 concludes the paper.

2 Algorithm Description

The Sim-EDA algorithm proposed in this paper is based on the island model [27] used by some multipopulation genetic algorithms. Instead of genetic operations, however, it uses a probabilistic model and follows a cycle of operations typically implemented in EDAs in which, in each generation, the population is constructed using the probabilistic model, the specimens are evaluated and then the probabilistic model is updated. In the Sim-EDA algorithm, after the specimens are evaluated, but before the probabilistic model is updated the specimens are migrated between subpopulations. In this algorithm a separate subpopulation is assigned to each problem instance along with a separate probabilistic model. Therefore, the working of the algorithm can be viewed as a number of EDA algorithm instances each solving one of the optimization problem instances which exchange information through migration of specimens. This approach is different than in EDAs that use a mixture of models, such as [10] because from this mixture of models they generate a common population and in the Sim-EDA algorithm each model generates a different subpopulation. A mechanism

of elitism is used in the Sim-EDA algorithm which stores the best specimen b_k for each problem instance separately. This specimen is updated after a new population is generated and evaluated. In turn, the updated best specimen b_k is added to a newly generated population in the next generation.

2.1 Algorithm Details

An overview of the Sim-EDA algorithm is presented in Algorithm 1. Steps 1., 2.1. and 2.4. can be implemented in a way specific to the chosen EDA algorithm. EDAs with different probabilistic models, update procedures and sampling methods can be used depending on the solved problems. In this paper the PBIL algorithm [1] is used for the Max-Cut problem [15], because it is well suited for problems defined on a $\{0,1\}^n$ decision space. For the TSP a different probabilistic model is used, that models probabilities of different cities to be successors of each of the cities in good tours. This model is used with a PBIL model update procedure and with a procedure that generates new permutations based on the modelled probabilities of predecessor-successor pairs. For different problems, of course, different EDAs could be used.

The Sim-EDA algorithm uses the following procedures.

BinaryTournament - performs a binary tournament between two specimens and returns the winner. In the case of a single-objective problem this consists of simply comparing the two specimens' fitnesses. In the case of multiobjective problems the tournament can be based on the concept of Pareto domination.

CreatePopulation - generates a required number of specimens from the given probabilistic model.

Evaluate - evaluates specimens in a given population using a problem-specific goal function. Because during the migration phase specimens are evaluated with respect to a different problem than the one solved by their original population this method has an argument which defines the number of the problem instance to use for evaluation.

Improve - an optional step which performs an improvement or local search around specimens in a given population. Similarly as the evaluation procedure this procedure has an argument which defines the number of the problem instance to use for calculating the goal function.

InitializeModel - initializes a probabilistic model in a way specific to the chosen problem and the EDA algorithm.

SelectBestSpecimens - selects a given number of specimens which have the highest fitness function values from a given set.

SelectSourcePopulation - chooses a source subpopulation from which specimens will be migrated to subpopulation d using the problem similarity matrix S and a chosen migration strategy. Migration strategies are described in Sect. 2.2.

Algorithm 1. An overview of the Sim-EDA algorithm.

IN: N_{gen} - the number of generations
N_{pop} - the size of each subpopulation
N_{sub} - the number of subpopulations
N_{imig} - the number of migrated specimens

Calculate the matrix $S_{[N_{sub} \times N_{sub}]}$

// 1. Initialize probabilistic models and best specimens
for $d = 1, \ldots, N_{sub}$ **do**
  M_d = InitializeModel()
  $B_d = \emptyset$
end for

// 2. The main loop
for $g = 1, \ldots, N_{gen}$ **do**

  // 2.1. Generate new specimens
  for $d = 1, \ldots, N_{sub}$ **do**
    P_d = CreatePopulation(N_{pop}, M_d)
    Evaluate(P_d, d)
    Improve(P_d, d)
    $P_d = P_d \cup B_d$
    B_d = SelectBestSpecimens(P_d, 1)
  end for

  // 2.2. Select source populations
  for $d = 1, \ldots, N_{sub}$ **do**
    s = SelectSourcePopulation(S, d)
    P'_d = SelectBestSpecimens(P_s, N_{imig})
  end for

  // 2.3. Migrate specimens
  for $d = 1, \ldots, N_{sub}$ **do**
    Evaluate(P'_d, d)
    Improve(P'_d, d)
    for $x \in P'_d$ **do**
      w = the weakest specimen in P_d
      $P_d = P_d - \{w\}$
      b = BinaryTournament(w, x)
      $P_d = P_d \cup \{b\}$
    end for
  end for

  // 2.4. Update probabilistic models
  for $d = 1, \ldots, N_{sub}$ **do**
    M_d = UpdateModel(P_d, M_d)
  end for
end for

UpdateModel - updates the probabilistic model based on the current population. The implementation of this method depends on the chosen model and the representation of solutions used in specimens.

Details of the InitializeModel, CreatePopulation and UpdateModel procedures specific for test problems used in this paper are discussed in Sect. 3.

2.2 Migration

The Sim-EDA algorithm uses an island model [2,27] with migration. The subpopulations are assigned to different instances of an optimization problem and the migration process is controlled by a problem similarity matrix $S_{[N_{sub} \times N_{sub}]}$. Entries in the problem similarity matrix $S_{[N_{sub} \times N_{sub}]}$ should reflect how similar the problem instances are. Obviously, the way of calculating $S_{i,j}$ for a given pair of subproblems is specific to the problem representation. The two problems tackled in this paper both use a representation based on a weighted graph. Thus, i-th problem instance can be represented using a cost matrix $C^{(i)}_{[N \times N]}$ (where N is the problem size) that contains the lengths of the edges of the graph. The similarity between the i-th and the j-th problem instance can be calculated as a negated sum of squared differences between elements of the cost matrices:

$$S_{i,j} = -\sum_{p=1}^{N}\sum_{q=1}^{N}(C^{(i)}_{p,q} - C^{(j)}_{p,q})^2. \tag{1}$$

This approach to utilizing problem instance similarity has been successfully used in previous works of the author of this paper for solving the TSP [13] and the firefighter problem [14]. In both previous papers the multipopulation approach with migration was used with evolutionary algorithms, not EDAs.

Migration can be performed using various strategies. In this paper three migration strategies are tested and compared to the Sim-EDA without migration in which each subpopulation works independently on a separate problem instance. The latter is, of course, equivalent to simply executing N_{sub} separate algorithm runs for solving each problem instance in turn.

The migration strategies used in this paper are as follows:

- **Nearest** - a strategy in which N_{imig} specimens are migrated to a population P_d from such population P_s, $s \neq d$ that maximizes the value of $S_{d,s}$:

$$s = \underset{t \in \{1,\ldots,N_{sub}\}-\{d\}}{\operatorname{argmax}} (S_{d,t}) \tag{2}$$

- **Rank** - in this strategy all subpopulations except P_d are ranked in an increasing order according to values of $S_{d,t}$, $t \in \{1,\ldots,N_{sub}\} - \{d\}$ and the source population is selected randomly using the roulette wheel selection procedure with probabilities proportional to the ranks.
- **Uniform** - a strategy in which the source population number s is selected randomly with uniform probability from $\{1,\ldots,N_{sub}\} - \{d\}$.

– **None** - no migration is performed. Used for comparison with other strategies.

The two first migration strategies require calculating the similarity matrix $S_{[N_{sub} \times N_{sub}]}$ as a preprocessing step before the optimization starts.

3 Test Problems

The experiments described in this paper were performed on two test problems: the Max-Cut problem and the Travelling Salesman Problem (TSP). Both problems can be represented using a weighted graph $G = \langle V, E \rangle$, with $w : V \times V \to \mathbb{R}$ representing weights of edges connecting vertices from V. The representations of solutions differ, however, in these problems and also the probabilistic models used in Sim-EDA are different. Further, we will define the problem size as the number of vertices in the graph $N = |V|$.

3.1 The Max-Cut Problem

In the Max-Cut problem the goal is to find such a partition $V' \cup V'' = V$, $V' \cap V'' = \emptyset$ of the set of vertices of graph G for which the weight of the edges having one vertex in V' and the other in V'' is maximal. These edges would be the ones to be "cut" if we wanted to separate the graph G into two disjoint subgraphs with vertices from V' and V'' respectively.

Assume, that the weights in the graph are represented as a matrix $C_{[N \times N]}$ in which the element c_{ij} represents the weight of the edge between the i-th and the j-th vertex. Formally, the Max-Cut problem can be stated as:

$$\begin{aligned} &\text{maximize } f(V') = \sum_{i \in V', j \in (V - V')} c_{ij}, \\ &\text{subject to } V' \in 2^V, \end{aligned} \tag{3}$$

in which we try to find the best subset V' of the set of vertices V with respect to the goal function (3). The other set V'' is, of course, the complement of V'. Test instances of the size $N = 12, 25, 50, 100, 250$ and 500 were used in the experiments for the Max-Cut problem. In the Max-Cut problem a binary array $\{0, 1\}^N$ can easily be used for representing a solution with 0s corresponding to V' and 1s to V''. The probabilistic model $\mathbb{P}$ is also an N-element array $[0, 1]^N$ in which the elements belong to the range $[0, 1]$ and represent the probabilities of a corresponding entry having a value of 1 in good solutions of the problem. The procedure InitializeModel() sets $\mathbb{P} = [0.5, 0.5, \ldots, 0.5]$. A new population is created in the CreatePopulation() procedure by generating a given number of binary vectors of length N with i-th coordinate set independently of the others to 1 with probability p_i and to 0 with probability $1 - p_i$. The UpdateModel() procedure performs a standard probabilistic model update used in the PBIL algorithm, which updates the model using the best genotype $g^{(+)}$ and the worst one $g^{(-)}$ selected using their fitness from the current population. The model

update mechanism from PBIL is applied for each element p_i, $i = 1, \ldots, N$ in $\mathbb{P}$ separately. The parameters used in this process are two learning rate parameters: the positive learning rate η_+ and the negative learning rate η_-. Their sum is denoted $\eta = \eta_+ + \eta_-$. If $g_i^{(-)} = g_i^{(+)}$ the element p_i of $\mathbb{P}$ is set to:

$$p_i = p_i \cdot (1 - \eta_+) + g_i^{(+)} \cdot \eta_+, \tag{4}$$

and if $g_{ij}^{(-)} \neq g_{ij}^{(+)}$ the element p_i is set to:

$$p_i = p_i \cdot (1 - \eta) + g_i^{(+)} \cdot \eta. \tag{5}$$

After probability update, each element g_i (except the elements on the diagonal) is mutated with probability P_{mut} by setting:

$$p_i = p_i \cdot (1 - \mu) + \alpha * \mu, \tag{6}$$

where:

α - a 0 or 1 value drawn randomly with equal probabilities $P(0) = P(1) = \frac{1}{2}$,

μ - a mutation-shift parameter controlling the intensity of mutation.

3.2 The Travelling Salesman Problem (TSP)

A real-life motivation for the TSP arises from various transportation problems. The TSP requires visiting N cities in such a way that each of them is visited exactly once and the tour taken is possibly the shortest. Given a cost matrix $C_{[N \times N]}$ the TSP can be formulated as:

$$\begin{aligned} &\text{minimize } f(\pi) = C_{\pi(N)\pi(1)} + \sum_{i=1}^{N-1} C_{\pi(i)\pi(i+1)} \\ &\text{subject to } \pi \in \mathit{\Pi}(N) \end{aligned} \tag{7}$$

where $\mathit{\Pi}(N)$ is the set of all permutations of numbers $1, \ldots, N$. A solution can be represented as a permutation and consequently the genotype of a specimen is an N-element integer array containing the permutation. A different representation of solutions than in the Max-Cut problem requires a different probabilistic model. In the case of the TSP the probabilistic model used in Sim-EDA is a matrix $\mathbb{Q}_{[N \times N]}$ in which an element $\mathbb{Q}[i, j] \in [0, 1]$ represents the probability that the number j is placed right after i in a permutation that constitutes a good solution of the given TSP instance. This model is initialized by setting uniform probabilities for all pairs $i, j \in \{1, \ldots, N\}$ such that $i \neq j$:

$$\mathbb{Q}_{i,j} = \begin{cases} 0 & \text{for } i = j \\ \frac{1}{N-1} & \text{for } i \neq j \end{cases} \tag{8}$$

When a new specimen is created its genotype X is initialized as presented in Algorithm 2. This algorithm fills the genotype iteratively starting from a random

value for $X[1]$. For each $k = 2, \ldots, N$ denote a set of numbers not yet assigned to X by $U = \{1, \ldots, N\} - \{X[1], \ldots, X[k-1]\}$. Two situations are possible. If there are some nonzero elements in $\mathbb{Q}_{i,j}$ for $i = X[k-1]$ and $j \in U$ then select one of the numbers from U as $X[k]$ with probabilities proportional to the respective elements in $\mathbb{Q}$. If all elements in $\mathbb{Q}_{i,j}$ are zero for $i = X[k-1]$ and $j \in U$ then select one of the numbers from U with uniform probability. This ensures that whenever the probabilistic model $\mathbb{Q}$ indicates that there are possible successors to $X[k-1]$ one of them is selected using elements from $\mathbb{Q}$. If there is no successor to $X[k-1]$ with non-zero probability according to the probabilistic model $\mathbb{Q}$ then one of the yet unused numbers is selected with uniform probability.

Algorithm 2. Initialization of a genotype for a new specimen for the TSP based on the probabilistic model $\mathbb{Q}$.

IN: $\mathbb{Q}$ - the probabilistic model
N - genotype length

OUT: X - a new genotype

$X[1] = \text{UniformSelection}(\{1, \ldots, N\})$
for $k = 2, \ldots, N$ **do**
  $U = \{1, \ldots, N\} - \{X[1], \ldots, X[k-1]\}$
  $i = X[k-1]$
  $S = \sum_{j \in U} \mathbb{Q}[i, j]$
  if $S > 0$ **then**
    $X[k] = \text{RouletteWheelSelection}(U, \{\mathbb{Q}[i, j]\}_{j \in U})$
  else
    $X[k] = \text{UniformSelection}(U)$
  end if
end for

The RouletteWheelSelection(A, B) procedure selects elements from A with probabilities proportional to values in B. The UniformSelection(A) procedure selects elements from A with uniform probability.

The update of the probabilistic model $\mathbb{Q}$ is performed similarly as in the PBIL algorithm, but because the probabilistic model $\mathbb{Q}$ is a matrix and the solution representation in the TSP is a permutation an additional preprocessing step is required. Instead of updating the model directly from specimens S_{best} and S_{worst} the genotypes X_{best} and X_{worst} of these specimens are converted to matrices Q_{best} and Q_{worst} in which an element $Q_{i,j}$ is set to 1 if i and j are consecutive elements in X (including $X[N] = i$ and $X[1] = j$) and otherwise to 0. Clearly, such matrix represents a probability distribution in which consecutive pairs found in a given genotype are given probability 1 and all others probability 0. The matrices Q_{best} and Q_{worst} are used instead of binary vectors in the standard probability model update procedure used in the PBIL algorithm.

Since the mutation in PBIL works independently on each element of the probabilistic model $\mathbb{P}$ the application of the same procedure to $\mathbb{Q}$ is straightforward.

Test instances of the size $N = 12, 25, 50, 100, 150, 200$ and 250 were used in the experiments for the TSP.

3.3 Generating Similar Problem Instances

For the experiments N_{sub} similar instances of each problem for each problem size N were required. Each problem instance is represented as a cost matrix $C^{(d)}_{[N\times N]}$, for $d = 1, \ldots, N_{sub}$. The first cost matrix $C^{(1)}_{[N\times N]}$ was generated by drawing the elements above the diagonal from the uniform probability distribution $U[0, 100]$. To ensure the symmetry of the matrix the elements below the diagonal were initialized by copying the corresponding elements from above the diagonal. Each of the remaining cost matrices $C^{(d)}_{[N\times N]}$, for $d = 2, \ldots, N_{sub}$ was initialized by changing $1/N_{sub}$ (i.e. $1/20 = 5\,\%$) of the elements not placed on the diagonal in $C^{(d-1)}_{[N\times N]}$. The replaced elements were drawn from the uniform probability distribution $U[0, 100]$ and the symmetry of the new matrix was ensured by setting both corresponding elements above and below the diagonal to the new value. In all the cost matrices the elements on the diagonal were set to 0. The above procedure produces N_{sub} cost matrices $C^{(d)}_{[N\times N]}$, for $d = 1, \ldots, N_{sub}$ which differ less for closer values of the index d and differ more for values of the index d further apart.

4 Experiments and Results

The experiments described in this paper were aimed at verifying the effectiveness of the Sim-EDA algorithm in solving selected combinatorial problems. In particular, the goal was to investigate the influence of the migration procedure on the results of the optimization process. The Sim-EDA algorithm was run on several instances of two test problems: the Max-Cut problem and the Travelling Salesman Problem (TSP) using four migration strategies described in Sect. 2.2. For each test problem and each migration strategy $N_{run} = 30$ runs of the algorithm were performed to allow statistical comparison of the results.

The parameters of the algorithm were set as follows. The number of problem instances (and thus the number of subpopulations) was set to $N_{sub} = 20$. Each subpopulation size was set to $N_{pop} = 100$ for the Max-Cut problem and $N_{pop} = 1000$ for the TSP. The number of migrated specimens was set to $10\,\%$ of the population size, thus $N_{imig} = 10$ for the Max-Cut problem and $N_{imig} = 100$ for the TSP.

In this paper the PBIL algorithm with negative learning rate was used as the EDA in the Sim-EDA framework. Parameters of the algorithm were set following the original paper on the PBIL algorithm [1]. This algorithm uses two learning rate parameters which in this paper were set to: learnRate = 0.1 and

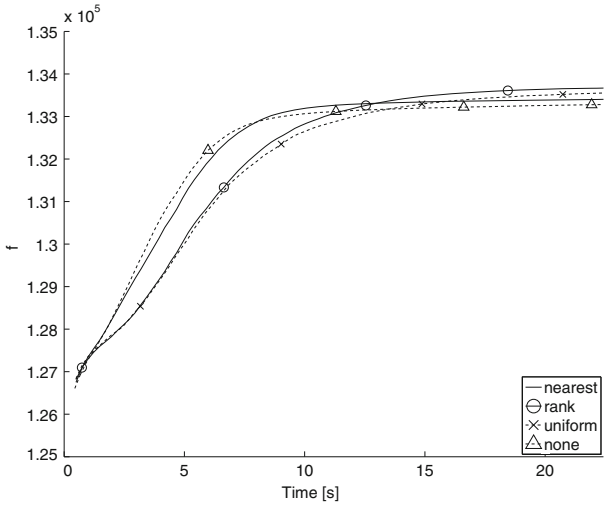


Fig. 1. Median results obtained for the Max-Cut problem with $N = 100$ nodes.

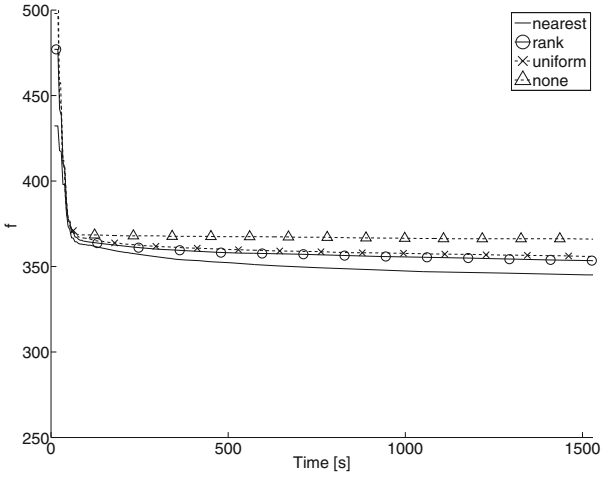


Fig. 2. Median results obtained for the TSP with $N = 100$ cities.

negLearnRate = 0.075. Two other parameters which control probabilities used in the mutation procedure were set to $P_{mut} = 0.02$ and $\mu = 0.05$.

The migration process used in the Sim-EDA algorithm requires some extra computation time for both the similarity matrix calculation as the preprocessing step and for additional computations during the migration phase in each generation. Therefore, results obtained with various migration strategies should not be compared with respect to the same number of generations, but with respect to time. In Figs. 1 and 2 examples of the performance of tested migration strategies with respect to time are presented.

Table 1. Results of the tests on the Max-Cut problem (higher values are better).

Instance size	Migration strategy	Mean	Median	Std. dev.	Comparison to "none"	
					p-value	Interp.
12	None	2052.2833	2043.2797	68.7213		
	Nearest	2052.7526	2043.2797	67.1400	0.43307	Insignificant
	Rank	2054.2401	2043.2797	66.5184	0.0003651	Significant
	Uniform	2053.6531	2043.2797	67.5602	0.0066871	Significant
25	None	8671.3856	8664.3626	136.6357		
	Nearest	8686.8558	8674.5851	132.2364	1.68E-05	Significant
	Rank	8701.5194	8686.9078	131.8232	2.00E-20	Significant
	Uniform	8702.0109	8689.4084	133.8095	9.00E-21	Significant
50	None	34169.0427	34157.2758	262.0070		
	Nearest	34213.7399	34183.5902	259.1844	3.61E-06	Significant
	Rank	34262.5991	34233.2313	302.8604	4.88E-21	Significant
	Uniform	34252.5286	34246.1427	292.6219	1.73E-19	Significant
100	None	133281.7688	133292.2493	505.6580		
	Nearest	133405.0093	133433.3314	509.5968	7.09E-08	Significant
	Rank	133486.5788	133651.4388	817.3963	4.58E-19	Significant
	Uniform	133195.4960	133514.7050	1114.0702	1.27E-03	Significant
250	None	817251.5645	817303.3203	1411.8478		
	Nearest	818132.9153	818057.8073	1488.0998	7.94E-26	Significant
	Rank	813343.0263	815448.3402	6016.6470	3.58E-37	Worse
	Uniform	809292.8215	809954.2165	7203.6887	3.48E-79	Worse
500	None	3214635.065	3214426.042	3388.4932		
	Nearest	3214979.768	3215141.950	3875.1273	4.89E-02	Significant
	Rank	3187810.589	3190216.098	18081.6794	7.56E-98	Worse
	Uniform	3175792.355	3168177.289	20508.3657	8.48E-98	Worse

For comparison of the migration strategies the following procedure has been adopted in this paper. Each strategy was allowed to run for $N_{gen} = 300$ generations. The average time t_0 in which the "none" strategy completed $N_{gen} = 200$ generations in $N_{run} = 30$ runs of the test was used as a reference point at which the results obtained by all the strategies were compared. Results for all tested strategies were recorded at time t_0, including any preprocessing time required by a given strategy and the time for migration. For each of the N_{sub} subpopulations in each of the $N_{run} = 30$ runs the best attained objective function value f_{best} was recorded. Tables 1 and 2 present the mean and median values calculated from the recorded f_{best} values.

For statistical verification of the results the Wilcoxon rank test was used. The Wilcoxon test was chosen because it does not assume the normality of the distributions which may be hard to guarantee in practical applications. This test was recommended in a survey [6] on methods suitable for evaluating evolutionary and swarm intelligence algorithms. In Tables 1 and 2 the column "p-value"

Table 2. Results of the tests on the TSP (lower values are better).

Instance size	Migration strategy	Mean	Median	Std. dev.	Comparison to "none"	
					p-value	Interp.
12	None	217.3898	216.1576	13.3413		
	Nearest	217.3898	216.1576	13.3413	1	Insignificant
	Rank	217.3898	216.1576	13.3413	1	Insignificant
	Uniform	217.3898	216.1576	13.3413	0.22656	Insignificant
25	None	164.4112	156.5974	27.472		
	Nearest	164.3935	156.5974	27.4952	0.38316	Insignificant
	Rank	164.3905	156.5974	27.4855	0.91015	Insignificant
	Uniform	164.3892	156.5974	27.486	0.40226	Insignificant
50	None	223.0045	220.3383	27.4457		
	Nearest	222.3719	219.5455	28.2546	0.088667	Insignificant
	Rank	224.3653	220.6636	27.8668	0.00072179	Worse
	Uniform	223.5734	219.0627	27.6526	0.30527	Insignificant
100	None	365.9890	364.7559	28.1933		
	Nearest	359.3402	360.5102	24.6559	4.2904E-18	Significant
	Rank	365.0730	364.5377	26.6054	0.012205	Significant
	Uniform	365.4464	364.6945	26.9252	0.21189	Insignificant
150	None	404.8644	403.0477	26.8652		
	Nearest	403.4235	401.3445	25.9356	1.6479e-008	Significant
	Rank	404.7463	402.9095	26.7110	0.03125	Significant
	Uniform	404.7797	402.9095	26.7887	0.125	Insignificant
200	None	456.2797	456.6390	23.8788		
	Nearest	455.8308	456.2452	23.5170	6.1035e-005	Significant
	Rank	456.2793	456.6390	23.8788	1	Insignificant
	Uniform	456.1204	456.5193	23.7636	0.0625	Insignificant
250	None	510.5612	510.1462	24.3592		
	Nearest	510.3769	510.0111	24.1334	0.0078125	Significant
	Rank	510.5425	510.1462	24.3338	1	Insignificant
	Uniform	510.5477	510.1462	24.3529	0.5	Insignificant

contains the p-value obtained by the Wilcoxon test for a null hypothesis that the migration strategy corresponding to a given row produces results with the same median as the "none" strategy. A low p-value indicates that it is statistically unlikely that the medians are equal. The column "interp." contains the interpretation of the p-value which is determined in the following way. If the median obtained by a given strategy is lower for the Max-Cut problem or higher for the TSP than that produced by the "none" strategy the interpretation is "worse". In the opposite case the interpretation depends on the result of the statistical test. If the obtained p-value is >0.05 the interpretation is "insignificant" and if the obtained p-value is ≤ 0.05 the interpretation is "significant".

In the case of the Max-Cut problem the strategies using migration outperformed the one not using migration for all small instances ($N \leq 100$). With

one exception (the "nearest" strategy for $N = 12$) this advantage is statistically significant. For large instances ($N = 250$ and 500) only the "nearest" strategy outperformed the strategy without migration, but this advantage was found to be statistically significant. Overall, the "nearest" strategy is the one that consistently outperformed the "none" strategy on all tested instances of the Max-Cut problem. With exception to the smallest problem instance ($N = 12$) the advantage of the "nearest" migration strategy is statistically significant.

For small instances of the TSP ($N = 12$ and 25) all the strategies with migration outperformed the "none" strategy, however, no statistical significance could be confirmed (cf. Table 2). For large instances ($N \geq 100$) the "nearest" strategy consistently produced significantly better results than those obtained when no migration was used. Overall, all the strategies with migration outperformed the "none" strategy in all cases except the "rank" strategy for $N = 50$. Not in all cases, however, the difference in results was large enough to allow stating statistical significance.

In summary, strategies with migration outperformed the "none" strategy in most cases. Especially, the "nearest" migration strategy consistently produced better results than those obtained without migration.

5 Conclusion

This paper presents an algorithm combining the Estimation of Distribution Algorithms with an island model. The proposed algorithm was designed with an intent of solving several instances of an optimization problem at once. In the case of the Max-Cut problem which can be applied to image segmentation [22] the proposed approach would allow processing several similar images at once. For other problems the proposed approach may be useful for simulating the effects of various changes in the input data. For example in the case of the TSP it is possible to determine how the changes in the graph of available routes will influence the length of the tour.

The proposed approach was tested on the Max-Cut problem instances of the size $N = 12, 25, 50, 100, 250$ and 500 and on the TSP instances of the size $N = 12, 25, 50, 100, 150, 200$ and 250. Strategies using migration outperformed the strategy without migration in most cases. In particular, the strategy of migrating specimens solely from the nearest subpopulation consistently produced better results than those obtained when no migration was used.

In this paper the algorithm was tested on synthetic data and the focus was on determining if the migration improves the performance of the algorithm and how effective various migration strategies are. Application to real-life problems has been left for a further work. Another interesting topic for further work is to use migration of probabilistic models instead of specimens. This approach has been shown to be effective in the paper [5], so it seems promising to develop methods combining probabilistic models migration with problem instance similarity measures. Another interesting idea comes from the field of study on the parameterized complexity [16], where machine learning methods are used to predict

the difficulty of a given problem instance based on the features of this instance. A similar approach could be used in Sim-EDA to decide, based on features of the instances of the optimization problem, between which subpopulations to migrate solutions.

References

1. Baluja, S.: Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning. Technical report, Pittsburgh (1994)
2. Bessaou, M., Petrowski, A., Siarry, P.: Island model cooperating with speciation for multimodal optimization. In: Schoenauer, M., et al. (eds.) Parallel Problem Solving from Nature PPSN VI. LNCS, vol. 1917, pp. 437–446. Springer, Heidelberg (2000)
3. Brest, J., et al.: Dynamic optimization using self-adaptive differential evolution. In: IEEE Congress on Evolutionary Computation, pp. 415–422. IEEE (2009)
4. Chen, J., Wineberg, M.: Enhancement of the shifting balance genetic algorithm for highly multimodal problems. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, pp. 744–751. IEEE Press, Portland (2004)
5. delaOssa, L., Gámez, J.A., Puerta, J.M.: Migration of probability models instead of individuals: an alternative when applying the island model to EDAs. In: Yao, X. (ed.) PPSN 2004. LNCS, vol. 3242, pp. 242–252. Springer, Heidelberg (2004)
6. Derrac, J., et al.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol. Comput. **1**(1), 3–18 (2011)
7. Dong, W., Yao, X.: Unified eigen analysis on multivariate Gaussian based estimation of distribution algorithms. Inf. Sci. **178**(15), 3000–3023 (2008)
8. Giacobini et al., M., Preuß, M., Tomassini, M.: Effects of scale-free and small-world topologies on binary coded self-adaptive CEA. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 86–98. Springer, Heidelberg (2006)
9. Jozefowiez, N., Semet, F., Talbi, E.G.: Target aiming pareto search and its application to the vehicle routing problem with route balancing. J. Heuristics **13**(5), 455–469 (2007)
10. Lancucki, A., Chorowski, J., Michalak, K., Filipiak, P., Lipinski, P.: Continuous population-based incremental learning with mixture probability modeling for dynamic optimization problems. In: Corchado, E., Lozano, J.A., Quintián, H., Yin, H. (eds.) IDEAL 2014. LNCS, vol. 8669, pp. 457–464. Springer, Heidelberg (2014)
11. Li, J.P., et al.: A species conserving genetic algorithm for multimodal function optimization. Evol. Comput. **10**(3), 207–234 (2002)
12. Mezmaz, M.S., Melab, N., Talbi, E.: Using the multi-start and island models for parallel multi-objective optimization on the computational grid. In: Second IEEE International Conference on e-Science and Grid Computing, 2006. e-Science 2006, pp. 112–120 (2006)
13. Michalak, K.: Sim-EA: an evolutionary algorithm based on problem similarity. In: Corchado, E., Lozano, J.A., Quintián, H., Yin, H. (eds.) IDEAL 2014. LNCS, vol. 8669, pp. 191–198. Springer, Heidelberg (2014)
14. Michalak, K.: The Sim-EA algorithm with operator autoadaptation for the multiobjective firefighter problem. In: Ochoa, G., Chicano, F. (eds.) EvoCOP 2015. LNCS, vol. 9026, pp. 184–196. Springer, Heidelberg (2015)

15. Newman, A.: Max cut. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms, pp. 1–99. Springer, New York (2008)
16. Nudelman, E., Leyton-Brown, K., H. Hoos, H., Devkar, A., Shoham, Y.: Understanding random SAT: beyond the clauses-to-variables ratio. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 438–452. Springer, Heidelberg (2004)
17. Peña, J.M., Lozano, J.A., Larrañaga, P.: Globally multimodal problem optimization via an estimation of distribution algorithm based on unsupervised learning of Bayesian Networks. Evol. Comput. **13**(1), 43–66 (2005)
18. Pelikan, M., Goldberg, D.E.: Hierarchical problem solving by the Bayesian optimization algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference 2000, pp. 267–274. Morgan Kaufmann (2000)
19. Santana, R., Mendiburu, A., Lozano, J.: Structural transfer using EDAs: an application to multi-marker tagging SNP selection. In: 2012 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2012)
20. Santana, R., Larranaga, P., Lozano, J.A.: Side chain placement using estimation of distribution algorithms. Artif. Intell. Med. **39**(1), 49–63 (2007)
21. Shim, V.A., et al.: Enhancing the scalability of multi-objective optimization via restricted Boltzmann machine-based estimation of distribution algorithm. Inf. Sci. **248**, 191–213 (2013)
22. de Sousa, S., Haxhimusa, Y., Kropatsch, W.G.: Estimation of distribution algorithm for the Max-Cut problem. In: Kropatsch, W.G., Artner, N.M., Haxhimusa, Y., Jiang, X. (eds.) GbRPR 2013. LNCS, vol. 7877, pp. 244–253. Springer, Heidelberg (2013)
23. Talbi, E.-G., Mostaghim, S., Okabe, T., Ishibuchi, H., Rudolph, G., Coello Coello, C.A.: Parallel approaches for multiobjective optimization. In: Branke, J., Deb, K., Miettinen, K., Słowiński, R. (eds.) Multiobjective Optimization. LNCS, vol. 5252, pp. 349–372. Springer, Heidelberg (2008)
24. Tsutsui, S., Fujimoto, Y., Ghosh, A.: Forking genetic algorithms: gas with search space division schemes. Evol. Comput. **5**(1), 61–80 (1997)
25. Uludag, G., et al.: A hybrid multi-population framework for dynamic environments combining online and offline learning. Soft Comput. **17**(12), 2327–2348 (2013)
26. Ursem, R.K.: Multinational GA optimization techniques in dynamic environments. In: Whitley, D., et al. (ed.) Genetic and Evolutionary Computation Conference, pp. 19–26. Morgan Kaufmann(2000)
27. Whitley, D., Rana, S., Heckendorn, R.B.: The island model genetic algorithm: on separability, population size and convergence. J. Comput. Inf. Technol. **7**, 33–47 (1998)
28. Yan, W., Xiaoxiong, L.: An improved univariate marginal distribution algorithm for dynamic optimization problem. AASRI Procedia **1**, 166–170 (2012). AASRI Conference on Computational Intelligence and Bioinformatics
29. Yang, S., Yao, X.: Dual population-based incremental learning for problem optimization in dynamic environments. In: Proceedings of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems, pp. 49–56 (2003)
30. Yuan, B., Orlowska, M., Sadiq, S.: Extending a class of continuous estimation of distribution algorithms to dynamic problems. Optim. Lett. **2**(3), 433–443 (2008)
31. Zhong, X., Li, W.: A decision-tree-based multi-objective estimation of distribution algorithm. In: 2007 International Conference on Computational Intelligence and Security, pp. 114–118 (2007)

Solving the Quadratic Assignment Problem with Cooperative Parallel Extremal Optimization

Danny Munera[1], Daniel Diaz[1], and Salvador Abreu[1,2(✉)]

[1] University of Paris 1-Sorbonne/CRI, Paris, France
salvador.abreu@acm.org
[2] Universidade de Évora/LISP, Évora, Portugal

Abstract. Several real-life applications can be stated in terms of the Quadratic Assignment Problem. Finding an optimal assignment is computationally very difficult, for many useful instances. We address this problem using a local search technique, based on Extremal Optimization and present experimental evidence that this approach is competitive. Moreover, cooperative parallel versions of our solver improve performance so much that large and hard instances can be solved quickly.

Keywords: QAP · Extremal optimization · Heuristics · Parallelism · Cooperation

1 Introduction

The Quadratic Assignment Problem (QAP) was introduced in 1957 by Koopmans and Beckmann [1] as a model of a facilities location problem. This problem consists in assigning a set of n facilities to a set of n specific locations minimizing the cost associated with the *flows* of items among facilities and the *distance* between them. This combinatorial optimization problem has many other real-life applications: scheduling, electronic chipset layout and wiring, process communications, turbine runner balancing, data center network topology, to cite but a few [2,3]. Unfortunately this problem is known to be NP-hard and finding efficient algorithms to solve it has attracted a lot of research in recent years.

Exact (or complete) methods like dynamic programming, cutting plane techniques and branch &bound procedures have been successfully applied to medium-size QAP instances but cannot solve larger instances (e.g. when $n > 20$). To tackle these problems, one must resort to incomplete methods which are designed to quickly provide good, albeit sub-optimal, solutions. This is the case of *approximation algorithms*, i.e. algorithms running in polynomial time yet able to guarantee solutions within a constant factor of the optimum. Unfortunately, it is known that there is no ϵ-approximation algorithm for the QAP [4]. Another class of incomplete methods is provided by meta-heuristics. Since 1990 several meta-heuristics have been successfully applied to the QAP: tabu search, simulated annealing, genetic algorithms, GRASP, ant-colonies [3]. The current trend

F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 251–266, 2016.
DOI: 10.1007/978-3-319-30698-8_17

is to specialize existing heuristics, to compose different meta-heuristics (hybrid procedures) and to use parallelism.

In this paper we propose EO-QAP: an Extremal Optimization (EO) procedure for QAP. EO is a nature-inspired general-purpose meta-heuristics to solve combinatorial optimization problems [5]. This local search procedure, a priori, has several advantages: it is easy to implement, it does not get confounded by local minima and takes only one adjustable parameter. We experimentally demonstrate that EO-QAP performs well on the set of QAPLIB benchmark instances. It is, however, known that it is difficult with EO to have fine control on the trade-off between search *intensification* and *diversification*: some strategies have been proposed to overcome this limitation [6], but they entail a more complex tuning process. In this paper we put forth two other approaches which contribute to a more effective handling of QAP using EO: firstly, we propose a simple extension to the original EO which allows the user to have more control over the stochastic behavior of the algorithm. Secondly, we propose to use *cooperative parallelism* to promote more intensification and/or diversification. Our implementation uses a parallel framework [7] written in X10 [8,9]. We show that the cooperative parallel version behaves very well on the hardest instances.

The rest of the paper is organized as follows. Section 2 discusses QAP and provides the necessary background. Section 3 presents our EO algorithm for QAP and proposes an extension to the original EO. Several experimental results are laid out and discussed in Sect. 4 and we conclude in Sect. 5.

2 Background

Before introducing the main object of this paper, we need to recall some background topics: the Quadratic Assignment Problem (QAP), Extremal Optimization (EO) and Cooperative Parallel Local Search (CPLS).

2.1 QAP

Since its introduction in 1957, QAP has been widely studied and several surveys are available [2,3,10,11].

A QAP problem of size n consists of two $n \times n$ matrices (a_{ij}) and (b_{ij}). Let $\Pi(n)$ be the set of all permutations of $\{1, 2, \ldots n\}$, the goal of QAP is to find a permutation $\pi \in \Pi(n)$ which minimizes the following objective function:

$$F(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{\pi_i \pi_j} \tag{1}$$

For instance, in facility location problems, the a matrix represents inter-facility flows and b encodes the inter-location distances. In that context, both matrices are generally *symmetric*: $\forall_{i,j}\ a_{ij} = a_{ji}$ and $b_{ij} = b_{ji}$. However, in other settings the matrices can become asymmetric. Indeed, QAP can be used to model scheduling, chip placement and wiring on a circuit board, to design

typewriter keyboards, for process communications, for turbine runner balancing among many other applications [2,12].

The computational difficulty of QAP stems form the fact that the objective function contains products of variables (hence the term quadratic) and in the fact that the theoretical search space of an instance of size n is the set of all permutations $\Pi(n)$ whose cardinality is $n!$. In 1976, Sahni and Gonzalez proved that QAP is NP-hard [4] (the famous traveling salesman problem can be formulated as a QAP). Moreover, the same authors proved that there is no ϵ-approximation algorithm for QAP (unless P=NP). In practice QAP is one of the most difficult combinatorial optimization problems with many real-life applications.

QAP can be (optimally) solved with exact methods like dynamic programming, cutting plane techniques and branch & bound algorithms (together with efficient lower bound methods). Constraint Programming does not work well on QAP and, surprisingly, SAT solvers have not been extensively used for QAP. However, general problems of medium size (e.g. $n > 20$) are out of reach for these methods (even if some particular larger instances can be solved). It is thus natural to use heuristics to solve QAP. In the last decades several metaheuristics were successfully applied to QAP: tabu search, simulated annealing, genetic algorithms, GRASP, ant-colonies [13]. In this paper we propose to use *Extremal Optimization* to attack QAP problems.

2.2 Extremal Optimization

In 1999, Boettcher and Percus proposed the Extremal Optimization (EO) procedure [5,14,15] as a meta-heuristics to solve combinatorial optimization problems. EO is inspired by self-organizing processes often found in nature. It based on the concept of *Self-Organized Criticality* (SOC) initially proposed by Bak [16,17] and in particular by the Bak-Sneppen model of SOC [18]. In this model of biological evolution, *species* have a *fitness* $\in$ [0,1] (0 representing the worst degree of adaptation). At each iteration, the species with the worst fitness value is updated, i.e. its fitness is replaced by a new random value. This change also affects all other species connected to this "culprit" element and their fitness value also gets updated. This results in an *extremal* process which progressively eliminates the least fit species (or forces them to mutate). Repeating this process eventually leads to a state where all species have a good fitness value, i.e. a SOC. EO follows this line: it inspects the current *configuration* (assignment of variables), selects the worst variable (the one having the lowest fitness) and replaces its value by a random value. However, always selecting the worst variable can lead to a deterministic behavior and the algorithm can stay blocked in a local minimum. To avoid this, the authors propose an extended algorithm which first ranks the variables in increasing order of fitness (the worst variable has thus a rank $k = 1$) and then resorts to a *Probability Distribution Function* (PDF) over the ranks k in order to introduce uncertainty in the search process:

$$P(k) = k^{-\tau} \qquad (1 \leq k \leq n) \tag{2}$$

This power-law probability distribution takes a single parameter τ which is problem-dependent. Depending on the value of τ, EO provides a wide variety of search strategies from pure random walk ($\tau = 0$) to deterministic (greedy) search ($\tau \to \infty$). With an adequate value for τ, EO cannot be trapped in local minima since any variable is susceptible to mutate (even if the worst are privileged). This parameter can be tuned by the user. Moreover, the original paper proposes a default value depending on n: $\tau = 1 + \frac{1}{ln(n)}$.

EO displays several a priori advantages: it is a simple meta-heuristic (it can be easily programmed), it is controlled by only one free parameter (a fine tuning of several parameters becomes quickly tedious) and it does not need to be aware about local minima. Nevertheless, EO has been successfully applied to large-scale optimization problems like graph bi-partitioning, graph coloring, Spin Glasses or the traveling salesman problem [14]. Boettcher and Percus point out, however, that depending on the problem, *"a drawback of the EO method is that a general definition of fitness for the individual variables may prove ambiguous or even impossible"* [19]. To overcome this, De Sousa and Ramos proposed an extension called Generalized Extremal Optimization [20,21]. Zhou and al. proposed a variant called Continuous Extremal Optimization to deal with continuous optimization problems [22]. It has been also argued that one main issue with EO is that is does not provide a fine control of the intensification. Randall and Lewis propose some intensification strategies to improve EO [6]. We present two alternatives to overcome this limitation: we propose a simple extension to improve the stochastic capabilities of EO and we show how cooperative parallelism can help to achieve intensification or diversification through communications.

2.3 Cooperative Parallel Local Search

Parallel local search methods have been proposed in the past [23–25]. In this article we are interested in *multi-walk* methods (also called *multi-start*) which consist in a concurrent exploration of the search space, either *independently* or *cooperatively* via communication between processes. The *Independent Multi-Walks* method (IW) [26] is the easiest to implement since the solver instances do not communicate with each other. However, the resulting gain tends to flatten when scaling over a hundred of processors [27], and can be improved upon. In the *Cooperative Multi-Walks* (CW) method [28], the solver instances exchange information (through communication), hoping to hasten the search process. However, implementing an efficient cooperative method is a very complex task: several choices have to be made about the communication which influence each other and which are problem-dependent [28].

We build on the framework for Cooperative Parallel Local Search (CPLS) proposed in [7,29]. This framework, available as an open source library in the X10 programming language, allows the programmer to tune the search process through an extensive set of parameters which, in the present version, statically condition the execution. CPLS augments the IW strategy with a tunable communication mechanism, which allows for the cooperation between the multiple instances to seek either an intensification or diversification strategy for the

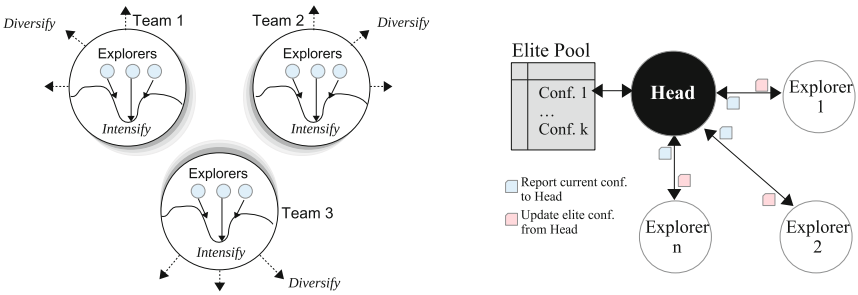


Fig. 1. CPLS framework structure

search. At present, the tuning process is done manually: we have not yet experimented with parameter self-adaptation (still an experimental feature).

The basic component of CPLS is the *explorer node* which consists in a local search solver instance. The point is to use all the available processing units by mapping each *explorer node* to a physical core. Explorer nodes are grouped into *teams*, of size NPT (see Fig. 1). This parameter is directly related to the trade-off between intensification and diversification. NPT can take values from 1 to the maximum number of nodes (frequently linked to maximum number of available cores in the execution). When NPT is equal to 1, the framework coincides with the IW strategy, it is expected that each 1-node team be working on a different region of the search space, without seek parallel intensification. When NPT is equal to the maximum number of nodes (creating only 1 team in the execution), the framework has the maximum level of parallel intensification, but it is not possible to enable parallel diversification between teams.

Each team seeks to *intensify* the search in the most promising neighborhood found by any of its members. The parameters which guide the intensification are the *Report Interval* (R) and *Update Interval* (U): every R iterations, each explorer node sends its current configuration and the associated cost to its *head node*. The head node is the team member which collects and processes this information, retaining the best configurations in an *Elite Pool* (EP) whose size $|EP|$ is parametric. Every U iterations, explorer nodes randomly retrieve a configuration from the EP, in the head node. An explorer node may *adopt* the configuration from the EP, if it is "better" than its own current configuration with a probability *pAdopt*. Simultaneously, the teams implement a mechanism to cooperatively *diversify* the search, i.e. they try to extend the search to different regions of the search space.

Typically, each problem benefits from intensification and diversification on some level. Therefore, the tuning process of the CPLS parameters seeks to provide the appropriate balance between the use of the intensification and diversification mechanisms, in hope of reaching better performance than the non-cooperative parallel solvers (e.g. Independent Multi-Walks). A detailed description of this framework may be found in [7].

3 EO-QAP: An EO Procedure for QAP

3.1 General Procedure

Our EO-QAP algorithm starts from a random permutation $\pi \in \Pi(n)$, with an associated cost given by $F(\pi)$. To ensure we only consider proper permutations, we only perform *swap* operations on pairs of elements from any given one: this is how value assignment is classically implemented in permutation problems, thereby eschewing the costly explicitly encoding of an `all-different` constraint. We define the permutation resulting from swapping π_i and π_j:

$$\pi_{i \leftrightarrow j} = \mu \mid \mu_i = \pi_j, \mu_j = \pi_i, \mu_k = \pi_k \quad \forall k \notin \{i, j\} \tag{3}$$

The neighborhood of π_i is the set $N(\pi_i)$ of the permutations obtained from π by swapping π_i with any another value. By extension, the neighborhood of π is the set $N(\pi)$ of all permutations obtained by swapping any two values:

$$N(\pi_i) = \{\pi_{i \leftrightarrow j} \mid 1 \leq j \leq n, j \neq i\} \tag{4}$$

$$N(\pi) = \bigcup_{i=1}^{n} N(\pi_i) \tag{5}$$

Most local search procedures (in particular *hill climbing*) would select, among all elements of $N(\pi)$, *the best* neighbor μ, i.e. the one minimizing the next global cost function. By doing so, they have to deal with the problem of being trapped in a local minimum. Instead, EO defines a *fitness* value λ_i for each value π_i, with the understanding that a value with a low fitness is more likely to mutate, i.e. to get swapped. We define the fitness value λ_i as the best possible improvement of the cost F when moving to a π_i's neighbor.

$$\lambda_i = \min_{\mu \in N(\pi_i)} F(\mu) - F(\pi) \tag{6}$$

A negative λ thus represents an improvement of the cost. At each iteration, the n fitness values are evaluated and ranked, with rank $k = 1$ for the worst fitness. EO-QAP will thus favor the mutation of a value which improves (decreases) the objective function. The value to mutate is chosen stochastically from a probability distribution over the rank order. This comes down to pick a value at rank k ($1 \leq k \leq n$) with a probability $P(k) = k^{-\tau}$. Let π_r be the value measured by the k^{th} fitness value, then π_r will be deemed the "culprit" and be forced to mutate. For this we need to choose a target value π_s for the swap. Several possibilities exist: one is to choose π_s randomly. Another, as in the original EO article, is to pick a random value using the same probability distribution. We propose a third possibility applying the *min-conflict* heuristic [30]: select the *best* possible value, that is, the value which minimizes the objective function of the next configuration. The algorithm then swaps π_r and π_s (thus moving to a neighbor) and iterates with this new configuration. The process stops when a some condition is reached (e.g. a time limit or a given cost is reached). The best solution found so far is then returned (see Algorithm 1).

Algorithm 1. EO-QAP: an EO procedure for QAP

```
 1: function EO-QAP
 2:     π ← a random permutation ∈ Π(n)
 3:     bestsol ← π
 4:     bestCost ← F(π)
 5:     while termination criterion is not reached do          ▷ e.g. a timeout
 6:         compute all fitness values λ of π
 7:         sort all λ in ascending order
 8:         let k be a random rank with a probability P(k)
 9:         let λ_r be the k^th fitness value (π_r must mutate)
10:         consider all possible moves from π_r and
11:         choose a value π_s minimizing the cost of the next configuration
12:         π ← π_{r↔s}                                       ▷ swap π_r and π_s
13:         if F(π) < bestCost then
14:             bestCost ← F(π)
15:             bestSol ← π
16:         end if
17:     end while
18:     return < bestSol, bestCost >
19: end function
```

Implementation Commentary and Complexity Analysis. A permutation π can be encoded by an array of n integers `sol[]` (with `sol[i]` $= \pi_i$). Line 6 computes the fitness λ_i for each value π_i. Thanks to Taillard and his amous *Robust Taboo Search* (RoTS), we know how to compute the volution of the global cost after a swap incrementally, instead of recomputing it each time from scratch – see Eqs. (1) and (2) which define $\Delta(\mu, ., .)$ in [31]. This results in an evaluation of all λ in $O(n^2)$, while a naïve algorithm is in $O(n^3)$. The simplest data structure to manage λ is just an array `fit[]` whose elements are pairs of the form <`index`,`lambda`>. Initially we have `fit[i]`=<`i`, λ_i >. Line 7 sorts the `fit[]` array on the second field (λ), in ascending order, this can be done in $O(n\ log_2(n))$. Line 8 picks a value at position k with a probability $P(k)$. Since the PDF and τ are constant along the execution of the algorithm, it is more efficient to pre-compute the n samples of $P(k), (1 \leq k \leq n)$ and store then in an array `prob[]`. To pick a random k, we can use a roulette-wheel selection on `prob[]` in $O(n)$ theoretically (but closer to $O(1)$ in practice due to the PDF). It is also possible to use a a binary search in $O(log_2(n))$ storing the cumulative PDF (`prob[k]` $= \sum_{i=1}^{k} P(i)$). Line 9: the variable to mutate is given by `sol[fit[k].index]`. Line 10 selects the other variable for the swap, as per the min-conflict heuristic; this is done in $O(n)$. However, as this variable had already been found when computing λ (Line 6), one just has to record it. To this end, the elements of the `fit[]` array are refactored as <`index`,`lambda`,`index2`>, where `index2` contains the index of the variable which minimizes the cost. The second variable for the swap is then simply given by `sol[fit[k].index2]`. The overall complexity of each iteration (i.e. of the main loop body) is thus $O(n^2)$.

3.2 Extending Extremal Optimization

Because it relies on just one parameter, EO is comparatively very simple to use (tuning many local search parameters can become very laborious). As we show in Sect. 4, the results of EO-QAP are very good on many instances of QAPLIB. However, some harder instances need a fine control of the trade-off between intensification and diversification. EO handles these two strategies with the same tool: the probability distribution $P(k) = k^{-\tau}$. Depending on the returned value, either the current path continues to be improved (with a high probability) or, in the extreme case, completely abandoned (with a low probability). Every variable has a non-zero chance of being selected and EO is not affected by local extrema. The choice of the probability distribution P has thus a great impact, also determined by its parameter τ, which must be selected by the user. Tuning τ for some hard problems (e.g. `tai40a`) turned out to be difficult. We therefore decided to *extend* EO so as to accept different probability distribution functions (PDF). The user can thus choose the most appropriate PDF. For simplicity, all proposed PDFs take a single input parameter τ, the other parameters (if any) being either constant or functionally dependent on τ. We now discuss a few interesting PDFs, and how they are influenced by the single τ parameter:

PDF definition	Usage in EO	Name
$Power(x,\tau) = x^{-\tau}$	$P(k) = Power(k,\tau)$	power law
$Expon(x,\mu) = \mu\ e^{-\mu x}$	$P(k) = Expon(k,\tau)$	exponential law
$Normal(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	$P(k) = Normal(k,1,\tau)$	normal law
$Gamma(x,k,\theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}$	$P(k) = Gamma(k,\tau,e^{\tau})$	gamma law

The EO algorithm behaves very differently, depending on the PDF and the chosen value for τ. Figure 2 shows the curves associated with these PDFs for a size $n = 40$, picking different values for τ. Clearly, with the power law, the first ranked variable has a very high probability to be selected, the probability then decreases very fast but the variables with a high rank (i.e."good" variables) are very likely to mutate (e.g. when $\tau = 0.5$). This results in less intensification, and suits some problems perfectly. If this behavior is unwanted, a larger value for τ may be used, nevertheless, this rapidly puts too strong a pressure on the besk ranked variables which, for some problems, will be selected too frequently. It can be difficult to find a good trade-off. On the other hand, the exponential law skews probabilities a little bit more in favor of lower ranked variables. This is clear from the shapes of the PDFs (see Fig. 2). The normal (Gaussian) law is also interesting because the curve decreases slowly for the very first ranks, then more rapidly and then more slowly again: the first ranked "worst" variables will then be selected with a high but comparable priority. Finally, we found the gamma law interesting because it is not strictly decreasing and can, for instance, give more priority to the second or third variable than to the first one. With this

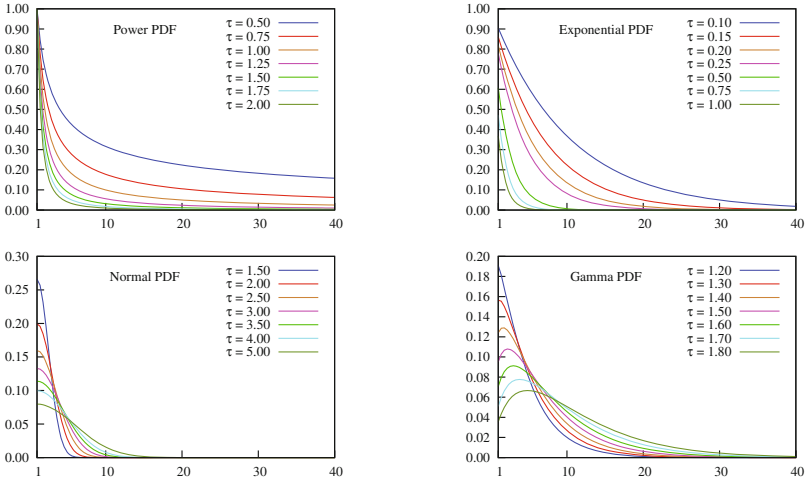


Fig. 2. Different probability distribution functions for EO

PDF we obtained good results for `tai40a` with $\tau = 1.5$. It is worth noticing that a shifted normal law can also be used as a non-stricly decreasing function, e.g. using $P(k) = Normal(k, 2, e^{\tau})$. It is worth noticing that the best PDF and τ combination is not the same when EO-QAP is run sequentially or in parallel.

Clearly, allowing different PDFs enhances the power of EO which can attack more problems efficiently. The user now has more precise control over the behavior of the algorithm, which remains simple with only two tunable parameters: the PDF and its τ value. It is even possible allow the user to provide his own customized PDF as a file of n values $P_1, \ldots P_n$. At run-time, the `prob[]` array above mentionned is populated as follows: `prob[k]` $= \frac{P_k}{\sum_{i=1}^{n} P_i}$ (each value being divided by the sum of all values to ensure the whole PDF $= 1$).

4 Experimental Evaluation

In this section we present experimental results on the entire QAPLIB test set. To do this, we developed an X10 implementation of EO-QAP.[1] Because EO is a stochastic procedure, we ran each problem 10 times and averaged the results. The number of possible experiments is very high: with different PDFs and τ values, varying the timeout, testing sequential or parallel runs, with different topologies and communication strategies, etc. We thus adopted a 3-stage protocol:

1. Attempt to solve all QAPLIB problems (134 instances) with a basic version of EO-QAP (i.e. without any tuning) and a very short timeout (in order to be able to try 10 runs for each of the 134 instances). All problems for which the Best Known Solution (BKS) was reached for every execution are definitively

[1] Source code and instances are available from http://cri-hpc1.univ-paris1.fr/qap/.

classified as *solved*. The others (solved less than 10 times) form the test set of the next stage.
2. To attack the remaining problem we ran EO-QAP (with same parameters and timeout) in independent parallel multi-walks, without communication, on 32 cores of a single machine. As previously, we collected the fully solved instances which need no longer be considered. The remaining instances form the input set for the next stage.
3. The remaining problems are the hardest ones: for these, we used a cooperative parallel version of EO-QAP on 128 cores, tuning the PDF and τ value, and using a larger timeout of 10 min.

4.1 Stage 1: Sequential Execution

In this first stage, the input test set consists of the 134 QAPLIB instances, which are run sequentially on an AMD Opteron 6376 clocked at 2.3 GHz, using a single core. This is the basic version of EO-QAP with the original power-law PDF and default value for τ (see Sect. 2.2). For each problem we report the BKS (which is sometimes the optimum), the number of times the BKS is reached (#BKS), i.e. the number of times the problem is solved, the average execution time (in seconds) and the Average Percentage Deviation (APD) which is the average of the 10 relative deviation *percentages* computed as follows: $100\frac{F(sol)-BKS}{BKS}$. We use a short timeout of 5 min. Even if the solver stops as soon as the BKS is reached a limited timeout is needed to be able to run the 134 instances 10 times.

Table 1 presents the whole results. Surprisingly, even with this straightforward and suboptimal setting, EO-QAP performs quite well: more than 50 % of the instances get totally solved. More precisely: 68 instances are fully solved. On average, the 41 others are solved 4.6 times (in orange). The average APD for the 66 instances not fully solved is about 2.2 %.

4.2 Stage 2: Independent Parallelism

In this stage, we ran the EO-QAP algorithm in parallel without communication with the same settings as in the first stage (default PDF, default τ, timeout 5 min). The machine was the same: a quad AMD Opteron 6376 clocked at 2.3 GHz, but using 32 cores. These parameters make it possible to assess what improvement we can easily obtain by means of parallel execution. Indeed, this from of parallelism (sometimes called *embarrassingly parallelism*) works by performing multiple independent walks to explore the search space. Each worker blindly explores a region of the search space, looking for a solution. The process ends as soon as any solver reaches a solution. Since all EO solvers start from a random point, we can expect that they will all visit different regions of the search space (i.e. ensuring a form of diversification), thus increasing the chance to find a solution. Such a parallelization of an algorithm is easy to implement and often behaves very well (see Sect. 2.3).

Table 1. Sequential execution (power-law, default τ, timeout = 300 s)

Problem	BKS	APD	#BKS	time(s)
bur26a	5426670	0.034	6	122.817
bur26b	3817852	0.101	5	151.415
bur26c	5426795	0.126	5	161.905
bur26d	3821225	0.120	2	251.497
bur26e	5386879	0.072	7	90.776
bur26f	3782044	0.142	6	120.325
bur26g	10117172	0.202	5	150.330
bur26h	7098658	0.245	6	120.161
chr12a	9552	0.000	10	0.011
chr12b	9742	0.000	10	0.005
chr12c	11156	0.000	10	0.178
chr15a	9896	0.000	10	2.206
chr15b	7990	0.000	10	0.143
chr15c	9504	0.000	10	1.043
chr18a	11098	0.000	10	1.400
chr18b	1534	0.000	10	0.041
chr20a	2192	0.000	10	2.951
chr20b	2298	0.000	10	3.568
chr20c	14142	0.000	10	0.632
chr22a	6156	0.000	10	2.234
chr22b	6194	0.000	10	2.784
chr25a	3796	0.000	10	6.803
els19	17212548	20.902	2	240.000
esc16a	68	0.000	10	0.000
esc16b	292	0.000	10	0.000
esc16c	160	0.000	10	0.000
esc16d	16	0.000	10	0.000
esc16e	28	0.000	10	0.000
esc16f	0	0.000	10	0.000
esc16g	26	0.000	10	0.000
esc16h	996	0.000	10	0.000
esc16i	14	0.000	10	0.000
esc16j	8	0.000	10	0.000
esc32a	130	0.000	10	0.292
esc32b	168	0.000	10	0.051
esc32c	642	0.000	10	0.001
esc32d	200	0.000	10	6.634
esc32e	2	0.000	10	0.000
esc32g	6	0.000	10	0.000
esc32h	438	0.000	10	18.223
esc64a	116	0.000	10	0.009
esc128	64	0.625	8	60.175
had12	1652	0.194	6	120.000
had14	2724	0.220	7	90.000
had16	3720	0.032	4	180.000
had18	5358	0.134	4	180.033
had20	6922	0.150	6	122.549
kra30a	88900	0.983	4	180.439
kra30b	91420	0.213	5	189.252
kra32	88700	0.826	5	150.539
lipa20a	3683	0.000	10	0.069
lipa20b	27076	0.000	10	0.005
lipa30a	13178	0.000	10	0.742
lipa30b	151426	0.000	10	0.037
lipa40a	31538	0.000	10	1.962
lipa40b	476581	0.000	10	0.067
lipa50a	62093	0.000	10	4.245
lipa50b	1210244	0.000	10	0.113
lipa60a	107218	0.000	10	18.825
lipa60b	2520135	0.000	10	2.282
lipa70a	169755	0.000	10	57.737
lipa70b	4603200	0.000	10	8.288
lipa80a	253195	0.047	9	158.337
lipa80b	7763962	0.000	10	18.203
lipa90a	360630	0.221	5	256.062
lipa90b	12490441	0.000	10	20.193
nug12	578	0.000	10	0.012
nug14	1014	0.000	10	0.341
nug15	1150	0.000	10	0.293
nug16a	1610	0.373	5	150.025
nug16b	1240	0.000	10	0.008
nug17	1732	0.000	10	17.504
nug18	1930	0.000	10	1.287
nug20	2570	0.000	10	0.482
nug21	2438	0.000	10	28.066
nug22	3596	0.501	5	158.960
nug24	3488	0.034	9	87.571
nug25	3744	0.000	10	0.591
nug27	5234	0.474	6	130.763
nug28	5166	0.031	9	101.182
nug30	6124	0.157	6	122.573
rou12	235528	0.000	10	0.013
rou15	354210	0.000	10	0.035
rou20	725522	0.000	10	1.668
scr12	31410	0.000	10	0.006
scr15	51140	0.000	10	0.023
scr20	110030	0.000	10	0.334
sko42	15812	0.197	3	221.158
sko49	23386	0.073	2	250.514
sko56	34458	0.172	3	236.856
sko64	48498	0.302	1	277.832
sko72	66256	0.503	1	287.034
sko81	90998	0.449	0	300.000
sko90	115534	0.675	0	300.000
sko100a	152002	0.612	0	300.000
sko100b	153890	0.264	0	300.000
sko100c	147862	0.760	0	300.000
sko100d	149576	0.583	0	300.000
sko100e	149150	0.687	0	300.000
sko100f	149036	0.652	0	300.000
ste36a	9526	0.426	7	139.233
ste36b	15852	2.976	2	253.286
ste36c	8239110	0.426	2	264.393
tai12a	224416	0.000	10	0.011
tai15a	388214	0.000	10	0.089
tai17a	491812	0.000	10	0.292
tai20a	703482	0.000	10	2.637
tai25a	1167256	0.000	10	6.330
tai30a	1818146	0.000	10	9.589
tai35a	2422002	0.000	10	42.399
tai40a	3139370	0.215	0	300.000
tai50a	4938796	0.511	0	300.000
tai60a	7205962	0.537	0	300.000
tai80a	13499184	0.750	0	300.000
tai100a	21052466	0.579	0	300.000
tai12b	39464925	7.995	2	240.000
tai15b	51765268	0.071	5	151.353
tai20b	122455319	21.993	1	270.000
tai25b	344355646	12.805	1	270.207
tai30b	637117113	15.479	1	270.022
tai35b	283315445	7.703	0	300.000
tai40b	637250948	10.085	0	300.000
tai50b	458821517	6.803	0	300.000
tai60b	608215054	6.559	0	300.000
tai80b	818415043	5.402	0	300.000
tai100b	1185996137	4.750	0	300.000
tai150b	498896643	2.686	0	300.000
tai64c	1855928	0.450	0	300.000
tai256c	44759294	0.431	0	300.000
tho30	149936	0.184	9	87.784
tho40	240516	0.147	2	241.778
tho150	8133398	1.196	0	300.000
wil50	48816	0.153	0	300.000
wil100	273038	0.409	0	300.000

Table 2. Independent parallelism on 32 cores (timeout = 300 s)

Problem	BKS	APD	#BKS	time(s)
bur26a	5426670	0.000	10	0.027
bur26b	3817852	0.000	10	0.021
bur26c	5426795	0.000	10	0.009
bur26d	3821225	0.000	10	9.311
bur26e	5386879	0.000	10	0.010
bur26f	3782044	0.000	10	0.009
bur26g	10117172	0.000	10	0.006
bur26h	7098658	0.000	10	0.010
els19	17212548	0.421	9	30.007
esc128	64	0.000	10	0.036
had12	1652	0.000	10	0.000
had14	2724	0.000	10	0.000
had16	3720	0.000	10	0.000
had18	5358	0.000	10	0.001
had20	6922	0.000	10	0.001
kra30a	88900	0.134	9	30.544
kra30b	91420	0.000	10	2.485
kra32	88700	0.000	10	0.195
lipa80a	253195	0.000	10	15.001
lipa90a	360630	0.000	10	32.361
nug16a	1610	0.000	10	0.001
nug22	3596	0.000	10	0.002
nug24	3488	0.000	10	0.004
nug27	5234	0.000	10	0.006
nug28	5166	0.000	10	0.059
nug30	6124	0.000	10	0.268
sko42	15812	0.000	10	1.138
sko49	23386	0.000	10	40.118
sko56	34458	0.001	9	71.411
sko64	48498	0.005	8	109.435
sko72	66256	0.041	3	236.308
sko81	90998	0.044	1	271.685
sko90	115534	0.125	1	288.550

Problem	BKS	APD	#BKS	time(s)
sko100a	152002	0.117	0	300.000
sko100b	153890	0.112	0	300.000
sko100c	147862	0.056	0	300.000
sko100d	149576	0.133	0	300.000
sko100e	149150	0.059	0	300.000
sko100f	149036	0.144	0	300.000
ste36a	9526	0.000	10	1.148
ste36b	15852	0.000	10	2.035
ste36c	8239110	0.000	10	5.148
tai40a	3139370	0.074	0	300.000
tai50a	4938796	0.286	0	300.000
tai60a	7205962	0.302	0	300.000
tai80a	13499184	0.497	0	300.000
tai100a	21052466	0.419	0	300.000
tai12b	39464925	0.000	10	0.001
tai15b	51765268	0.000	10	0.001
tai20b	122455319	0.045	9	30.003
tai25b	344355646	0.074	8	94.831
tai30b	637117113	0.638	6	122.746
tai35b	283315445	0.364	3	229.160
tai40b	637250948	0.339	7	91.070
tai50b	458821517	1.222	0	300.000
tai60b	608215054	1.318	0	300.000
tai80b	818415043	2.012	0	300.000
tai100b	1185996137	0.900	0	300.000
tai150b	498896643	1.546	0	300.000
tai64c	1855928	0.012	8	60.004
tai256c	44759294	0.294	0	300.000
tho30	149936	0.000	10	0.235
tho40	240516	0.002	8	77.261
tho150	8133398	0.436	0	300.000
wil50	48816	0.000	10	27.545
wil100	273038	0.111	0	300.000

The results of this experiment are summarized in Table 2. This form of parallelism brings a significant improvement in performance and reach. Exactly half of the 66 problem instances now become fully solved (the average time being $3.2s$). However, among the 33 remaining ones, 19 remain never solved and, on average, the remaining 14 get solved 6 out of 10 times. Moreover, the average APD over the not fully solved 33 is 0.372 %. This contrasts with the previous situation (2.2 %), which indicates a significant improvement in the quality of solutions: even when the optimum is not reached, the solution which was found is close.

4.3 Stage 3: Cooperative Parallelism

In this final experiment, we attacked the 33 hardest instances with parallelism and cooperation. This was simplified thanks to the CPLS framework which provides the necessary abstraction layers and already handles the communication (see Sect. 2.3). The sequential EO-QAP needed a very simple adaptation: every R iterations it has to send its current configuration to the Elite Pool and, every U iterations, it retrieves a configuration from the pool, which it nondeterministically[2]

[2] With a probability *pAdopt*.

adopts it if it is better than the current one. The CPLS system already provides library functions for all these operations. The resulting solver is then composed of several EO-QAP instances running in parallel, which cooperate by communicating in order to converge faster on a solution. As per [7], the CPLS parameters which control the cooperation are as follows:

- *Team Size* (NPT): we tested various configurations and defined $NPT = 16$. There are thus 8 teams composed of 16 explorer nodes each running the EO-QAP procedure. This is constant for all problems.
- *Report Interval* (R): we manually tuned it (starting from the average number of iterations collected during the previous stage divided by 10).
- *Update Interval* (U): we experimented different ratio and retained $U/R = 2$.
- *Elite Pool* (EP): its size is fixed to 4 for all problems.
- *pAdopt*: is set to 1. An EO-QAP instance receiving a better configuration than its current one always switches to this new one.

This experiment has been carried out on a cluster of 16 machines, each with 4×16-core AMD Opteron 6376 CPUs running at 2.3 GHz and 128 GB of RAM. The nodes are interconnected with InfiniBand FDR 4× (i.e. 56 GBPS.) We had access to 4 nodes and used up to 32 cores per node, i.e. 128 cores. We stay with a timeout of 5 min. Finally, we tested deeply two PDFs (power and exponential) and their τ value and retained the best combination for each problem instance (we could not yet test the Normal law and the Gamma law, while efficient in sequential seems, not well suited for parallelism or else using a different τ value).

Table 3 presents the results obtained on the hardest instances. The table also reports the average number of iterations, the report and update interval (R and U), the number of times the winning algorithm has adopted an elite configuration, the PDF and τ value used. In this last stage, 15 new problems become fully solved. Moreover the average time to solve them is only 24.5 s. Only 8 remain unsolved. On average, the remaining 10 get solved 5 out of 10 times. Moreover, the average APD over the 18 not fully solved is only 0.25 %. Even when the optimum is not reached the returned solution is close to this optimum.

The table shows that an efficient execution corresponds to a limited number of "adoptions" of an elite configuration (less than 5 changes). This is directly correlated to the value of R and U. There are some exceptions like `tai25b` and `tho40` which are both fully solved. We plan to analyze in details these both situations (e.g. varying R and U).

Regarding the power and exponential PDF, there is no winner. It is worth noticing that sometimes the difference is huge. For instance, `sko90` is solved 9 times with the power law but only 2 times with the exponential law. The reverse occurs for `tai`XX`a` for which the exponential law performs much better.

The performance of the cooperative parallel EO-QAP, is on par with the best competing solutions, while retaining a much simpler internal structure [32]. Due to space limitations, we do not develop this further.

Table 3. Cooperative parallelism on 128 cores (timeout = 300 s)

Problem	BKS	APD	#BKS	time(s)	#iters	R	U	#adopt	PDF	τ
els19	17212548	0.000	10	0.003	23	10	20	0.2	pow	1.70
kra30a	88900	0.000	10	0.026	599	100	200	2.6	pow	0.20
sko56	34458	0.000	10	4.776	35658	7000	14000	2.5	pow	0.60
sko64	48498	0.000	10	4.822	26494	7000	14000	1.5	pow	0.60
sko72	66256	0.000	10	13.442	57956	15000	30000	1.4	pow	0.80
sko81	90998	0.008	7	118.905	399748	20000	40000	9.4	pow	1.00
sko90	115534	0.000	10	92.951	253547	25000	50000	5.0	pow	0.60
sko100a	152002	0.012	5	224.206	492441	50000	100000	4.2	pow	1.00
sko100b	153890	0.001	8	146.679	322560	50000	100000	2.6	pow	1.00
sko100c	147862	0.000	10	145.032	319871	50000	100000	2.4	pow	1.00
sko100d	149576	0.014	6	200.948	442626	50000	100000	3.6	pow	1.00
sko100e	149150	0.000	10	103.370	228094	50000	100000	1.6	pow	1.00
sko100f	149036	0.011	4	245.929	542031	50000	100000	4.8	pow	1.00
tai40a	3139370	0.022	7	171.113	2701406	350000	700000	3.4	exp	0.18
tai50a	4938796	0.026	5	208.882	1997696	350000	700000	2.4	exp	0.16
tai60a	7205962	0.132	2	285.329	1833023	400000	800000	1.9	exp	0.17
tai80a	13499184	0.385	0	300.000	1037282	400000	800000	1.0	exp	0.16
tai100a	21052466	0.297	0	300.000	657489	100000	200000	3.0	exp	0.13
tai20b	122455319	0.000	10	0.001	69	20	40	0.8	exp	0.38
tai25b	344355646	0.000	10	0.600	23331	400	800	17.0	pow	1.40
tai30b	637117113	0.000	10	0.121	3360	500	1000	3.0	pow	0.20
tai35b	283315445	0.000	10	0.737	15659	500	1000	14.2	pow	0.80
tai40b	637250948	0.000	10	0.061	973	500	1000	0.4	exp	0.12
tai50b	458821517	0.214	2	266.446	2571166	250000	500000	4.5	exp	0.03
tai60b	608215054	0.205	3	256.836	1676681	250000	500000	2.6	pow	0.40
tai80b	818415043	1.192	0	300.000	1035635	45000	90000	8.8	pow	1.05
tai100b	1185996137	0.465	0	300.000	658761	50000	100000	5.5	pow	1.02
tai150b	498896643	1.088	0	300.000	281965	70000	140000	1.5	pow	0.94
tai64c	1855928	0.000	10	0.010	27	6	12	0.3	exp	0.04
tai256c	44759294	0.263	0	300.000	81624	10000	20000	1.3	exp	0.04
tho40	240516	0.000	10	1.243	20091	22000	44000	0.2	pow	1.00
tho150	8133398	0.144	0	300.000	281711	50000	100000	1.7	pow	1.00
wil100	273038	0.061	0	300.000	662233	50000	100000	5.4	exp	0.09

5 Conclusion and Further Work

We have proposed EO-QAP: an Extremal Optimization (EO) procedure for QAP. The basic sequential version of EO-QAP, while simple behaves rather well on several instances of QAPLIB. To attack hardest instances we first proposed a simple extension to the original EO procedure allowing for different probability distribution functions (PDF). The user can select the most adequate PDF depending on the degree of intensification wanted. Moreover, we resorted to cooperative parallelism using a framework written in the X10 parallel language, which provides the user with a fine degree of control of the intensification and diversification for the search. The cooperative version of EO-QAP displays very good results: using 128 cores, 116 instances of QAPLIB are systematically solved at each execution, 10 instances are solved half the time and only 8 instances remain unsolved (but the obtained solution is near to the optimum).

We now plan to attack other known hard instances. Future work includes the study of a default parameter for the other PDFs (e.g. exponential) and how to take into account the hardness of the problem to define this value (e.g. in terms of the landscape ruggedness of the instance to solve). Moreover, we plan

to explore portfolio approaches, i.e. ones which combine multiple solvers as well as experiment with techniques for parameter auto-tuning. This experimentation entails a deep analysis of the parallel performance behavior.

Acknowledgments. The authors wish to acknowledge Stefan Boettcher (Emory University) for his explanations about the Extremal Optimization method. The experimentation used the cluster of the University of Évora, which was partly funded by grants ALENT-07-0262-FEDER-001872 and ALENT-07-0262-FEDER-001876.

References

1. Koopmans, T.C., Beckmann, M.: Assignment problems and the location of economic activities. Econometrica **25**(1), 53–76 (1957)
2. Commander, C.W.: A survey of the quadratic assignment problem, with applications. Morehead Electron. J. Appl. Math. **4**, 1–15 (2005). MATH-2005-01
3. Bhati, R.K., Rasool, A.: Quadratic assignment problem and its relevance to the real world: a survey. Int. J. Comput. Appl. **96**(9), 42–47 (2014)
4. Sahni, S., Gonzalez, T.: P-complete approximation problems. J. ACM **23**(3), 555–565 (1976)
5. Boettcher, S., Percus, A.: Nature's way of optimizing. Artif. Intell. **119**(1–2), 275–286 (2000)
6. Randall, M., Lewis, A.: Intensification strategies for extremal optimisation. In: Deb, K., et al. (eds.) SEAL 2010. LNCS, vol. 6457, pp. 115–124. Springer, Heidelberg (2010)
7. Munera, D., Diaz, D., Abreu, S., Codognet, P.: A parametric framework for cooperative parallel local search. In: Blum, C., Ochoa, G. (eds.) EvoCOP 2014. LNCS, vol. 8600, pp. 13–24. Springer, Heidelberg (2014)
8. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., Von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. In: SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, pp. 519–538. ACM, San Diego (2005)
9. Saraswat, V., Tardieu, O., Grove, D., Cunningham, D., Takeuchi, M., Herta, B.:A Brief Introduction to X10 (for the High Performance Programmer). Technical report (2012)
10. Burkard, R.E.: Quadratic assignment problems. In: Pardalos, P.M., Du, D.Z., Graham, R.L. (eds.) Handbook of Combinatorial Optimization, 2nd edn, pp. 2741–2814. Springer, New York (2013)
11. Loiola, E.M., de Abreu, N.M.M., Netto, P.O.B., Hahn, P., Querido, T.M.: A survey for the quadratic assignment problem. Eur. J. Oper. Res. **176**(2), 657–690 (2007)
12. Zaied, A.N.H., Shawky, LAE-f: A survey of quadratic assignment problems. Int. J. Comput. Appl. **101**(6), 28–36 (2014)
13. Said, G., Mahmoud, A.M., El-Horbaty, E.S.M.: A comparative study of metaheuristic algorithms for solving quadratic assignment problem. Int. J. Adv. Comput. Sci. Appl. (IJACSA) **5**(1), 1–6 (2014)
14. Boettcher, S., Percus, A.G.: Extremal optimization: an evolutionary local-search algorithm. In: Bhargava, H.K., Ye, N. (eds.) Computational Modeling and Problem Solving in the Networked World, vol. 21, pp. 61–77. Springer, US (2003)

15. Boettcher, S.: Extremal optimization. In: Hartmann, A.K., Rieger, H. (eds.) New Optimization Algorithms to Physics, pp. 227–251. Wiley-VCH Verlag, Berlin (2004)
16. Bak, P., Tang, C., Wiesenfeld, K.: Self-organized crtiticality: an explenation of 1/f noise. Phys. Rev. Lett. **59**(4), 381–384 (1987)
17. Bak, P.: How Nature Works: The Science of Self-organized Criticality, 1st edn. Copernicus (Springer), New York (1996)
18. Bak, P., Sneppen, K.: Punctuated equilibrium and criticality in a simple model of evolution. Phys. Rev. Lett. **71**(24), 4083–4086 (1993)
19. Boettcher, S., Percus, A.G.: Optimization with extremal dynamics. Phys. Rev. Lett. **86**(23), 5211–5214 (2001)
20. De Sousa, F.L., Ramos, F.M.: Function optimization using extremal dynamics. In: International Conference on Inverse Problems in Engineering Rio de Janeiro, Brazil (2002)
21. De Sousa, F.L., Vlassov, V., Ramos, F.M.: Generalized extremal optimization for solving complex optimal design problems. In: International Conference on Genetic and Evolutionary Computation, pp. 375–376 (2003)
22. Zhou, T., Bai, W.J., Cheng, L.J., Wang, B.H.: Continuous extremal optimization for Lennard-Jones clusters. Phys. Rev. E **72**(1), 016702 (2005)
23. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience, New York (2005)
24. Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: recent advances and new trends. Int. Trans. Oper. Res. **20**(1), 1–48 (2013)
25. Diaz, D., Abreu, S., Codognet, P.: Parallel constraint-based local search on the Cell/BE multicore architecture. In: Essaaidi, M., Malgeri, M., Badica, C. (eds.) Intelligent Distributed Computing IV. SCI, vol. 315, pp. 265–274. Springer, Heidelberg (2010)
26. Verhoeven, M., Aarts, E.: Parallel local search. J. heuristics **1**(1), 43–65 (1995)
27. Caniou, Y., Codognet, P., Richoux, F., Diaz, D., Abreu, S.: Large-scale parallelism for constraint-based local search: the costas array case study. Constraints **20**(1), 1–27 (2014)
28. Toulouse, M., Crainic, T., Sansó, B.: Systemic behavior of cooperative search algorithms. Parallel Comput. **30**, 57–79 (2004)
29. Munera, D., Diaz, D., Abreu, S., Codognet, P.: Flexible cooperation in parallel local search. In: Symposium on Applied Computing (SAC), pp. 1360–1361. ACM Press, New York (2014)
30. Minton, S., Philips, A., Johnston, M.D., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint-satisfaction and scheduling problems. J. Artif. Intell. Res. **58**, 161–205 (1993)
31. Taillard, É.D.: Comparison of iterative searches for the quadratic assignment problem. Location Sci. **3**(2), 87–105 (1995)
32. James, T., Rego, C., Glover, F.: A cooperative parallel tabu search algorithm for the quadratic assignment problem. Eur. J. Oper. Res. **195**, 810–826 (2009)

Author Index